Written Examination, May 29th, 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 4 problems which are weighted approximately as follows:
Problem 1: 20%, Problem 2: 20%, Problem 3: 30%, Problem 4: 30%

Marking: 7 step scale.

Problem 1 (20%)

1. Declare a function: `repeat: string -> int -> string`, so that `repeat` $s\,n$ builds a new string by repeating the string s altogether n times. For example: `repeat "ab" 4` = `"abababab"` and `repeat "ab" 0 = ""`.

2. Declare a function `f` $s_1\,s_2\,n$ that builds a string with n lines alternating between s_1 and s_2. For example: `f "ab" "cd" 4` = `"ab\ncd\nab\ncd"` and `f "XO" "OX" 3` = `"XO\nOX\nXO"`. Note that `\n` is the escape sequence for the newline character.
   Give the type of the function.

3. Consider now certain patterns generated from the strings `"XO"` and `"OX"`. Declare a function `viz` $m\ n$ that gives a string consisting of n lines, where
   - the first line contain m repetitions of the string `"XO"`,
   - the second line contain m repetitions of the string `"OX"`,
   - the third line contain m repetitions of the string `"XO"`,
   - and so on.

   For example, `printfn "%s" (viz 4 5)` should generate the following output

   ```
   XOXOXOXO
   OXOXOXOX
   XOXOXOXO
   OXOXOXOX
   XOXOXOXO
   ```

4. Reconsider the function `repeat` from Question 1.
   1. Make a tail-recursive variant of `repeat` using an accumulating parameter.
   2. Make a continuation-based tail-recursive variant of `repeat`.

Problem 2 (20%)

1. Declare a function `mixMap` so that

$$\texttt{mixMap}\ f\ [x_0;x_1;\ldots;x_m]\ [y_0;y_1;\ldots;y_m] = [f(x_0,y_0);f(x_1,y_1);\ldots;f(x_m,y_m)]$$

2. Declare a function `unmixMap` so that

$$\texttt{unmixMap}\ f\ g\ [(x_0,y_0);(x_1,y_1);\ldots;(x_n,y_n)] = ([f\,x_0;f\,x_1;\ldots;f\,x_n],[g\,y_0;g\,y_1;\ldots;g\,y_n])$$

3. Give the most general types for `mixMap` and `unmixMap`.

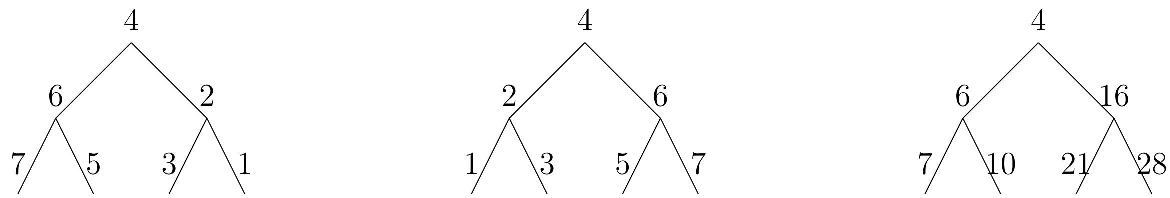
 ... Continued on next page

Problem 3 (30%)

Consider the following F# declarations of a type for binary trees and a binary tree `t`:

```
type Tree<'a> = Lf | Br of Tree<'a> * 'a * Tree<'a>;;

let t = Br(Br(Br(Lf,1,Lf),2,Br(Lf,3,Lf)),4,Br(Br(Lf,5,Lf),6,Br(Lf,7,Lf)));;
```



`reflect t` The tree `t` `accumulate t`

An illustration of the tree `t` is given in the middle part of the above figure. The left part of the figure shows the reflection of `t`, that is, a mirror image of `t` formed by exchanging the left and right subtrees all the way down.

1. Declare a function `reflect` that can reflect a tree as described above.

The right part of the figure shows a tree obtained from `t` by accumulating the values in the nodes of `t` as they are visited through a pre-order traversal. For example, the values in the nodes of `t` are visited in the sequence: 4, 2, 1, 3, 6, 5, 7. Hence, the node of `accummulate t` corresponding to the node of `t` with value 3, has value 10 = 4+2+1+3.

2. Declare a function `accumulate` that can accumulate the values in a tree as described above. Hint: You may declare an auxiliary function having an accumulating parameter.

