Problem 1 (35%)

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

We consider now football tournaments, especially the management of standings and results of matches. An example *standings* for a tournament with four *teams* "T1" to "T4" is

```
let ss = [("T1", (3, 5, 1, 9));
          ("T3", (3, 4, 2, 4));
          ("T2", (3, 4, 4, 4));
          ("T4", (3, 0, 6, 0))]
```

We call an element (t, sc) in a standings a *team score*, where t is a team and $sc = (m, gs, gc, p)$ is a *score* consisting of the number of *matches* m the team has played, the number of *goals* gs the team has *scored*, the number of *goals* gc the team has *conceded*, that is, goals scored by an opposing team, and the *points* p the team has collected. In the above standings `ss`, team "T1" has played 3 matches, scored 5 goals, conceded 1 goal, and collected 9 points. The following type declarations model the involved concepts:

```
type Team      = string
type Goal      = int        // can be assumed to be nonnegative
type Point     = int        // can be assumed to be nonnegative
type Matches   = int        // can be assumed to be nonnegative
type Score     = Matches * Goal * Goal * Point
type TeamScore = Team * Score
type Standings = TeamScore list
```

A score $sc_1 = (m_1, gs_1, gc_1, p_1)$ is *better* than a score $sc_2 = (m_2, gs_2, gc_2, p_2)$ if and only if

- the points p_1 of sc_1 is greater than the points p_2 of sc_2 or
- the points of sc_1 and sc_2 are equal and the *goal difference* $gs_1 - gc_1$ of sc_1 is better (that is, greater) than the goal difference $gs_2 - gc_2$ of sc_2.

Two scores sc_1 and sc_2 are considered *equal* if they have the same points and the same goal difference.

1. Declare three functions `better` sc_1 sc_2, `equal` sc_1 sc_2 and `betterOrEqual` sc_1 sc_2 testing whether score sc_1 is better than score sc_2, whether scores sc_1 and sc_2 are equal, and whether score sc_1 is better than or equal to score sc_2, respectively.

 ... Continued on next page

For a standings $[(t_0, sc_0); \ldots (t_i, sc_i); \ldots (t_{n-1}, sc_{n-1})], n \geq 0$, we require two properties

- P_1: the teams $t_0, \ldots t_i, \ldots t_{n-1}$ should be different
- P_2: the scores appear in descending order with the best score appearing first, that is, sc_i is better than or equal to sc_{i+1}, for $0 \leq i < n-1$.

Notice that the above example standings `ss` satisfies both properties.

2. Declare a function `properlyOrdered: Standings -> bool` so that `properlyOrdered` *ss* is true iff *ss* satisfies P_2.

We call the score `(0,0,0,0)`, where no match is played, no go is scored, no goal is conceded and no point is collected, the *initial score.*

3. Declare a function `init: Team list -> Standings` so that `init` $[t_0; \ldots; t_{n-1}], n \geq 0$, creates a standings with n teams associated with the initial score. You may assume that $t_0, \ldots, t_{n-1}$ are different teams.

4. Declare a function `extractScoreOf: Team -> Standings -> Score*Standings`. The value of `extractScoreOf` t ss is a pair (sc, ss') where sc is the score of t in ss, and ss' is obtained from ss by deletion of the element (t, sc). For example

```
extractScoreOf "T2" ss =
   ((3,4,4,4), [("T1",(3,5,1,9)); ("T3", (3,4,2,4)); ("T4", (3,0,6,0))])
```

   You may assume that ss satisfies properties P_1 and P_2; but the function must raise an exception when t is not a team in ss.

5. Suppose $sc = (m, gs, gc, p)$ is the score of a team that completes a new match scoring g_1 goals and conceding g_2 goals. The *updated* score is $sc' = (m+1, gs+g_1, gc+g_2, p+p')$, where p' is 0 if the game is lost $(g_1 < g_2)$, p' is 1 in case of a draw $(g_1 = g_2)$, and p' is 3 if the game is won $(g_1 > g_2)$. Declare the function `update` sc g_1 g_2 $=$ sc'. For example, `update (3,4,4,4) 2 0 = (4,6,4,7)`.

6. Declare a function `insertTeamScore: TeamScore -> Standings -> Standings`. The value of `insertTeamScore` (t, sc) ss is the standings obtained from ss by insertion of (t, sc) in a proper position (satisfying property P_2). You may assume that ss satisfies P_1 and P_2 and that t is not a team in ss. For example

```
insertTeamScore ("T2",(4,6,4,7)) [("T1",(3,5,1,9));
                                   ("T3",(3,4,2,4));
                                   ("T4",(3,0,6,0))] =
[("T1",(3,5,1,9)); ("T2", (4,6,4,7)); ("T3",(3,4,2,4)); ("T4",(3,0,6,0))]
```

7. A *match result* (t_1, t_2, g_1, g_2) consists of two teams t_1 and t_2, and goals g_1 and g_2 scored by t_1 and t_2, respectively.

```
type MatchResult = Team * Team * Goal * Goal
                      // teams can be assumed to be different
```

For example, ("T2","T1",2,0) is the result of a match where "T2" won 2-0 over "T1".

