Fontys
Information and Communication Technology
Academic Preparation

Module : Functional Programming (FP)	Date : 26-03-2024
Lecturer : Jesús Ravelo	Time : 4.00pm-5.30pm
Accepted resources : material on your laptop, no access to internet.	

Next to these instructions, you will find an Elm project called Exam. In the project, you will find four modules, each corresponding to one of the four questions below. Implement the required functions in each module.

Also, all of the modules of the Elm project contain some tests. Make sure your implementations pass all these tests, and make sure that you also do some extra sensitive testing (but only manually, you do not need to add more automatic test cases to the code).

Question A *(20 points)*

Implement function `elemAt` with signature `elemAt: Int -> List a -> Maybe a` , to find the element at a specific index in a list, provided the index is valid (which means non-negative and smaller than the length of the list). That is,

`elemAt  0 ['a', 'e', 'i', 'o', 'u']` returns `Just 'a'` ,
`elemAt  3 ['a', 'e', 'i', 'o', 'u']` returns `Just 'o'` ,
`elemAt  6 ['a', 'e', 'i', 'o', 'u']` returns `Nothing` ,
`elemAt -3 ['a', 'e', 'i', 'o', 'u']` returns `Nothing` ,
and so on.

You must only use recursion, without using any of the predefined Elm functions.

Question B *(30 points)*

Implement three versions of function `last` with signature `last: List a -> Maybe a` . This function receives a list and returns its last element, if there is one. That is, `last []` is `Nothing` , while `last [1, 3, 5]` is `Just 5` .

You must implement three versions of this function:

- `lastA` , where you do not use any of the predefined `List` functions but only use recursion.

- `lastB` , defined as `lastB = List.foldl accLeft initLeft` , where you only need to implement the helpers `accLeft` and `initLeft` .

- `lastC` , defined as `lastC = List.foldr accRight initRight` , where you only need to implement the helpers `accRight` and `initRight` .

Note that in the last two cases you may not change the already given definition of `lastB` and `lastC` . You must only implement the helpers. You may use any predefined Elm function for the helpers, if you wish, but you do not have to.

Question C *(20 points)*

We want to implement a very simple generator of grade reports of students. All grades are paired with the name of the corresponding student in a list of type `List (String, Int)` . For example, in

```
[ ("Anna", 8), ("Jane", 6), ("Jane", 7),
  ("Paul", 6), ("Anna", 7), ("Jane", 9) ]
```

we see two grades registered for student Anna, three for Jane and one for Paul.

A report for a student is a triple of type `(String, List Int, Float)` which contains the name of the student, the corresponding list of grades and the average grade. For example, with the list of grades above, Jane would have report `("Jane", [6, 7, 9], 7.33)`.

Implement function `report` with signature

```
List (String, Int) -> String -> Maybe (String, List Int, Float) ,
```

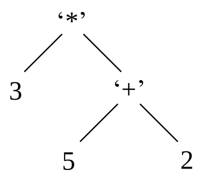
to extract, from a given list of grades and for a given student, the corresponding report. The function is expected to return an actual report (a `Just` value) if the student appears on the list; otherwise, it returns `Nothing` . That is, for example, if we call `grades` the example list above,

`report grades "Jane"` returns `Just ("Jane", [6, 7, 9], 7.33)` ,
`report grades "Paul"` returns `Just ("Paul", [6], 6)` ,
`report grades "Pete"` returns `Nothing` ,
and so on.

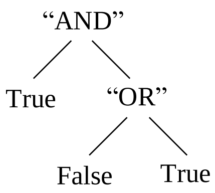
If you wish, you may use any of the predefined Elm functions. However, you do not have to.

Question D *(30 points)*

Consider binary trees that represent expressions with binary operators. For example,



and



.

Such trees can be represented with the type

```
type Expr a b = Leaf a | Node b (Expr a b) (Expr a b)
```

,

where `a` would be the type of the information stored at the leaves and `b` would be the type of the information stored at the internal nodes.

The two examples above would be represented with the values

```
Node '*' (Leaf 3) (Node '+' (Leaf 5) (Leaf 2))
```

and

```
Node "AND" (Leaf True) (Node "OR" (Leaf False) (Leaf True))
```

of types `Expr Int Char` and `Expr Bool String`, respectively.

Now implement the following two functions:

- Function `height` with signature `height: Expr a b -> Int`, which returns the height of the expression tree. That means the number of levels that can be traversed from the root of the tree to the lowest leaf. For both of the examples above, it would return `2`.

- Function `boolEval` with signature `boolEval: Expr Bool String -> Maybe Bool`, which returns the result of evaluating a Boolean expression. This function should succeed, returning a `Just` value, as long as the only strings used are `"AND"` and `"OR"`; otherwise, it should return `Nothing`. For the example above, it would return `Just True`.

End of exam.-