Principles of Programming Languages, 2021.08.31

Important notes
- Total available time: 1h 30'.
- You may use any written material you need, and write in Italian, if you prefer.
- You cannot use electronic devices during the exam: every phone must be turned off and kept on your table.
- You cannot use library functions not covered in class in your code.

Exercise 1, Scheme (13 pts)

1) Define a procedure which takes a natural number *n* and a default value, and creates a *n* by *n* matrix filled with the default value, implemented through vectors (i.e. a vector of vectors).

2) Let S = {0, 1, ..., n-1} x {0, 1, ..., n-1} for a natural number *n*. Consider a *n* by *n* matrix M, stored in a vector of vectors, containing pairs (x,y) ∈ S, as a function from S to S (e.g. f(2,3) = (1,0) is represented by M[2][3] = (1,0)). Define a procedure to check if M defines a **bijection** (i.e. a function that is both injective and surjective).

Exercise 2, Haskell (11 pts)

Consider a *Slist* data structure for lists that store their **length.** Define the *Slist* data structure, and make it an instance of Foldable, Functor, Applicative and Monad.

Exercise 3, Erlang (8 pts)

Define a function which takes two list of PIDs [x_1, x_2, ...], [y_1, y_2, ...], having the same length, and a function *f*, and creates a different "broker" process for managing the interaction between each pair of processes x_i and y_i.
At start, the broker process *i* must send its PID to x_i and y_i with a message *{broker, PID}*. Then, the broker *i* will receive messages *{from, PID, data, D}* from x_i or y_i, and it must send to the other one an analogous message, but with the broker PID and data D modified by applying *f* to it.
A special *stop* message can be sent to a broker *i*, that will end its activity sending the same message to x_i and y_i.

Solutions

Es 1

```
(define (create-matrix size default)
  (define vec (make-vector size #f))
  (let loop ((i 0))
    (if (= i size)
        vec
        (begin
          (vector-set! vec i (make-vector size default))
          (loop (+ 1 i))))))

(define (bijection? m)
  (define size (vector-length m))
  (define seen? (create-matrix size #f))
  (call/cc (lambda (exit)
             (let loop ((i 0))
               (when (< i size)
                 (let loop1 ((j 0))
                   (when (< j size)
                     (let ((datum (vector-ref (vector-ref m i) j)))
                       (if (vector-ref (vector-ref seen? (car datum)) (cdr datum))
                           (exit #f)
                           (vector-set! (vector-ref seen? (car datum)) (cdr datum) #t)))
                     (loop1 (+ 1 j))))
                 (loop (+ 1 i))))
             #t)))
```

Es 2

```
data Slist a = Slist Int [a] deriving (Show, Eq)

makeSlist v = Slist (length v) v

instance Foldable Slist where
  foldr f i (Slist n xs) = foldr f i xs

instance Functor Slist where
  fmap f (Slist n xs) = Slist n (fmap f xs)

instance Applicative Slist where
  pure v = Slist 1 (pure v)
  (Slist x fs) <*> (Slist y xs) = Slist (x*y) (fs <*> xs)

instance Monad Slist where
  fail _ = Slist 0 []
  (Slist n xs) >>= f = makeSlist (xs >>= (\x -> let Slist n xs = f x
                                                in xs))
```

Es 3

```
broker(X, Y, F) ->
    X ! {broker, self()},
    Y ! {broker, self()},
    receive
        {from, X, data, D} ->
            Y ! {from, self(), data, F(D)},
            broker(X, Y, F);
        {from, Y, data, D} ->
            X ! {from, self(), data, F(D)},
            broker(X, Y, F);
        stop ->
            X ! stop,
            Y ! stop,
            ok
    end.


twins([],_,_) ->
     ok;
twins([X|Xs],[Y|Ys],F) ->
    spawn(?MODULE, broker, [X, Y, F]),
    twins(Xs, Ys, F).
```