Written Examination, May 27th, 2021 Course no. 02157

The duration of the examination is 4 hours.

Course Name: Functional programming

Allowed aids: All aids

The problem set consists of 5 problems which are weighted approximately as follows:
Problem 1: 40%, Problem 2: 10%, Problem 3: 30%, Problem 4: 10%, Problem 5: 10%

Marking: 7 step scale.

... Continued on next page

Formalities

This is a written 4 hours' exam to be taken at the student's home. It is an online exam with all aids allowed and open access to the internet. In particular, it is expected that you use an F# system. The exam set is announced on the Digital Exam platform. The exam set is released at 9:00 on Thursday, May 27, and your solution must be uploaded 4 hours later, that is no later than 13:00.

The exam set consists of a pdf-file `exam02157.pdf`. Furthermore, there is an accompanying file `ProgramSkeleton02157.fsx` containing program snippets from the pdf-file. You should hand in your solution in the form of a single F# file (with extension `.fsx` or `.fs`) , where the file name consists of your name and study number, for example, `JohnSmithS012345.fsx`. The file contents should start with your name and study number.

Textual answers to questions, explanations etc. should be included as comments in the solution file. For example, the answer to Question 2 in Problem 2 could appear as follows:

```
(*
Question 2.2

----- your answer to this question -----

*)
```

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. If a program you want to hand in does not pass the compiler, then include it in a comment as shown above.

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 the start of Problem 1. You are not allowed to use imperative features, like assignments, arrays and so on, in your solutions.

Exam Fraud

This is a strictly individual exam. You are not allowed to discuss any part of the exam with anyone in or outside the course. Submitting answers (code or text) you have not written entirely by yourself, or sharing your answers with others, is considered exam fraud.

This is an open-book exam and you are welcome to make use of any reading material from the course, or elsewhere. However, make sure to use proper and specific citations for any material from which you draw inspiration - including what you may find on the Internet, such as snippets of code.

Also note that it is not allowed to copy any part of the exam text (or supplementary skeleton file) and publish or share it with others during or after the exam.

Breaches of the above policy will be handled in accordance with DTU's disciplinary procedures.

... Continued on next page

Problem 1 (40%)

All questions in this problem, with the exception of question 1.5, should be solved without using functions from the libraries `List`, `Seq`, `Set` and `Map`.

Consider a situation where a huge population should get vaccinated at a vaccination centre having *capacity* to vaccinate several persons simultaneously in the same *time slots*. A time slot is identified by an integer. The centre keeps track of the *available slots*. This is captured by the following type declarations:

```
type TimeSlot       = int
type Capacity       = int
type AvailableSlots = (TimeSlot * Capacity) list
```

For a value $tcs = [(t_0, c_0); \ldots; (t_{n-1}, c_{n-1})]$, $n \geq 0$, of type `AvailableSlots` we require that

1. the capacities in c_i, $0 \leq i < n$, are positive, and that
2. the time slots occur in ascending order, that is, $t_0 < t_1 < \cdots < t_{n-1}$.

We call this condition (1. and 2.) the *available slot invariant*.

If (t_i, c_i) is an element of tcs we say that the time slot t_i is *free* and c_i more persons can get vaccinated in this time slot. That is, c_i is the remaining (or currently available) capacity of this time slot. If there is no element (t, c) for time slot t in tcs, then t is not a free time slot — the remaining capacity is 0.

1. Declare a function of type `AvailableSlots -> bool` that checks whether the available slot invariant holds.

When solving the below questions you may assume that values of type `AvailableSlots` satisfy the available slot invariant when they appear as arguments to functions. When a function returns a value tcs of type `AvailableSlots`, then tcs must satisfy the invariant.

2. Declare a function `incSlot: TimeSlot->AvailableSlots -> AvailableSlots`. The value of `incSlot` t tcs is obtained from tcs by incrementing the capacity of time slot t by one.
3. Declare a function `dcrSlot: TimeSlot->AvailableSlots -> AvailableSlots`. The value of `dcrSlot` t tcs is obtained from tcs by decrementing the capacity of time slot t by one. If t is not free in tcs, then the value of the function is tcs.
4. Declare a function `acquireLT: TimeSlot->AvailableSlots-> TimeSlot option`. The value of `acquireLT` t tcs is `Some` t' if t' is the first free time slot in tcs that is *later than* t, i.e. $t < t'$. If there in no free time slot later than t in tcs, then `None` is returned.

