Concurrent Programming[1]

Exercise Booklet 7: Erlang – Sequential Fragment

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

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

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}.
```

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.
```

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

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

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

   (a) `mult/2`. Multiplies its two arguments.

   (b) `double/1`. Returns the double of the argument.

   (c) `distance/2`: consumes two tuple representing coordinates and returns the Euclidean distance between them.

   (d) `and/2`.

   (e) `or/2`.

   (f) `not/1`.

6. Implement the following functions:

   (a) `fibonacci/1`

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

7. Implement the following functions

   (a) `sum/1` that sums up all the numbers in a list.

   (b) `maximum/1` that computes the maximum of a non-empty list of numbers.

   (c) `zip/2` that zips two lists.

   (d) `append/2` that appends two lists (you may not use `++`).

   (e) `reverse/1` that computes the reverse of a list.

   (f) `evenL/1` that returns the sublist of even numbers in a given list of numbers.

   (g) `take/2` such that `take(N,L)` returns a list with the first N elements of L.

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

(h) `drop/2` that returns the result of dropping the first N elements of L.

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().
```

9. Define in Erlang the following operations on lists:

(a) `map/2`

(b) `filter/2`

(c) `fold/2`

10. Represent binary trees using tuples:

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

Then implement:

(a) `sumTree/1` a tail recursive function that adds all the numbers in a tree.

(b) `mapTree/2`

(c) `foldTree/2`

11. Represent general trees using tuples and lists.

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

Then implement:

(a) `mapGTree/2`

(b) `foldGTree/3`