Concurrent Programming[1]

Exercise Booklet 7: Erlang – Sequential Fragment

Exercise 1. What is the result of typing these two lines?

```
1> {A,B} = {2,3}.
2> B.
```

Exercise 2. What is the result of these two lines, if they're typed after the previous two?

```
3> {A,C} = {2,5}.
4> {A,D} = {6,6}.
```

Exercise 3. What is the output of each of these lines?

```
1> A=2+3.
2> B=A-1.
3> A=B+1.
4> A=B.
```

Exercise 4. What is the output of each of these lines?

```
5> f(A).
6> A=B.
7> f().
```

Exercise 5. Write the following functions in Erlang (place them in a module `basic.erl`)

1. `mult/2`. Multiplies its two numeric arguments.
2. `double/1`. Returns the double of the numeric argument.
3. `distance/2`: consumes two tuples representing coordinates and returns the Euclidean distance between them.
4. `my_and/2`. Use `if`.
5. `my_or/2`. Use `if`.
6. `my_not/1`. Use `if`.

Exercise 6. Implement the following functions:

1. `fibonacci/1`.
2. `fibonacciTR/1`: tail recursive fibonacci (you might need a helper function).

Exercise 7. Implement the following functions

[1] Some exercises are taken from Simon Thompson's online tutorial on Erlang.

1. `sum/1` that sums up all the numbers in a list.
2. `maximum/1` that computes the maximum of a non-empty list of numbers.
3. `zip/2` that zips two lists.
4. `append/2` that appends two lists (you may not use `++`).
5. `reverse/1` that computes the reverse of a list.
6. `evenL/1` that returns the sublist of even numbers in a given list of numbers.
7. `take/2` such that `take(N,L)` returns a list with the first N elements of L.
8. `drop/2` that returns the result of dropping the first N elements of L.

Exercise 8. Type this out in a file `test.erl`.

```
-export([take/2]).
-include_lib("eunit/include/eunit.hrl").

take(_,[]) -> [].

take_1_test() ->
   ?assertEqual(take(0,[]),[]).
take_2_test() ->
    ?assertEqual(take(0,[1,2,3]),[]).
```

Then type out the following in a shell and write down the output:

```
1> c(test).
{ok,test}
2> test:test().
```

Exercise 9. Define in Erlang the following operations on lists:

1. `map/2`.
2. `filter/2`.
3. `fold/2`.

Exercise 10. Represent binary trees using tuples:

- `{empty}` and
- `{node,Number, LSubtree,RSubtree}`.

Then implement:

1. `sumTree/1` a tail recursive function that adds all the numbers in a tree.
2. `mapTree/2`
3. `foldTree/2`

Exercise 11. Represent general trees using tuples and lists.

- `{node,Number,[GTree1,...,GTreeN]}`.

Then implement:

1. `mapGTree/2`
2. `foldGTree/3`