Written Examination, December 15th, 2023 Course no. 02157

The duration of the examination is 4 hours.

Course Name: Functional programming

Allowed aids: No Aid

The problem set consists of 3 problems which are weighted approximately as follows:
Problem 1: 40%, Problem 2: 40%, Problem 3: 20%

Marking: 7 step scale.

In your programs you are allowed to introduce helper functions; but you must also provide a declaration for each of the required functions, so that it has exactly the type and effect asked for.

You are, in general, allowed to use the .NET library including the modules described in the textbook, e.g., List, Set, Map, Seq, etc. But be aware of the special condition stated in Problem 1.

You are not allowed to use imperative features, like assignments, arrays and so on, in your solutions.

... Continued on next page

Problem 1 (40%)

Questions 1 to 6 in this problem should be solved without using functions from the libraries `List`, `Seq`, `Set` and `Map`.

We consider now a housing association that has a number playgrounds. A playground has a name (type `Name`) and a playing house for children is placed on each playground. The association maintains a register (type `Register`) containing a description (type `Description`) of the house placed on each playground.

The following type declarations model the involved concepts:

```
type Name        = string
type Material    = Wood | Plastic
type Area        = int          // m2 can be assumed to be non-negative
type Adjustment  = int          // may be negative
type Description = Material * Area * Adjustment

type Register   = (Name * Description) list

let reg1 = [("pl1",(Wood,4,-2));[("pl2",(Plastic,3,3)) ];;
```

A description (m, a, adj) of a house is a triple describing its material m (plastic or wood), its area a (in square meters - a non-negative integer) and an adjustment adj that will have a role in the assessment of the value of the house. See below.

The register `reg1` declared above contains descriptions of two playgrounds `"pl1"` and `"pl2"`. On `"pl1"` there is a $4m^2$ wooden house having a negative adjustment (-2) and on `"pl2"` there is a $3m^2$ plastic house having a positive adjustment (3).

1. Declare a function `find` n rec that returns the description of the house placed at playground with name n according to register reg. A suitable exception should be raised if there is no playground named n in reg.

By a *category* we understand a Boolean-valued function on descriptions:

```
type Category = Description -> bool;;
```

We say that a description (m, a, adj) *belongs to* a category cat if $cat(m, a, adj)$ is true. For example, let (m, a, adj) belong to a category *small plastic playing house* if m is Plastic and a is smaller that 4.

2. Declare a value, of type Category, for the category: small plastic playing house.
3. Declare a function `extract` cat reg that returns the list of all playground names from register reg whose descriptions belong to category cat. State the type of `extract`.

... Continued on next page

The housing association has a yearly competition where the playground with the "best" house is awarded. The assessment of a house with description (m, a, adj) has two parts:

- The adjustment assessment. This part should capture how "nice" the house is. It is the value *adj*. It may be negative for a house in bad shape, for example.
- The area assessment. This part is based on a pair of integers (wsm, psm), where *wsm* and *psm* are the square meter assessments for wood and plastic, respectively. We call such a pair a *square meter assessment*. (See type `SqMeterAssess` below.)

  The *area assessment* is given by $wsm \cdot a$ for a wooden house and by $psm \cdot a$ for a plastic house, where a is the area.

```
type SqMeterAssess = int*int  // can be assumed to be positive integers
```

4. Declare two functions `areaAssess` and `totalAssess` that compute the area assessment and the total assessment, respectively, for given square meter assessment and description. The total assessment is the sum af the area assessment and the adjustment. Both functions should have type `SqMeterAssess -> Description -> int`.

5. The association would like to have a sanity check concerning adjustments having a major influence on the assessment. To this end two functions should be declared:

   - `check: SqMeterAssess -> Description -> bool`, that is true when the absolute value of the adjustment does not exceed 50% of the area assessment. (Hint: this can be checked by an expression of the form `2 * abs adj <= aa`), where `adj` is the adjustment, `aa` is the area assessment and `abs` is the built-in function that computes absolute values.
   - `inspectionList: SqMeterAssess -> Register -> Register`.

     The value of `inspectionList` *sqa reg* is a list containing those elements of *reg* where the description part violates the check function above.

6. Declare a function `winners` *sqa reg* that return the names of all playgrounds in register *reg* having the highest total assessment of their houses, given square meter assessment *sqa*. An exception should be raised if the register is empty. Notice that several descriptions may have the same total assessment.

7. Make non-recursive declarations of

   1. the function `inspectionList` from Question 5 and
   2. the function `extract` from Question 3.

   You may select among the following functions from the `List` library in your declarations.

```
List.fold     : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
List.foldBack : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
List.map      : ('a -> 'b) -> 'a list -> 'b list
List.filter   : ('a -> bool) -> 'a list -> 'a list
```

... Continued on next page

Problem 2 (40%)

Consider the following declaration of a function `f`:

```
let rec f xs = match xs with
               | x::y::rest -> (x,y)::f(y::rest)
               | _          -> [];;
```

1. State the most general type of `f`. (Notice that any other type of `f` is an instance of the most general type.) Furthermore, present an argument showing why your answer is the most general type.
2. Describe what `f` is computing. Your description should focus on what its is computing rather than on how the computations are performed.

   Furthermore, suggest an appropriate name for `f` that reflects what it is computing.
3. Make an evaluation of the expression `f [1;2;3;4;5;6;7]`.
4. The declaration of `f` is not tail recursive. Explain briefly why and make a tail-recursive variant of `f` that is based on an accumulating parameter.
5. Make a continuation-based tail-recursive variant of `f`.

Let a function `h` be declared as follows:

`h xs` $=$ `List.fold` e_1 e_2 `xs` where e_1 and e_2 are two expressions.

Notice that `List.fold: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`.

6. Suppose that `h` has type: `int list -> (int * int list)`.

   What do you know about the types of e_1 and e_2?
   Give an argument supporting your answer.
7. Give values for e_1 and e_2 so that `h` has the type `int list -> (int * int list)`.

Consider now the following evaluation:

```
   g true ['a'; 'b'; 'c'; 'd'] []
⇝ g false ['b'; 'c'; 'd'] ['a']
⇝ 'b' :: g true ['c'; 'd'] ['a']
⇝ 'b' :: g false ['d'] ('c'::['a'])
⇝ 'd'::'b'::g true [] ('c'::['a'])
⇝ 'd'::'b'::'c'::['a']
=  ['d';'b';'c';'a']
```

Without knowing how `g` is declared, a fellow student claims: *"g must be tail recursive"*.

8. State whether you agree or disagree with this claim and provide an explanation to your fellow student, so that it becomes clear why you agree or disagree.

... Continued on next page

Problem 3 (20%)

Consider the following tree type `T<'a,'b>` where leaves (constructor `L`) carry values of type `'a` and nodes (constructor `N`) carry values of type `'b`.

```
type T<'a,'b> = | L of 'a
                | N of 'b * T<'a,'b> * T<'a,'b>
```

1. Declare two values, where
   - one should have type `T<int,string>` and
   - the another should have type `T<bool,int list>`.

   You must use both constructors in both values.
2. Declare a function `contains` v t that checks whether v is a value in some leaf in t.
3. Declare a function `nodes` t that gives the list of all values occurring in nodes of t. The order in which values occur in the list is of no concern.
4. Declare a function `replace` v v' t that produces the tree obtained from t by replacing values in nodes that are equal to v with v'.
5. State the types of the functions: `contains`, `nodes` and `replace`.