... Continued on next page

Persons can make reservations for vaccinations, where a *reservation* is a pair (p, t) where p is a person and t is the time slot (s)he has for the vaccination. A *state* of our system is given by available time slots and a list of reservations:

```
type Person      = string
type Reservation = Person * TimeSlot
type State       = AvailableSlots * Reservation list
```

A person can have at most one reservation in a reservation list.

The order in which pairs occur in a list of reservations is of no significance.

5. Declare a function of type `Person -> Reservation list -> bool` that checks whether a person has a reservation. You should solve this problem by use of a function from the `List` library – avoid recursion in your declaration.

You are asked below to declare functions that can make a reservation for a person p in a state $s = (tcs, rs)$. The general setting is first find a free time t slot in tcs, then form a new reservation $r = (p, t)$ and return a new state (tcs', rs'), where tcs' is obtained from tcs by decrementing the capacity of t by 1 and rs' is obtained from rs by adding reservation r. Further details are given in questions 1.6. and 1.7.

6. Declare a function `reserveFirst` p (tcs, rs) that can make a reservation for p using the first free time slot in tcs. Exceptions should be raised when p already has a reservation in rs and when there is no free time slot in tcs.
7. Declare a function `reserveLaterThan` t p (tcs, rs). The function should return `None` when there is no free time slot later than t in tcs and when p already has a reservation in rs. Otherwise `Some`(tcs', rs') is returned, where a reservation for p is made using the first free time slot that is later than t.

The last question in this problem concerns a function for deletion of a reservation for a person p in a state $s = (tcs, rs)$. If p does not have a reservation in rs, then s is the value of the function. Otherwise, a reservation $r = (p, t)$ occurs in rs, and the value of the deletion function is (tcs', rs'), where tcs' is obtained from tcs by incrementing the capacity of the time slot for t by 1 and rs' is obtained from rs by removing reservation r.

8. Declare a function for deletion of a reservation for a person in a state.

... Continued on next page

Problem 2 (10%)

The function `countBy` from the `List` library could have the following declaration:

```
let rec ins x = function
                | (y,n) :: ys when x=y -> (y,n+1)::ys
                | pair  :: ys          -> pair::ins x ys
                | []                   -> [(x,1)];;
ins: 'a -> ('a * int) list -> ('a * int) list when 'a : equality

let rec cntBy f xs acc = match xs with
                         | []      -> acc
                         | x::rest -> cntBy f rest (ins (f x) acc);;
cntBy: ('a -> 'b) -> 'a list -> ('b * int) list -> ('b * int) list
       when 'b : equality

let countBy f xs = cntBy f xs [];;
countBy: ('a -> 'b) -> 'a list -> ('b * int) list when 'b : equality
```

where `ins` and `cntBy` are helper functions. Notice that the F# system automatically infers the types of `ins`, `cntBy` and `countBy`.

1. Give an argument showing that

   ```
   'a -> ('a * int) list -> ('a * int) list when 'a : equality
   ```

   is the most general type of `ins` and that

   ```
   ('a -> 'b) -> 'a list -> ('b * int) list -> ('b * int) list
      when 'b : equality
   ```

   is the most general type of `cntBy`. That is, any other type for `ins` is an instance of `'a -> ('a * int) list -> ('a * int) list when 'a : equality`. Similarly for `cntBy`.

An example using `countBy` is:

```
countBy (fun x -> x%2) [1 .. 3];;
val it : (int * int) list = [(1, 2); (0, 1)]
```

2. Give an evaluation showing that `countBy (fun x -> x%2) [1 .. 3]` evaluates to `[(1,2); (0,1)]`. Present your evaluation using the notation $e_1 \rightsquigarrow e_2$ from the textbook, where you can use `=>` in your F# file rather than $\rightsquigarrow$. You should include at least as many evaluation steps as there are calls of `ins`, `cntBy` and `countBy`.

... Continued on next page

Problem 3 (30%)

Consider the following type `T` for binary trees, where leaves (constructor `L`) carry integer values and branch nodes (constructor `B`) carry integer values as well.

```
type T = L of int | B of T * int * T;;

let t0 = B(L 1, 0, B(B(L 3, 2, L 4), 5, L 6));;
let b2 = B(B(L 2, 1, L 2), 0, B(L 2,1,L 2));;
```

The values `t0` and `b2` are shown as trees in the following figure. The roots of both trees carry the value 0 and the leaves of `t0` and `b2` carry the values 1, 3, 4, 6 and 2, respectively.

