Fall Exsam

Problem 1 (20%)

```
type Name = stringtype
type Event = stringtype
type Point = inttype
type Score = Name * Event * Point

type Scoreboard = Score list

let sb = [("Joe", "June Fishing", 35); ("Peter", "May Fishing", 30);("Joe", "May Fishing

(*
    1. Declare a function:inv: Scoreboard -> bool, that checks whether a scoreboard sat-
*)

let inv sb =
    let maxValue = sb.head

    let rec loop sb max =
        match sb with
        | head::tail  ->
                        let (_, _, point) = head
                        if point > max
                        then loop tail head && false
                        else loop tail head && true

    loop sb maxValue

(*
    2.Declare a function insert: Score -> Scoreboard -> Scoreboard, so that inserts sbg
*)

let rec insert s = function
    | head::tail ->
                    let (_, _, p) = head
                    let (_,_, np) = tail.head
                    let (_, _, sp) = s
                    if p > sp && np < sp
                    then s::tail
                    else insert s tail

(*
    3.Declare a functionget: Name*Scoreboard -> (Event*Point) list, where the valueofge
*)

let get (n, sb) = List.map (fun x ->
                                    let (name, event, point) = x
                                    if name = n
                                    then (evnet, point)
                                    else null
                                    ) sb
```

```
(*
    4.Declare a functiontop: int -> Scoreboard -> Scoreboard option.  The value oftopksl
*)

let top
```

Problem 2 (15%)

```
(*
    1.Declare a functionreplacea bxsthat gives the list obtained fromxsby replacing evel
*)

let replace a b xs = List.map (fun x -> if x = a then b else x) xs

(*
    2. Give the (most general) type ofreplace.

    type: 'a -> 'a -> 'a list -> 'a list
*)

(*
    TODO
    3.Is yourreplacefunction tail recursive? Give the brief informal explanation of youl
*)
```

Problem 3 (10%)

```
let pos = Seq.initInfinite (fun i -> i+1)

let seq1 = seq { yield (0,0) // tilføj
                 for i in pos do
                 yield (i,i)  // tilføj
                 yield (-i,-i) } // tilføj

let val1 = Seq.take 5 seq1

let nat = Seq.initInfinite id

let seq2 = seq { for i in nat do
                yield (i,0)
                for j in [1 .. i] do
                yield (i,j) }
```

1.Give the types of the sequencespos,seq1 and val1 and describe their values.

pos :

type: seq

Er en udenlig sekvens som starter fra 1 og dernæst stiger med 1 slåldes at sekvensen vil blive [1;2;3;4;5.....]

seq1 :

type: seq<int * int>

starter med at tiføje tuple (0,0) dernæst bliver der for hver i tilføjet en tuple som indeholder (i, i) og lige efter en tuple som indeholder (-i, -i).

Det vil sige at den føste del af sekvensen vil være [(0,0); (1,1); (-1,-1) ...]

val1 :

type: type: seq

vil indeholde de første 5 værdier af pos og vil derfor indeholde [1;2;3;4;5]

2.Give the type of seq2 and describe the sequence. Furthermore, give the value of val2.

nat vil blot indeholde [0,1;2;3;4 ...] det vil sige de naturlige tal da Identity funktionen anvendes.

seq2 vil bestå af (i,0) efterfuldt i x (i, j)
Det vil sige at den første del af sekevensen vil være

[(0,0); (1,0); (1,1); (2,0); (2,1) .....]

Problem 4 (25%)

```
type Tree<'a,'b> =
                | A of 'a | B of 'b
                | Node of Tree<'a,'b> * Tree<'a,'b>
```

where a valueAais called anA leafand a valueBbis called aB-leaf.

