Written Examination, December 17th, 2015 Course no. 02157

The duration of the examination is 4 hours.

Course Name: Functional programming

Allowed aids: All written material

The problem set consists of 3 problems which are weighted approximately as follows:
Problem 1: 30%, Problem 2: 35%, Problem 3: 35%

Marking: 7 step scale.

Problem 1 (30%)

We consider the use of *appliances* (in Danish 'husholdningsapparater') like washing machines, dishwashers and coffee machines. A *usage* of an appliance a is a pair (a, t), where t is the time span (in hours) the appliance is used. A *usage list* is a list of the individual usages during a full day, that is, 24 hours. This is modelled by:

```
type Appliance = string
type Usage     = Appliance * int

let ad1 = ("washing machine", 2)
let ad2 = ("coffee machine", 1)
let ad3 = ("dishwasher", 2)
let ats = [ad1; ad2; ad3; ad1; ad2]
```

where `ats` is a value of type `Usage list` containing one usage of the dishwasher and two usages of the washing machine and the coffee machine.

1. Declare a function: `inv: Usage list -> bool`, that checks whether all time spans occurring in a usage list are positive.
2. Declare a function `durationOf: Appliance -> Usage list -> int`, where the value of `durationOf` *a ats* is the accumulated time span appliance *a* is used in the list *ats*. For example, `durationOf "washing machine" ats` should be 4.
3. A usage list *ats* is *well-formed* if it satisfies `inv` and the accumulated time span of any appliance in *ats* does not exceed 24. Declare a function that checks this well-formedness condition.
4. Declare a function `delete`(a, ats), where a is an appliance and *ats* is a usage list. The value of `delete`(a, ats) is the usage list obtained from *ats* by deletion of all usages of a. For example, deleting usage of the coffee machine from `ats` should give `[ad1; ad3; ad1]`. State the type of `delete`.

We now consider the *price* of using appliances. This is based on a *tariff* mapping an appliance to the price for one hour's usage of the appliance:

```
type Price  = int
type Tariff = Map<Appliance, Price>
```

5. Declare a function `isDefined` *ats trf*, where *ats* is a usage list and *trf* is a tariff. The value of `isDefined` *ats trf* is true if and only if there is an entry in *trf* for every appliance in *ats*. State the type of `isDefined`.
6. Declare a function `priceOf: Usage list -> Tariff -> Price`, where the value of `priceOf` *ats trf* is the total price of using the appliances in *ats*. The function should raise a meaningful exception when an appliance is not defined in *trf*.

 ... Continued on next page

Problem 2 (35%)

Consider the following F# declarations of two functions `g1` and `g2`:

```
let rec g1 p = function
               | x::xs when p x -> x :: g1 p xs
               | _              -> [];;

let rec g2 f h n x =
   match n with
   | _ when n<0 -> failwith "negative n is not allowed"
   | 0          -> x
   | n          -> g2 h f (n-1) (f x);;
```

1. Give the (most general) types of `g1` and `g2` and describe what each of these two functions computes. Your description for each function should focus on *what* it computes, rather than on individual computation steps.
2. The function `g1` *is not* tail recursive.
   - Make a tail-recursive variant of `g1` using an accumulating parameter.
   - Make a continuation-based tail-recursive variant of `g1`.
3. The function `g2` *is* tail recursive. Give a brief informal explanation of why.

Consider now the following F# declarations of three functions `f1`, `f2` and `f3`:

```
let f1 m n k = seq { for x in [0..m] do
                        for y in [0..n] do
                           if x+y < k then
                              yield (x,y) };;

let f2 f p sq = seq { for x in sq do
                         if p x then
                            yield f x };;

let f3 g sq = seq { for s in sq do
                       yield! g s };;
```

4. What is the value of `List.ofSeq (f1 2 2 3)`?
5. Give an alternative declaration of `f2` using functions from the `Seq` library.
6. Give the (most general) types of `f1`, `f2` and `f3` and describe what each of these three functions computes. Your description for each function should focus on *what* it computes, rather than on individual computation steps.

 ... Continued on next page

Problem 3 (35%)

We consider *rivers*, where a river has a *name*, a source contributing with an *average stream flow rate* (in Danish: 'middelvandføring') and a list of *tributaries* (in Danish: 'bifloder'). A tributary is itself a river. We assume that names are unique for a river and will use the phrase 'the river n' to mean 'the river with name n'. Consider a simple example (where average stream flow rate is abbreviated to flow):

- A river named "R" has flow $10m^3/s$ from its source and it has three tributaries named "R1", "R2" and "R3", respectively.
- The river "R1" has flow $5m^3/s$ from its source and no tributaries.
- The river "R2" has flow $15m^3/s$ from its source and one tributary named "R4".
- The river "R3" has flow $8m^3/s$ from its source and no tributaries.
- The river "R4" has flow $2m^3/s$ from its source and no tributaries.

The following F# types are used to model rivers with tributaries by trees:

```
type Name        = string
type Flow        = int     // can be assumed positive in below questions
type River       = R of Name * Flow * Tributaries
and  Tributaries = River list
```

1. Declare F# values `riv` and `riv3` corresponding to the rivers "R" and "R3".
2. Declare a function `contains` : `Name` → `River` → `bool`. The value of `contains` $n\,r$ is true if and only if the name of r is n, or n is the name of a tributary occurring somewhere in r. For example, `"R"`, `"R1"`, `"R2"`, `"R3"` and `"R4"` constitute all names contained in `riv`.
3. Declare a function `allNames` r which returns a list with all names contained in the river r. The order in which names occur in the list is of no significance.
4. Declare a function `totalFlow` r which returns the total flow in the river mouth (in Danish 'udmunding') of r, by adding the flow from the source of r to the total flows of r's tributaries. For example `totalFlow riv` $= 40$.
5. Declare a function `mainSource` : `River` → (`Name` ∗ `Flow`). If $(n, fl) =$ `mainSource` r, then fl is the biggest flow of some source occurring in the river r and n is the name of a river having this "biggest" source. For example, `mainSource riv = ("R2",15)` and `mainSource riv3 = ("R3",8)`.
6. Declare a function `tryInsert` : `Name` → `River` → `River` → `River option`. The value of `tryInsert` $n\ t\ r$ is `Some` r' if n is the name of a river in r and r' is obtained from r by adding t as a tributary of n. The value of `tryInsert` $n\ t\ r$ is `None` if n is not a name occurring in r.
7. Discuss briefly possible limitations of the above tree-based model of rivers.