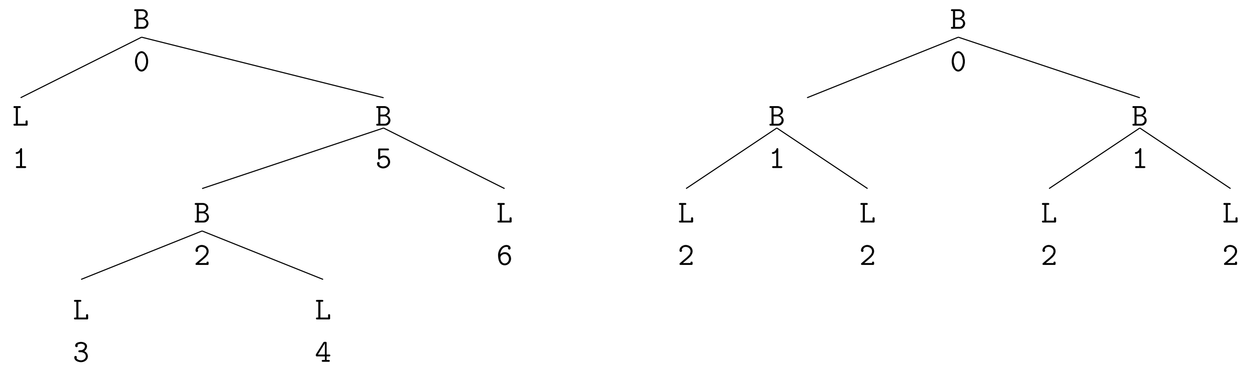

Figure 1: `t0` is shown to the left and `b2` to the right

The *height of a tree* is the maximal number of edges from the root to a leaf node (constructor `L`). For example, `t0` has height 3 and `b2` has height 2.

The *depth of a node n in a tree* is the number of edges from the root to n. For example, each node in `b2` carry the depth of the node as value. This applies for branch nodes as well as for leaf nodes.

A binary tree t is *perfectly balanced* if for every node n in the tree:

- $n =$ `L`(v) is a leaf node, or
- $n =$ `B`$(\mathit{left}, v, \mathit{right})$ is a branch node and the left and right subtrees *left* and *right* have the same height.

The binary tree `b2` is perfectly balanced.

1. Declare a function `extractAtDepth` so that `extractAtDepth`(d, t) gives the list of all values occurring in nodes having depth d in t. The order in which values occur in the list is of no significance. For example, `extractAtDepth(1,b2)` $= [1; 1]$.

... Continued on next page

2. Declare a function `isPerfectlyBalanced` so that `isPerfectlyBalanced` h t is true if and only if t is a perfectly balanced tree with height h.

3. Declare a function `makePerfectlyBalanced` so that `makePerfectlyBalanced` h t is a perfectly balanced tree b_h with height h. Furthermore, each node n in b_h should carry the depth of n as value. That is, `makePerfectlyBalanced 2` = `b2`.

Suppose that a fellow student from 02157 shows you the following program:

```
let rec fHelp t acc =
    match t with
    | L s         -> acc @ [s]
    | B(t1,s,t2) -> fHelp t1 ((fHelp t2 acc) @ [s]);;

let f t = fHelp t [];;
```

and claims: "`fHelp` is a tail-recursive function".

4. The claim is wrong, that is, `fHelp` is not tail recursive. Give an explanation where you point out why the declaration of `fHelp` is not tail recursive.

5. Give a brief explanation of what `f` computes.

Your fellow student makes a new claim: "The is no way in which a significantly better (in terms of run time) implementation of `f` can be obtained when using recursive functions having an accumulating parameter."

6. Make an argument that rejects this new claim. Furthermore, discuss how the run-time efficiency of `f` could be improved.

... Continued on next page

Problem 4 (10%)

Consider an expression of the form:

`List.foldBack` e_x e_y e_z

We would like this expression to have the type `int * bool list`

1. What does this imply concerning the types of e_x, e_y and e_z?

   Justify your answer. Your justification should be based on the fact that `List.foldBack` has the type `('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`.

2. Give three values v_x, v_y and v_z so that `List.foldBack` v_x v_y v_z has type `int * bool list`.

Problem 5 (10%)

Consider the following declaration:

```
let h f a b = seq { for i in [0..a] do
                         for j in [0 .. b] do
                              yield (i,j,f(i,j))};;
```

1. Explain what `h` computes by describing the value of the expression `h` f a b. Furthermore, give an example using `h` that results in a non-empty sequence of values.

2. Give an alternative declaration for `h` that is based on functions from the `Seq` library. That is, you should not use sequence expressions.

   You are allowed to introduce helper functions.