Fontys
Information and Communication Technology
Academic Preparation

Module : Functional Programming (FP)	Date : 02-04-2025
Lecturer : Jesús Ravelo	Time : 4.15pm-5.45pm
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 named `Question`*X*`.elm`, with *X* being `A`, `B`, `C` or `D`, each corresponding to one of the four questions below. Implement the required functions in each module.

Also, there are four modules named `TestsQuestion`*X*`.elm`, which contain some tests for the corresponding `Question`*X*`.elm`. 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).

Some of the sub-questions are marked [*], which means that they are the most challenging parts of this exam. You might want to leave those challenging sub-questions for last.

Question A *(20 points)*

Implement function `howManyTimes` with signature `howManyTimes: a -> List a -> Int`, which determines how many times a given element is present in a given list. That is,

> `howManyTimes 'a' ['b', 'a', 'b', 'e', 'l']` returns `1`,
> `howManyTimes 'b' ['b', 'a', 'b', 'e', 'l']` returns `2`,
> `howManyTimes 'c' ['b', 'a', 'b', 'e', 'l']` returns `0`,
> and so on.

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

Question B *(30 points)*

Implement three versions of function `bits` with signature `bits: List Bool -> String`. This function receives a list of boolean values and, interpreting `False` and `True` respectively as binary digits 0 and 1, returns a string that represents the corresponding sequence of binary digits. That is, for example, `bits [True, True, False, True]` returns `"1101"`.

You must implement three versions of this function:

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

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

- `bitsC` , defined as `bitsC = 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 definitions that have already been given of `bitsB` and `bitsC` . You must only implement the helpers.

The only predefined Elm function you are allowed to use in the implementation of `bitsA` and the helpers of `bitsB` and `bitsC` is the string concatenation operator `++` .

Question C *(30 points)*

We want to implement a couple of simple report generators for a travel agency, based on a general function for handling mathematical relations (defined below).

Sub-question C.(i) A mathematical relation can be represented as a list of pairs without repetitions. For example, list `[ ('A', 3), ('A', 7), ('C', 3), ('A', 4), ('D', 1) ]` is a relation between characters and integer numbers.

Under a given relation, the image of a left element is the collection of right elements related to it. For example, under the relation given above, the image of `'A'` is `[3, 7, 4]` , the image of `'B'` is `[]` , and the image of `'C'` is `[3]` .

Implement function `image` with signature `image: List (a, b) -> a -> List b` , which determines the image under a given relation of a given left element.

You may not use recursion. You must use a combination of `List.map` , `List.filter` and other basic Elm features.

You may assume that the input list contains no repeated pairs.

Sub-question C.(ii) We want to implement a simple report generator for a travel agency, based on function `image` defined above. All flights handled by the travel agency are represented in a list of type `List (String, String)` , that is, a relation between strings and strings. For example, in

```
[ ("Amsterdam", "Hamburg"), ("Amsterdam", "Berlin"),
  ("Eindhoven", "Berlin"), ("Eindhoven", "Munich"),
  ("Hamburg", "Amsterdam"), ("Hamburg", "Warsaw"),
  ("Berlin", "Warsaw"), ("Berlin", "Bucharest"),
  ("Berlin", "Munich"), ("Munich", "Sofia") ]
```

we see that from Amsterdam it is possible to fly directly to Hamburg and Berlin, from Eindhoven to Berlin and Munich, and so on.

For simplicity, we assume all flights to be just one-way: being able to fly from Eindhoven to Berlin does not mean we can fly from Berlin to Eindhoven. When both ways are possible, it has to be explicitly listed, like with Amsterdam and Hamburg in the example above.

As said earlier, we want to generate travel reports. A travel report for an origin city will include: the name of the origin city, the number of cities to which we can fly directly, and the list of such cities. For example, with the flights relation given above as example, the report for Amsterdam will be the triple `("Amsterdam", 2, ["Hamburg", "Berlin"])`, the report for Berlin will be the triple `("Berlin", 3, ["Warsaw", "Bucharest", "Munich"])`, and so on.

Implement function `travelReport1` with type

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

that generates, for a given flights relation and for a given origin city, the corresponding travel report.

You must use the previous function `image` plus any other Elm core library functions of your choice.

[*] Sub-question C.(iii) We now want to generate travel reports for cities that can be reached in two steps, that is, by taking two consecutive flights. For example, with the flights relation given above, the report for Amsterdam could be `("Amsterdam", 3, ["Warsaw", "Bucharest", "Munich"])`, the report for Berlin could be `("Berlin", 1, ["Sofia"])`, and so on.

Note that the result should not contain duplicated destinations, and note that coming back to the origin city must not be considered. It is acceptable, however, that the list of destinations is generated in any order.

Implement then function `travelReport2` with the same type as before,

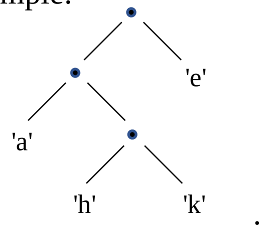
```
List (String, String) -> String -> (String, Int, List String),
```

but generating the travel report of destinations reachable in two steps.

You must use the previous function `image` plus any other Elm core library functions of your choice.

Question D *(20 points)*

A Huffman tree is a binary tree with characters at its leaves. For example:



Such trees can be represented with the type

```
type Huffman = Leaf Char | Node Huffman Huffman,
```

where the example above would be represented with the value

```
Node (Node (Leaf 'a') (Node (Leaf 'h') (Leaf 'k'))) (Leaf 'e').
```

Huffman trees are used for coding characters as sequences of bits. The code of each character is determined by its position in the tree. Traversing the tree from its root to a character determines the code, taking each step to the left as a 0 and each step to the right as a 1.

Therefore, with the example Huffman tree above, character 'a' is assigned code "00", character 'h' is assigned code "010", character 'k' is assigned code "011", and character 'e' is assigned code "1". Other characters do not receive a code.

Sub-question D.(i) The height of a tree is the maximum number of branches that can be traversed from the root to any of the leaves. For a Huffman tree, the height corresponds to the maximum length of any of the corresponding character encodings. For the example tree above, the height is 3.

Implement function `height` with signature `height: Huffman -> Int`, which returns the height of a Huffman tree.

[*] Sub-question D.(ii) We now want to decode single characters (not full messages). Implement function `decode` with signature `decode: Huffman -> String -> Maybe Char`, which returns the encoded character in a Huffman tree, provided the given code is correct.

For example, if we call `ht` the example tree above,

`decode ht "00"` returns `Just 'a'`,
`decode ht "01"` returns `Nothing`,
`decode ht "011"` returns `Just 'k'`,
`decode ht "0111"` returns `Nothing`,
and so on.

If you wish, you may use any of the predefined Elm functions in the core library.

End of exam.-