Principles of Programming Languages, 2019.09.03

Important notes
- Total available time: 2h.
- You may use any written material you need, and write in Italian, if you prefer.
- You cannot use electronic devices during the exam.
- You cannot use library functions not covered in class in your code.

Exercise 1, Scheme (9 pts)

Consider the following code:

```
(define (a-function lst sep)
  (foldl (lambda (el next)
           (if (eq? el sep)
               (cons '() next)
               (cons (cons el (car next))
                     (cdr next))))
         (list '()) lst))
```

1) Describe what this function does; what is the result of the following call?

(a-function '(1 2 nop 3 4 nop 5 6 7 nop nop 9 9 9) 'nop)

2) Modify *a-function* so that in the example call the symbols *nop* are not discarded from the resulting list, which must also be reversed (of course, without using *reverse*).

Exercise 2, Haskell (14 pts)

Consider the data structure *Tril*, which is a generic container consisting of three lists.

1) Give a data definition for Tril.

2) Define *list2tril*, a function which takes a list and 2 values x and y, say $x < y$, and builds a Tril, where the last component is the ending sublist of length x, and the middle component is the middle sublist of length y-x. Also, *list2tril L x y = list2tril L y x.*

E.g. *list2tril [1,2,3,4,5,6] 1 3* should be a Tril with first component [1,2,3], second component [4,5], and third component [6].

3) Make Tril an instance of Functor and Foldable.

4) Make Tril an instance of Applicative, knowing that the concatenation of 2 Trils has first component which is the concatenation of the first two components of the first Tril, while the second component is the concatenation of the ending component of the first Tril and the beginning one of the second Tril (the third component should be clear at this point).

Exercise 3, Erlang (9 pts)

1) Define a *split* function, which takes a list and a number n and returns a pair of lists, where the first one is the prefix of the given list, and the second one is the suffix of the list of length n.

E.g. split([1,2,3,4,5], 2) is {[1,2,3],[4,5]}.

2) Using *split* of 1), define a *splitmap* function which takes a function f, a list L, and a value n, and splits L with parameter n, then launches two process to map f on each one of the two lists resulting from the split. The function s*plitmap* must return a pair with the two mapped lists.

Solutions

Es 1
a-function returns a list of lists, where each list is taken backwards, and sep is used for a separator. The resulting list is: ((9 9 9) () (7 6 5) (4 3) (2 1))

The modified function is:
```
(define (another-function lst sep)
  (foldr (lambda (el next)
           (if (eq? el sep)
               (cons (list el) next)
               (cons (cons el (car next))
                     (cdr next))))
         (list '()) lst))
```

Es 2
```
data Tril a = Tril [a] [a] [a] deriving (Show, Eq)

instance Functor Tril where
  fmap f (Tril x y z) = Tril (fmap f x)(fmap f y)(fmap f z)

instance Foldable Tril where
  foldr f i (Tril x y z) = foldr f (foldr f (foldr f i z) y) x

(Tril x y z) +++ (Tril a b c) = Tril (x ++ y) (z ++ a) (b ++ c)


trilconcat t = foldr (+++) (Tril [][][]) t
trilcmap f t = trilconcat $ fmap f t

instance Applicative Tril where
  pure x = Tril [x][][]
  x <*> y = trilcmap (\f -> fmap f y) x

list2tril lst n1 n2 = let (_,_,[x,y,z]) = foldr helper (n1, n2, [[]]) lst
                      in Tril x y z
  where
    helper el (0, m, next)   = (-1, m-1, [el]:next)
    helper el (n, 0, next)   = (n-1, -1, [el]:next)
    helper el (n, m, (x:xs)) = (n-1, m-1, (el:x):xs)
```

Es 3
```
helper(E, {0, L}) ->
    {-1, [[E]|L]};
helper(E, {V, [X|Xs]}) ->
    {V-1, [[E|X]|Xs]}.

split(L, N) ->
    {_, R} = lists:foldr(fun helper/2, {N, [[]]}, L),
    R.

mapper(F, List, Who) ->
    Who ! {self(), lists:map(F, List)}.

splitmap(F, L, N) ->
    [L1, L2] = split(L, N),
    P1 = spawn(?MODULE, mapper, [F, L1, self()]),
    P2 = spawn(?MODULE, mapper, [F, L2, self()]),
    receive
        {P1, V1} ->
            receive {P2, V2} ->
                    {V1, V2}
            end
    end.
```