Written Examination, May 30th, 2016 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 5 problems which are weighted approximately as follows:
Problem 1: 15%, Problem 2: 30%, Problem 3: 20%, Problem 4: 20%, Problem 5: 15%

Marking: 7 step scale.

Problem 1 (15%)

1. Declare a function `count` for counting the number of occurrences of a given element in a given list. E.g. `count(3,[0;2;3;1;4;3;5]) = 2`. What is the type of your function?

2. An *association list* is a list of pairs $[(x_1, v_1); (x_2, v_2); \dots (x_n, v_n)]$, which associates a value v_i with an element x_i, for $1 \leq i \leq n$. For example, the value associated with `"z"` in `[("x",3); ("y",5); ("z",0)]` is `0`.

   Declare a function `get` for finding the first value v associated with a given element x in an association list. An exception should be raised if no value is found. Give the type of the declared function.

3. Consider the program skeleton:

   ```
   fun sumGt k xs = List.fold (...) ... xs;
   ```

   Fill in the two missing pieces (represented by the dots ...), so that `sumGt` k xs is the sum of those elements in xs which are greater than k. For example, `sumGt` $4\,[1;5;2;7;4;8] = 5+7+8 = 20$.

... Continued on next page

Problem 2 (30%)

We shall now consider *containers* that can either have the form of a *tank*, that is characterized by it length, width and height, or the form of a *ball*, that is characterized by its radius. This is captured by the following declaration:

```
type Container =
   | Tank of int * int * int // (length, width, height)
   | Ball of int             // radius
```

1. Declare two F# values of type `Container` for a tank and a ball, respectively.
2. A tank is called *well-formed* when its length, width and height are all positive and a ball is well-formed when its radius is positive. Declare a function `isWF` : `Container` → `bool` that can test whether a container is well-formed.
3. Declare a function `volume` c computing the volume of a container c. (Note that the volume of ball with radius r is $\frac{4}{3} \cdot \pi \cdot r^3$.)

A *cylinder* is characterized by its radius and height, where both must be positive integers.

4. Extend the declaration of the type `Container` so that it also captures cylinders, and extend the functions `isWF` and `volume` accordingly. (Note that the volume of cylinder with radius r and height h is $\pi \cdot r^2 \cdot h$.)

A *storage* consist of a collection of uniquely named containers, each having a certain *contents*, as modelled by the type declarations:

```
type Name     = string
type Contents = string
type Storage  = Map<Name, Contents*Container>
```

where the name and contents of containers are given as strings.

Note: You may choose to solve the below questions using a list-based model of a storage (`type Storage = (Name * (Contents*Container)) list`), but your solutions will, in that case, at most count 75%.

5. Declare a value of type `Storage`, containing a tank with name `"tank1"` and contents `"oil"` and a ball with name `"ball1"` and contents `"water"`.
6. Declare a function `find` : `Name` → `Storage` → `Contents` $*$ `int`, where `find` n stg should return the pair (cnt, vol) when cnt is the contents of a container with name n in storage stg, and vol is the volume of that container. A suitable exception must be raised when no container has name n in storage stg.

... Continued on next page

Problem 3 (20%)

Consider the following F# declarations of a type `T<'a>` for binary trees having values of type `'a` in nodes, three functions `f`, `h` and `g`, and a binary tree `t`:

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

let rec f g t1 t2 =
   match (t1,t2) with
   | (L,L) -> L
   | (N(ta1,va,ta2), N(tb1,vb,tb2))
           -> N(f g ta1 tb1, g(va,vb), f g ta2 tb2);;

let rec h t = match t with
              | L            -> L
              | N(t1, v, t2) -> N(h t2, v, h t1);;

let rec g =
   function
   | (_,L)                      -> None
   | (p, N(t1,a,t2)) when p a -> Some(t1,t2)
   | (p, N(t1,a,t2))            -> match g(p,t1) with
                                   | None -> g(p,t2)
                                   | res  -> res;;

let t = N(N(L, 1, N(N(L, 2, L), 1, L)), 3, L);;
```

1. Give the type of `t`. Furthermore, provide three values of type `T<bool list>`.
2. Give the (most general) types of `f`, `h` and `g` 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.
3. Declare a function `count` $a\,t$ that can count the number of occurrences of a in the binary tree t. For example, the number of occurrences of 1 in the tree `t` is 2.
4. Declare a function `replace`, so that `replace` $a\,b\,t$ is the tree obtained from t by replacement of every occurrence of a by b. For example, `replace 1 0 t` gives the tree `N(N(L, 0, N(N(L, 2, L), 0, L)), 3, L)`.

... Continued on next page

Problem 4 (20%)

Consider the following F# declarations:

```
let rec f x y = if x <= y then y :: f x (y-1) else [];;

let rec g h pr =
   match pr with
   | ([],_)          -> false
   | (_, [])         -> false
   | (x::xs, y::ys) -> h(x,y) || g h (x::xs, ys) || g h (xs, y::ys);;

let k pr = g (fun (x,y) -> x=y) pr;;
```

1. Give the (most general) type of `f`. Give the values of the expressions `f 1 0` and `f 1 3`. Describe what `f` computes.
2. Give the (most general) type of `g`, and describe what `g` computes.
3. Give the (most general) type of `k`, and describe what `k` computes.
4. The function `f` *is not* tail recursive.
   - Make a tail-recursive variant of `f` using an accumulating parameter.
   - Make a continuation-based tail-recursive variant of `f`.

Problem 5 (15%)

Consider now the following declaration of a type `ListTree<'a>` for so-called *list trees*, where a node can have 0, 1, or more children, that is, a list of children.

```
type ListTree<'a> = Node of 'a * (ListTree<'a> list);;

let t1 = Node(1,[])
let t2 = Node(2, [t1;t1])
let t3 = Node(3, [t1;t2;t2])
let t  = Node(1, [t3])
```

1. Declare a function `contains` $a\,t$ that can test whether the list tree t contains a. For example, `contains 2 t` is true and `contains 0 t` is false. State the type of `contains`.
2. Declare a function `maxNoC` t that gives the max number of children of any node in the list tree t. For example, `maxNoc t` is 3.
3. Declare a replace function `replace` $a\,b\,t$ for list trees, that is, it should compute the list tree obtained from t by replacement of every occurrence of a by b.