1. Give three values of type Tree<bool,int list>using the constructors A, B and Node.

```
let valueOne = Node( A "a", B "b" )
let valueTwo = Node(Node(A "a", B "b"), A "a")
let valuethree = Node(Node(A "a", Node(A "a", B "b" ) ), A "a")
```

2. Declare a function that counts the number of occurrences of A-leaves in a tree.

```
let countA tree =

    let rec loop tree count =
        match tree with
        | Node (x, y) -> loop x 0 + loop y 0
        | A (_) -> 1
        | B (_) -> 0

    loop tree 0
```

3.Declare a function

todo mangler.....

```
let subst = function
    | (a, 'a, b, 'b) ->
```

Problem 5 (30%)

1.Declare a functiontoListtwhich returns a list of all the values occurring in the nodesof the treet. The order in which values occur in the list is of no significance.

```
type T<'a> = N of 'a * T<'a> list

let td = N("g", [])
let tc = N("c", [N("d",[]); N("e",[td])])
let tb = N("b", [N("c",[])])
let ta = N("a", [tb; tc; N("f",[])])


// svar

let rec namesList = function // function som iterare over liste i en node (N)
    | [] -> []
    | n::ns -> (namesNode n) @ (namesList ns)
and namesNode = function // function som udpakker node til dens navn og kalder videre t
    | N(name,children) -> name :: (namesList children)

let toList node = namesNode node // det er ikke nødventigt med denne metode med blot fo

// test
toList ta

(*
output:
val it : string list = ["a"; "b"; "c"; "c"; "d"; "e"; "g"; "f"]
*)
```

2.Declare a function map f t, which returns the tree obtained from thet by applying the function f to the values occurring in the nodes of t. Give the type of map.

```
let map f t =
    let rec mapList =
        function
        | [] -> []
        | n :: ns -> (nodeMap n) :: (mapList ns)

    and nodeMap =
        function
        | N (name, children) -> N(f name, mapList children)

    nodeMap t


map (fun x -> x) ta
map (fun x -> x + " hej") ta
map (fun x -> if x = "c" then "seeee" else x) ta


// alternativ løsning:

(*
    Løsning hvor funktionen f bliver parset rundt som parameter.
*)

let rec mapL f =
    function
    | [] -> []
    | n :: ns -> (map f n) :: (mapL f ns)
and map f =
    function
    | N (name, children) -> N(f name, mapL f children)

(*
Output:

map (fun x -> x) ta

val it : T<string> =
  N ("a",
     [N ("b", [N ("c", [])]); N ("c", [N ("d", []); N ("e", [N ("g", [])])]);
      N ("f", [])])

-----------------------------------------------------------------

map (fun x -> x + " hej") ta

val it : T<string> =
  N ("a hej",
     [N ("b hej", [N ("c hej", [])]);
      N ("c hej", [N ("d hej", []); N ("e hej", [N ("g hej", [])])]);
      N ("f hej", [])])
```

```
-----------------------------------------------------------------

map (fun x -> if x = "c" then "seeee" else x) ta

val it : T<string> =
  N ("a",
     [N ("b", [N ("seeee", [])]);
      N ("seeee", [N ("d", []); N ("e", [N ("g", [])])]); N ("f", [])])

*)
```

3.Declare a functionisPathistthat checks whetherisis a path int.

```
type Path = int list

let rec path t =
    function
    | [] -> false
    | n :: ns -> (isPath t n) || (path t ns)

and isPath is t =
    match is with
    | node when node = t -> true
    | N (_, children) -> false || (path t children)

isPath ta tb // true
isPath ta td // false
```

4.Declare a functionget: Path→T<'a>→T<'a>. The value ofgetistis the subtree identified by is in t.

```
let rec getChildren =
    function
    | [], children -> children
    | head :: tail, children -> [ get tail (List.item head children) ]

and get path =
    function
    | N (name, children) -> N(name, getChildren (path, children))
```

5.Declare a functiontryFindPathto:'a→T<'a>→Path option. Whenvoccurs insome node oft, then the value oftryFindPathtov tisSomepath, wherevoccurs in thenode oftidentified bypath. The value oftryFindPathtov tisNonewhenvdoes notoccur in a node oft. There is no restriction concerning which path the function shouldreturn whenvoccurs more than once int.