2012

DIT027 – Exercises on Sequential Erlang

The exercises are supposed to be done individually. You are allowed to discuss the problems with other students and share **ideas** with other students, but do **not** share code! There are 5 problems to solve. The problem have a pre-defined order, and in some cases the solution to an earlier (simpler) problem is needed to solve a later problem.

Don't forget to write comments in your code to make your solutions clearer!

Make sure that you follow ALL the instructions closely!

Deadline for submission: 30/9 23:59 (Submission in GUL only!)

Grading

If you submit working, well commented and reasonably looking solutions to some of the problems listed below you are going to get extra credits on the regular exam. 3 solved exercises gives 1 credit, 4 solved exercises gives 2 credits and 5 solved exercises gives 3 credits. Note: All problems have a well defined correct solution, and you should not have too much trouble verifying that your solution work. (Remember to try (a lot) more examples than just the ones listed in the problem description, they are just examples, your implementation should work for more general input.)

GOOD LUCK!

0. (Not really a problem) Download the Erlang file named **seq_erlang.erl** from GUL, write the solutions to all problems in this single file/module. Fill out your name, email-address, and personal number in this file.

Don't change any of the predefined code in this file. (Except for the variables named **_Variable**; these you really should change once you start using them!!) All functions mentioned in this assignment are exported in seq_erlang.erl. Empty functions are defined in there for you as well, you need to fill in the rest! You are allowed to define more functions if you need them, but you should not export those. (Of course you are allowed to temporarily export them for testing, but you must clean up your code before submission!)

Note: You are of course not allowed to use any of the library functions (lists:reverse, etc.) while solving these assignments. BIFs, with the exception of element/2 are, however, allowed.

1. **Basic functions –** In this problem you are supposed to write a few basic Erlang functions.

   A. Write a function **sum/1**, which given a list of integers should return the sum of the integers. The function should not crash for the empty list.

   Ex: sum([1, 4, 9]) → 14, sum([3]) → 3

   B. Write a function **reverse/1**, which given a list returns that list in reverse. Again, the function should not crash for the empty list.

   Ex: reverse([4]) → [4], reverse([a, x, y, k]) → [k, y, x, a]

   C. Write the function **seq/2** that given two integers N and M, returns the list of numbers between N and M (inclusive). If N > M the empty list is returned.

   Ex: seq(3, 6) → [3, 4, 5, 6], seq(1, 1) → [1], seq(3, -2) → [].

2. **Duplicate list elements –** Implement a function **list_duplicate/1**, which takes a list and returns a list containing all elements duplicated.

   Examples:

   - **list_duplicate([1, 2, 3]) → [1, 1, 2, 2, 3, 3]**
   - **list_duplicate([foo, monkey]) → [foo, foo, monkey, monkey]**
   - **list_duplicate([]) → []**

3. **Compress a list –** implement the so called 'run-length encoding' of a list (**compress/1**). It is a simple compression scheme for data in list format. Consecutive, equal, elements are represented by the element and the number of elements. Since nothing is gained by doing this for 'singleton' elements these are not compressed.

   Examples:

   - **compress([2, 2, 2, 1, 5, 5]) → [{2, 3}, 1, {5, 2}]**
   - **compress([1, 2, 3]) → [1, 2, 3]**
   - **compress([a, a, b, a, a, a]) → [{a, 2}, b, {a, 3}]**

4. **Decompress a list –** Compression is useless without decompression, implement the reverse of the function in question 3, **decompress/1**.

   - **decompress({2, 3}, 1, {5, 2}) → [2, 2, 2, 1, 5, 5]**
   - **decompress([1, 2, 3]) → [1, 2, 3]**
   - **decompress([{a, 2}, b, {a, 3}]) → [a, a, b, a, a, a]**

5. **Simple database –** Write functions that create, read, write, delete, match, merge and destroy a simple database. Since you are supposed to use lists and records as your main data structures and not store things permanently on disk, **destroy/1** is trivial and given to you. You may have (partly) solved this exercise during Francesco's visit, you are perfectly allowed to re-use that solution here. The record is given in the skeleton-module.

   The interface is the following:

   - new() → Db.
   - destroy(Db) → ok.
   - write(Key, Elem, Db) → NewDb. (old Element, if existing, overwritten)
   - delete(Key, Db) → NewDb.
   - read(Key, Db) → {ok, Elem} | {error, instance}.
   - match(Elem, Db) → [Key1, …, KeyN]. (Order of the keys undefined!)
   - merge(Db1, Db2) → NewDb. (Keys from Db1 takes precedence!)

   Note, when testing your code, remember that Erlang variables are single-assignment! The following is an example interaction with the database:

```
1> Db1 = seq_erlang:new().
[]
2> Db2 = seq_erlang:write(hans, gothenburg, Db1).
[{db_entry,hans,gothenburg}]
3> Db3 = seq_erlang:write(francesco, london, Db2).
[{db_entry,francesco,london},{db_entry,hans,gothenburg}]
4> seq_erlang:read(hans, Db3).
{ok,gothenburg}
5> Db4 = seq_erlang:write(thomas, gothenburg, seq_erlang:new()).
[{db_entry,thomas,gothenburg}]
6> Db5 = seq_erlang:write(hans, stockholm, Db4).
[{db_entry,hans,stockholm},{db_entry,thomas,gothenburg}]
7> seq_erlang:read(francesco, Db5).
{error,instance}
8> Db6 = seq_erlang:merge(Db3, Db5).
[{db_entry,thomas,gothenburg},{db_entry,hans,gothenburg},
{db_entry,francesco,london}]
9> seq_erlang:match(gothenburg, Db6).
[thomas,hans]
10> Db7 = seq_erlang:delete(hans, Db6).
[{db_entry,thomas,gothenburg},{db_entry,francesco,london}]
```

```
11> seq_erlang:destroy(Db7).
ok
```

Remember that the command **f().** flushes all bound variables when experimenting in the shell. Even better, write some code in a test module to test your implementation thoroughly. Also, **do not use** the tuple-representation of the db_entry-record directly!

GOOD LUCK!