Consider now the following declarations:

```
let rec k i t =
   match t with
   | Lf          -> Lf
   | Br(tl,a,tr) -> Br(k (i*i) tl, i*a, k (i*i) tr);;

let rec h n m =
   function
   | Br(tl,a,tr) when n=m -> h n 1 tl @ [a] @ h n 1 tr
   | Br(tl,_,tr)          -> h n (m+1) tl @ h n (m+1) tr
   | Lf                   -> []

let q n t = h n n t;;
```

3. Give the most general types of `k` and `q` 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.

 ... Continued on next page

Problem 4 (30%)

The focus of this problem is on courses and curricula at DTU. A *course* is uniquely identified by a *course number* and a course is described by a title and a number of ECTS point. The course base is a map from course numbers to course descriptions. This is captured by the following declarations:

```
type CourseNo   = int
type Title      = string
type ECTS       = int
type CourseDesc = Title * ECTS

type CourseBase = Map<CourseNo, CourseDesc>
```

We require in this problem that *valid* ECTS points are positive integers that are divisible by 5, that is, 5, 10, 15, 20, ... is the sequence of valid ECTS points.

1. Declare a function `isValidCourseDesc: CourseDesc -> bool`,
   where `isValidCourseDesc` *desc* is true if the ECTS part of *desc* is valid.
2. Declare a function `isValidCourseBase: CourseBase -> bool`,
   where `isValidCourseBase` *cb* is true if every course description occurring the course base *cb* is valid, that is, it satisfies the predicate `isValidCourseDesc`.

We shall from now on assume that course descriptions and course bases are valid.

The educations of DTU are organized so that students are required to earn a number of ECTS points within certain course groups. For the BSc programmes, technological core courses (Danish: "teknologiske linjefag") constitutes one such course group. Course groups are organized into a *mandatory part* and a *optional part*. Students following a certain programme must take all courses belonging to the mandatory part, and some of the courses in the optional part. For the bachelor programme in Software Technology, the courses 02131 Embedded Systems and 02141 Computer Science Modelling are among the mandatory technological core courses, while 02157 Functional programming and 02158 Parallel programming are among the optional courses. This is described by the following type declarations, where mandatory courses and optional courses are represented by sets of course numbers:

```
type Mandatory   = Set<CourseNo>
type Optional    = Set<CourseNo>
type CourseGroup = Mandatory * Optional
```

3. Declare a function `disjoint: Set<'a> -> Set<'a> -> bool`, where `disjoint` s_1 s_2 is true if the two sets s_1 and s_2 have no common element, that is, they are disjoint.
4. Declare a function `sumECTS: Set<CourseNo> -> CourseBase -> int`,
   where `sumECTS` *cs* *cb* is the sum of all ECTS points of the courses with numbers in *cs*, where the ECTS points are extracted from course descriptions in the course base *cb*.

 ... Continued on next page

5. A course group (*man*, *opt*) for a bachelor programme is *valid* for a given course base *cb* if:
   - *man* and *opt* are disjoint,
   - the sum of all mandatory ECTS points (i.e. the ECTS sum for all courses in *man*) is less than or equal to 45,
   - the set of optional courses *opt* is empty when the mandatory ECTS points add up to 45, and
   - the total number of ECTS points of mandatory and optional courses should be at least 45.

   Declare a function `isValidCourseGroup: CourseGroup -> CourseBase -> bool` that can check whether a course group is valid for a given course base.

The bachelor programmes are organized according to the *flag model*, with three course groups for *basic natural science courses*, *technological core courses* and *project and professional skills courses*, respectively.

The group of *elective courses* form the fourth component of the flag model. This group is described by a predicate on course numbers, characterizing the courses the study leader has accepted as suitable elective courses. Furthermore, a *course plan* is given by a set of courses numbers:

```
type BasicNaturalScience      = CourseGroup
type TechnologicalCore        = CourseGroup
type ProjectProfessionalSkill = CourseGroup
type Elective                 = CourseNo -> bool

type FlagModel  = BasicNaturalScience * TechnologicalCore
                * ProjectProfessionalSkill * Elective
type CoursePlan = Set<CourseNo>
```

A flag model (*bns*, *tc*, *pps*, *ep*) is *valid* if

- the three course groups *bns*, *tc* and *pps* are all valid,
- no course belongs to more than one of the course groups *bns*, *tc* and *pps*, and
- any course belonging to a course group *bns*, *tc* or *pps* must qualify as an elective course, that is, it must satisfy the predicate *ep*.

6. Declare a function `isValid: FlagModel -> CourseBase -> bool` that can test whether a flag model is valid for a given course base.

A course plan *cs satisfies* a (valid) flag model of a bachelor programme (for a given course base), if the number of ECTS points earned from the courses in *cs* is 180, subject to the requirement that 45 points are earned in each course group of the flag model, including the elective courses.

7. Declare a function `checkPlan: CoursePlan -> FlagModel -> CourseBase -> bool` that can check whether a course plan satisfies a flag model for a given course base.