# *Principles of Programming Languages, 2018.09.05*

**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.

## Exercise 1, Scheme (10 pts)

Define in a purely functional way a procedure called *revlt*, which takes three lists, *($x_1$ ... $x_L$) ($y_1$ ... $y_M$) ($z_1$ .. $z_N$)* and returns the list of vectors: *(#($x_O$ $y_O$ $z_O$) ... #($x_1$ $y_1$ $z_1$))*, where $O \geq 1$ is the smallest among L, M, and N.

E.g. *(revlt '(1 2 3) '(4 5 6 7) '(8 9 10))* is the list *'(#(3 6 10) #(2 5 9) #(1 4 8))*.

## Exercise 2, Haskell (11 pts)

A "dual list", or *Dupl*, is a pair of independent lists.

1) Define a datatype for Dupl. Can it derive Show and/or Eq? If not, make Dupl an instance of both of them.

2) Make Dupl an instance of Functor, Foldable, and Applicative.

## Exercise 3, Erlang (10 pts)

Define a function *create_pipe*, which takes a list of names and creates a process of each element of the list, each process registered as its name in the list; e.g. with [one, two], it creates two processes called 'one' and 'two'. The processes are "connected" (like in a list, there is the concept of "next process") from the last to the first, e.g. with [one, two, three], the process structure is the following:

*three → two → one → self*,

this means that the next process of 'three' is 'two', and so on; *self* is the process that called *create_pipe*.

Each process is a simple repeater, showing on the screen its name and the received message, then sends it to the next process.

Each process ends after receiving the 'kill' message, unregistering itself.

## Solutions

Es 1

```
(define (revlt l1 l2 l3)
  (let loop ((p1 l1)
             (p2 l2)
             (p3 l3)
             (out '()))
    (if (or (null? p1)(null? p2)(null? p3))
        out
       (let ((x1 (car p1))
             (x2 (car p2))
             (x3 (car p3)))
         (loop (cdr p1) (cdr p2) (cdr p3)
               (cons (vector x1 x2 x3) out))))))
```

Es 2

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

instance Functor Dupl where
    fmap f (Dupl l r) = Dupl (fmap f l) (fmap f r)

tfoldr :: (a -> b -> b) -> b -> (Dupl a) -> b
tfoldr f i (Dupl l r) = foldr f i (l ++ r)

instance Foldable Dupl where
    foldr = tfoldr

instance Applicative Dupl where
    pure x = Dupl [x] []
    (Dupl f1 f2) <*> (Dupl x1 x2) = Dupl (f1 <*> x1) (f2 <*> x2)
```

Es 3

```
repeater(Next, Name) ->
    receive
        kill ->
            unregister(Name),
            Next ! kill;
        V ->
            io:format("~p got ~p~n", [Name, V]),
            Next ! V,
            repeater(Next, Name)
    end.

create_pipe([], End) -> End;
create_pipe([X|Xs], Next) ->
    P = spawn(?MODULE, repeater, [Next, X]),
    register(X, P),
    create_pipe(Xs, X).
```