Assignment 5

Erlang

Due Date: Monday, April 10th 2017, 11:59pm

Directions

We will take some points off for: code with the wrong type or wrong name, duplicate code, code with extra unnecessary cases, or code that is excessively hard to follow. You should always assume that the inputs given to each function will be well-typed, thus your code should not have extra cases for inputs that are not of the proper type. (Assume that any human inputs are error checked before they reach your code. Avoid duplicating code by using helping functions, library functions (when not prohibited in the problems). It is a good idea to check your code for these problems before submitting. Be sure to test your programs on Eustis to ensure they work on a Unix system.

This is a group assignment. Make a group of 3-4 in WebCourses and use that to turn in the assignment.

Deliverables

For this assignment, turn in the following files:

- concat.erl
- substaddr.erl
- count_matching.erl
- power.erl
- var.erl
- logger.erl

For problems that require an English answer, put your solution in Solution.pdf. For coding problems, put your solution in the appropriate Erlang source file (named with ".erl" suffix).

Testing Code

The testing code for Erlang is given in the testing.erl file supplied with this homework's zip file. Note that to run our tests, all your modules involved must first be compiled. Then to execute our tests in a module named m_tests, type m_tests:main(). to the Erlang prompt.

Problem 1 (10 pts)

In Erlang, without using any functions from Erlang's `lists` module, write a function `concat/1`, whose type is given by the following.

```
-spec concat(Lists :: [[T]]) -> [T].
```

The function concat takes a list of lists of elements (of some type T) and returns a list of the elements (of type T) formed by concatenating the inner lists together in order. The following are examples written using the Erlang testing module. You can use the built-in ++ in your code, or any other helping functions. Run these tests by compiling your file and the testing file, and then running

```
concat_tests:main().
```

Problem 2 (20 pts)

This problem is about “sales data records.” There are two records types that are involved in the type `salesdata()`, which are defined in the file `salesdata.hrl` (note that file extension carefully, it’s hrl with an “h”) shown below and included with the testing files.

```
-record(store, {address :: string(), amounts :: [integer()]}).
-record(group, {gname :: string(), members :: [salesdata:salesdata()]}).
```

The type salesdata() itself is defined by the following, which says that a sales data values is either a store record or a group record.

```
-module(salesdata).
-include(“salesdata.hrl”).
-export_type([salesdata/0]).

-type salesdata() :: #store{} | #group{}.
```

Your task is to write, in Erlang, a function

```
-spec substaddr(salesdata:salesdata(), string(), string()) -> salesdata:salesdata().
```

that takes a sales data record SD, two strings New and Old, and returns a sales data record that is just like SD except that all store records in SD whose address field’s value is Old in SD are changed to New in the result.

Your solution must follow the grammar; we will take points off for not following the grammar! In particular, you should never call substaddr with a list argument; be sure to use a helping function for lists instead.

To run our tests, run `substaddr_tests:main()`. Note how the testing file does **`-include`**`(“salesdata.hrl”)`; your solution file (`substaddr.erl`) must also include this directive if you want to use the Erlang record syntax.

To be clear, your solution should go in a file `substaddr.erl`, as the tests import `substaddr/3` from that file. Thus to use the record syntax and typing, your solution file `substaddr.erl` should start out as follows. (Note that you cannot import the type `salesdata/0` from `salesdata.erl`, as Erlang does not currently permit type imports.)

```
-module(substaddr).
-export([substaddr/3]).
-include(“salesdata.hrl”).
-import(salesdata, [store/2, group/2]).

-spec substaddr(salesdata:salesdata(), string(), string()) -> salesdata:salesdata().
```

Problem 3 (15 pts)

In Erlang, without using the lists module or any list comprehensions, write a tail-recursive function

```
-spec count_matching (fun((T) -> boolean()), list(T)) -> non_neg_integer().
```

that takes a predicate, Pred, and a list, Lst, and returns the number of elements in Lst that satisfy Pred.

Note that your code must use tail recursion and is not allowed to use any functions from the list module or list comprehensions.

Hint: In Erlang, you can call function closure, say Pred on an argument X using the syntax Pred(X). However, you cannot call an arbitrary function in the guard of an if expression, as the syntax only allows guard sequences there. Instead of using an if-expression, try using a **case** expression instead.

Problem 4 (10 pts)

In Erlang, write a stateless server in a module named power. This server responds to messages of the form $\{\mathrm{Pid}, \mathrm{power}, \mathrm{N}, \mathrm{M}\}$, where Pid is the sender's process id, N, and M are non-negative integers. When the server receives such a message, it responds by sending a message of the form $\{\mathrm{answer}, \mathrm{Res}\}$ to Pid, where Res is N^{M}, that is N raised to the Mth power. In your solution you can use the library function $\mathrm{math{:}\,pow}$, which is defined so that $\mathrm{math{:}\,pow(N, M)}$ returns N^{M}. Find the tests in the testing zip file. To run our tests, run `power_tests:main()`.

Problem 5 (15 pts)

Write, in Erlang, a module `var`, whose `start/1` function returns the process id of a server. The server's state contains a value, which is initially the value given to `start/1` as its argument. The server responds to the following messages:

- $\{\mathrm{assign}, \mathrm{NewVal}\}$, which makes the server continue with NewVal as its new value.
- $\{\mathrm{Pid}, \mathrm{fetch}\}$, which causes the serve to send the message $\{\mathrm{value}, \mathrm{Value}\}$ to Pid, where Value is the server's current value. The server's value is unchanged by this message.

Do *not* use the ets, dets, or mnesia modules in your solution. (This is to keep the problem simple.) Instead, store the value in an argument to the server's loop, as we have shown in class.

Problem 6 (20 pts)

In an Erlang module named logger, write a function start/0, which creates a log server and returns in process id. A server created by `logger:start()` keeps track of a list of log entries. The entries are simply Erlang values (of any type). The server responds to two types of messages:

- $\{\mathrm{Pid}, \mathrm{log}, \mathrm{Entry}\}$, where Pid is the sender's process id, and Entry is a value. This message causes the server to remember Entry in its list. The server responds by sending to Pid a message of the form $\{\mathrm{SPid}, \mathrm{log\_is}, \mathrm{Entries}\}$, where SPid is the server's process id.
- $\{\mathrm{Pid}, \mathrm{fetch}\}$, where Pid is the sender's process id. The server responds by sending a message to Pid of the form $\{\mathrm{SPid}, \mathrm{log\_is}, \mathrm{Entries}\}$, where SPid is the server's process id, and Entries is a list of all the entries that have been previously received by the log server (SPid), in the order in which they were received (oldest first).

To run our tests, run `logger_tests:main()`.