Walter Cazzola

Home Page
ADAPT Lab.
Curriculum Vitae
Research Topic

Exam of Programming Languages

08 July 2016

Exercise ML/OCaML: A «Continued» QuickSort.

```
let cqsort  (>:) l =
   let rec cqsort l k = match l with
     ([] | [_]) -> k l
   | h :: t -> cqsort
       (List.filter (fun x -> h >: x) t)
       (fun ll -> k (cqsort
            (List.filter (fun x -> x >: h) t)
            (fun gl -> (ll @ [h]) @ gl)))
   in cqsort l (fun x -> x)
```

Exercise Erlang: Joseph's Problem.

```
-module(hebrew).
-export([start/3]).

start(Label, N, K) ->
   receive
     {neighbor, PID, master, M} -> loop(Label, PID, M, N, K);
     Other -> io:format("### error ~p~n", [Other])
   end.

loop(Label,Next,Master,N,K) ->
   receive
     % Your neighbor is died! Long live to the new neightbor!
     {newn, NewNext} -> loop(Label, NewNext, Master, N, K);
     % You are the last standing! Communicate it to the master!
     {msg, _, from, _, stepn, N, stepk, 1} ->
         Master ! {msg, "I'm the survivor!", from, self(), label, Label};
     % This is not your lucky day! You drawn the shortest match.
     {msg, MSG, from, Prev, stepn, Sn, stepk, K} ->
         Prev!{newn,Next}, Next!{msg,MSG,from,Prev,stepn,Sn+1,stepk,1};
     % Phwee!
     {msg, MSG, from, _, stepn, Sn, stepk, Sk} ->
         Next! {msg, MSG, from, self(), stepn, Sn, stepk, Sk+1},
         loop(Label, Next, Master, N, K)
   end.
```

```
-module(joseph).
-export([joseph/2]).

joseph(N,K) ->
  PIDS = [spawn(hebrew, start, [G, N, K]) || G <- lists:seq(1,N)],
  [P! {neighbor, Next, master, self()} ||
      {P, Next} <- lists:zip(PIDS, tl(PIDS)++[hd(PIDS)])],
  hd(PIDS) ! {msg, "Who will survive?", from,
            lists:last(PIDS), stepn, 1, stepk, 1},
  receive
     {msg, "I'm the survivor!", from, _, label, L} ->
        io:format("In a circle of ~p people, killing number ~p~n
            Joseph is the Hebrew in position ~p~n", [N, K, L]);
     Other -> io:format("### error ~p~n", [Other])
  end.
```

Exercise Scala: WtF -- What the F...un?

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

class WtFCombinators(var the_stack: Stack[Int],
          var the_table: HashMap[Char,(Int, String)],
          var args_table: Array[Int]) extends JavaTokenParsers {
  def wtf_program = def_sect ~ wtf_body ^^ {_ => (the_stack,the_table)}
  def def_sect = rep(adef)
  def wtf_body = expr ~ rep(expr)
  def adef = "def" ~> fun_name ~ args ~ ("=" ~> """.*\n""".r ) ^^ {
           case c ~ n ~ s => the_table(c) = (n, s.dropRight(1));
```

```
      }

  def fun_name = """[A-Z]""".r ^^ { s => s.charAt(0) }
  def block = """\[.*?\]""".r ^^ {s => s.substring(1,s.length-1)}
  def args = decimalNumber ^^ { n => n.toInt }
  def expr: Parser[Any] = (intexpr | varexpr | fun_call | if_expr
      | unop ^^ { (f:(Int => Int)) => the_stack.push(f(the_stack.pop)) }
      | "!" ^^ { _ => println(format("%s", the_stack.pop)) }
  )
  def intexpr = "0" ^^ { _ => the_stack.push(0) }
  def unop = (
        "-" ^^ { _ => (a: Int) => a-1 }
      | "+" ^^ { _ => (a: Int) => a+1 }
  )
  def if_expr = "?" ~> block ~ (":" ~> block) ^^ {
    case b1 ~ b2 =>
      if (the_stack.pop == 0) parseAll(wtf_body, b1)
      else parseAll(wtf_body, b2)
  }
  def varexpr = "$" ~> decimalNumber ^^ {
     n => the_stack.push(args_table(n.toInt))
  }
  def fun_call = fun_name ^^ { c =>
    val argc = the_table(c)._1
    var local_args_table = new Array[Int](10)
    argc to 1 by -1 foreach( n => local_args_table(n) = the_stack.pop )
    val p1 = new WtFCombinators(the_stack, the_table, local_args_table)
    p1.parseAll(p1.wtf_body, the_table(c)._2)
  }
}

object WtFEvaluator {
  def main(args: Array[String]) = {
     val p =
       new WtFCombinators(
             new Stack[Int](),
             new HashMap[Char,(Int,String)](),
             new Array[Int](10))

     args.foreach { filename =>
        val src = scala.io.Source.fromFile(filename)
        val lines = src.mkString
        p.parseAll(p.wtf_program, lines) match {
            case p.Success((s,t),_) =>
               println(s)
               println("Symbol Table :-")
               t foreach { m => println(m) }
            case x => print(x.toString)
        }
        src.close()
     }
  }
}
```