Exam of Programming Languages

27 January 2016

Exercise OCaML: Hooray! It's Prime!!!

```
let range ?step:(s=1) i j =
  let rec range' n acc =
    if n > j then (List.rev acc) else range' (n+s) (n::acc)
  in range' i [] ;;

let trialdivision x =
  Printf.printf "Trial-Division's Primality Test\t" ;
  (List.length
     (List.filter
        (fun y -> (x mod y) == 0)
        (range 2 (int_of_float (sqrt (float x))+1)))
  == 0);;

(* modular exponent *)

let modexp b e m =
   let rec modexp' c b e' e m =
     if e' <= e then modexp' ((c*b) mod m) b (e'+1) e m
     else c
   in modexp' 1 b 1 e m ;;

(* lucas-lehmer test of primality.                                *)
(* m is the prime of Marsenne, i.e., 2^p-1 where p is an odd prime *)

let lucaslehmer m =
  Printf.printf "Lucas-Lehmer's Primality Test\t" ;
  let rec lucaslehmer p s m =
    if p==0 then s==0
    else lucaslehmer (p-1) ((s*s-2) mod m) m
  in lucaslehmer ((int_of_float ((log ((float m)+.1.))/.(log 2.)))-2) 4 m ;;

let littlefermat p =
  Printf.printf "Little Fermat's Primality Test\t" ;
  (List.length
    (List.filter
      (fun y -> (modexp y (p-1) p) <> 1)
      (range ~step:3 2 (int_of_float (log (float p))+1)))
  == 0);;

let is_prime x =
  let criteria = [
     ((fun y -> y<10000), trialdivision);
     ((fun y -> y<=524287), lucaslehmer);
     ((fun y -> true), littlefermat)] in
    let rec is_prime' x = function
      (p,t)::tl -> if (p x) then (t x)
                   else is_prime' x tl
    | []         -> false
    in is_prime' x criteria ;;
```

Exercise Erlang: You Are Hot!

```
-module(tempsys).
-export([startsys/0]).
-define(CONCAT_ATOM(A, B), list_to_atom(lists:concat([A,B]))).

fromC(X)  -> X.
fromDe(X) -> 100-X*2/3.
fromF(X)  -> (X-32)*5/9.
fromK(X)  -> X-273.15.
fromN(X)  -> X*100/33.
fromR(X)  -> (X-491.67)*5/9.
fromRe(X) -> X*5/4.
fromRo(X) -> (X-7.5)*40/21.

toC(X)  -> X.
toDe(X) -> (100-X)*3/2.
toF(X)  -> X*9/5+32.
```

```
toK(X)  -> X+273.15.
toN(X)  -> X*33/100.
toR(X)  -> X*9/5+491.67.
toRe(X) -> X*4/5.
toRo(X) -> X*21/40+7.5.

regT(T={K,V}) -> register(?CONCAT_ATOM(from, K), spawn(fun() -> loop(V) end)), T.
regTto(T={K,V}) -> register(?CONCAT_ATOM(to, K), spawn(fun() -> loopto(V) end)), T.

%this is the second line of actors
loopto(F) ->
  receive
    {client, C, stub, From, celsius, X} -> From ! {client, C, result, F(X)}, loopto(F);
    Other -> io:format("LoopTo Error: ~p~n", [Other])
  end.

%this is the first line of actors
loop(F) ->
  receive
    {who, From, to, T, val, X} ->
        ?CONCAT_ATOM(to,T)!{client, From, stub, self(), celsius, F(X)}, loop(F);
    {client, C, result, X} -> C!{result, X}, loop(F);
    Other -> io:format("Loop Error: ~p~n", [Other])
  end.

startsys() ->
  FromT = [{'C', fun fromC/1}, {'De', fun fromDe/1}, {'F', fun fromF/1},
           {'K', fun fromK/1}, {'N', fun fromN/1}, {'R', fun fromR/1},
           {'Re', fun fromRe/1}, {'Ro', fun fromRo/1}],
  ToT   = [{'C', fun toC/1}, {'De', fun toDe/1}, {'F', fun toF/1},
           {'K', fun toK/1}, {'N', fun toN/1}, {'R', fun toR/1},
           {'Re', fun toRe/1}, {'Ro', fun toRo/1}],
  lists:map(fun regT/1, FromT),lists:map(fun regTto/1, ToT).
```

Exercise Scala: LogLang.

```
import scala.util.parsing.combinator._
import scala.collection.mutable._

import java.io.{File,FileInputStream,FileOutputStream}
import scala.language.postfixOps
import util.Try

class LogLangCombinators() extends JavaTokenParsers {
   def program = rep1(task)
   def task = "task" ~> ident ~ ( "{" ~> rep1(stmt) <~ "}" )
   def stmt = remove | rename | backup | merge
   def remove = "remove" ~> unquoted ^^ {
     case s =>
       Try(new File(s).delete()).getOrElse(false)
   }
   def rename = "rename" ~> unquoted ~ unquoted ^^ {
     case s ~ t =>
       Try(new File(s).renameTo(new File(t))).getOrElse(false)
   }
   def backup = "backup" ~> unquoted ~ unquoted ^^ {
     case s ~ t =>
       Try((
         () => {
          new FileOutputStream(new File(t)).getChannel() transferFrom(
            new FileInputStream(new File(s)) getChannel, 0, Long.MaxValue );
          true
         })()
       ).getOrElse(false)
   }
   def merge = "merge" ~> unquoted ~ unquoted ~ unquoted ^^ {
     case s1 ~ s2 ~ t =>
       Try((
         () => {
          new FileOutputStream(new File(t)).getChannel() transferFrom(
              new FileInputStream(new File(s1)) getChannel, 0, Long.MaxValue);
          new FileOutputStream(new File(t), true).getChannel() transferFrom(
              new FileInputStream(new File(s2)) getChannel, 0, Long.MaxValue);
          true
         })()
       ).getOrElse(false)
   }
   def unquoted = stringLiteral ^^ { case s => s.substring(1, s.length-1) }
}

object LogLangEvaluator {
  def main(args: Array[String]) = {
```

```
      val p = new LogLangCombinators()

      args.foreach { filename =>
         val src = scala.io.Source.fromFile(filename)
         val lines = src.mkString
         p.parseAll(p.program, lines) match {
             case p.Success(s,_) =>
               s.foreach {
                  _ match {
                    case p.~(s1,l) => {
                       println("Task "+ s1);
                       l.zipWithIndex.foreach{ case(e,i) => println("  [op"+(i+1)+"]  "+e
                    }
               }
         }
             case x => print(x.toString)
         }
         src.close()
      }
   }
}
```