Declare a function newStandings: MatchResult -> Standings -> Standings. The value of newStandings *mr ss* is the standings obtained from *ss* by incorporation of the match result *mr*. For example

```
newStandings ("T2","T1",2,0) ss = [("T1", (4, 5, 3, 9));
                                   ("T2", (4, 6, 4, 7));
                                   ("T3", (3, 4, 2, 4));
                                   ("T4", (3, 0, 6, 0))]
```

Problem 2 (20%)

The function partition from the List library could have the following declaration:

```
let rec partition p xs =
   match xs with
   | []        -> ([],[])
   | x::rest -> let (xs1,xs2) = partition p rest
                if p x
                then (x::xs1,xs2)
                else (xs1,x::xs2)
```

1. What is the type of partition?
   Justify your answer briefly.

2. Declare a and b so that partition a b gives the result

   *val it: int list * int list = ([6; 8; 9], [1; 4; 2])*

3. The above declaration of partition is not tail recursive. Explain briefly why and provide a declaration of a tail-recursive variant of partition that is based on an accumulating parameter.
   Your tail-recursive declaration must be based on an explicit recursion.

4. Complete the following declaration of partition that makes use of List.foldBack:

   ```
   let partition p xs = List.foldBack ...  ...  ...
   ```

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

... Continued on next page

Problem 3 (30%)

We consider now *hierarchies* as trees (type `H<'d>`) with arbitrarily many subtrees (type `Children<'d>`). The nodes and leaves (Constructor `N`) carry descriptions. The type `H<'d>` is polymorphic in order to support descriptions (type variable `'d`) with many forms.

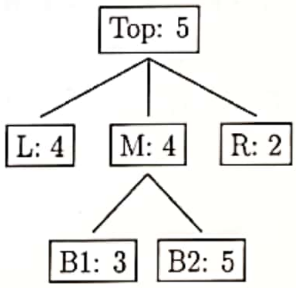
```
type H<'d> = N of 'd * Children<'d>
and Children<'d> = H<'d> list;;
```

1. Declare a value of type `H<bool>` that contains lists of lengths 0, 1 and 2. Make a suitable drawing of the value.

2. Declare a function `descriptionsOf: H<'d> -> 'd list` that gives a list with all descriptions occurring in a hierarchy. The sequence in which descriptions occur in the result is of no importance.

3. Declare a function `map: ('d -> 'e) -> H<'d> -> H<'e>`. The value of `map` f h is the hierarchy obtained from h by application of f to every description in h.

We shall now consider an *organization* as a special kind of hierarchy, where a node, called an organizational *entity*, in the hierarchy carries a description (t, n), where t is the *title* - a string - of an organizational entity, and n is the number of employees associated with that entity. An example organization is shown in the following figure:



where the entity in the root of the organization has title: `"Top"`, five employees, and three sub-entities with titles `"L"`, `"M"` and `"R"`, respectively, and so on.

4. State a type for organizations using the type for hierarchies. Furthermore, give an F# value corresponding to organization shown in the figure above.

5. Declare a function `numberOf` o that gives the number of persons employed by an organization o. For example, the above organization has 23 employees.

6. Declare a function `largest` o that given an organization o returns a pair (ts, m), where m is the maximal number of employees associated with any entity in o and ts is a list containing all titles of entities in o having exactly m employees. The function should, for the above organization, return a pair $(ts, 5)$ where ts contains only `"Top"` and `"B2"`.

 ... Continued on next page

Problem 4 (10%)

A fellow student analyses the declarations:

```
let rec f g xs = match xs with
                 | []       -> 0
                 | x::tail -> g x + f g tail;;

let h(x,y) = x+y+1;;
```

and claims that the following is an evaluation of `f h [(1,2);(2,3);(3,4)]`:

```
   f h [(1,2);(2,3);(3,4)]
⇝ 1+2+1 + h [(2,3);(3,4)]
⇝ 4 + 6 + f h [(3,4)]
⇝ h (3,4)
⇝ 8
⇝ 18
```

This claim is wrong.

1. Give detailed feedback to your fellow student. The feedback should be so that the fellow student can understand why it is wrong and produce a correct evaluation on the basis of your feedback.
2. Make a correct evaluation of `f h [(1,2);(2,3);(3,4)]`.

Problem 5 (5%)

The sequence of natural numbers can be declared as follows:

```
let nat = Seq.initInfinite id;;
```

We want to partition this sequence into an infinite sequence of triples:

```
(0,1,2), (3,4,5), (6,7,8), (9,10,11)  ...
```

Let us call this sequence `nat3`.

1. Give two declarations of `nat3`. One should be based on functions from the `Seq` library, the other should be based on sequence expressions.

Written Examination, December 16th, 2022 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: 35%, Problem 2: 20%, Problem 3: 30%, Problem 4: 10%, Problem 5: 5%

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