Problem 3

Consider the following declarations:

```
fun f(x, [])     = []
  | f(x, y::ys) = (x+y)::f(x+1, ys);

fun g h []       = []
  | g h (x::xs) = h x :: g (fn y => h(h y)) xs;

fun q p []       = []
  | q p (x::xs) = let val ys = q p xs
                  in if p x then x::ys else ys@[x] end;
```

... Continued on the next page

3A: Determine the types of f, g, and h, and explain how the functions work by giving the value of the following expressions:

1. f$(x, [y_1, y_2, y_3, \ldots, y_n])$
2. g h $[x_1, x_2, x_3, \ldots, x_n]$
3. q p $[x_1, x_2, x_3, \ldots, x_n]$

Problem 4:

Consider the types `expr` and `pat` declared by:

```
infix Add
datatype expr = Const of int | Var of string
              | Add of expr * expr

infix PAdd
datatype pat  = PConst of int | PVar of string
              | PAdd of pat * pat
```

Below we use "expression" and "pattern" to name values of type `expr` and `pat`, respectively.

A: Declare a function `vars`:

```
vars: pat -> string list
```

traversing a pattern (a value of type `pat`) while building a list of the strings s appearing as `PVar` s in the pattern.

... Continued on the next page

B: A pattern (a value of type `pat`) is considered illegal if the same string s appears in the form `PVar` s more than once in the pattern. Declare a function `legal`:

```
legal: pat -> bool
```

giving the value `false` for illegal patterns and `true` otherwise.
(Hint: Declare the function: `duplicate: ''a list -> bool` to decide whether a list contains duplicates, i.e. several occurrences of the same value. Use this function together with the function `vars` from question A.)

C: The types `binding` and `bindingList` are declared by:

```
type binding     = string * expr
type bindingList = binding list
```

A pattern (a value of type `pat`) can *match* an expression (a value of type `expr`), and the matching will then build a list of bindings (a value of type `bindingList`). The matching with associated bindings is governed by the following rules:

1. The pattern `PConst` x matches the expression `Const` x' if $x = x'$, but gives no bindings.
2. The pattern `PVar` s matches every expression e, and gives the list `[(`s`,`e`)]` of bindings.
3. The pattern p_1 `PAdd` p_2 matches the expression e_1 `Add` e_2 if p_1 matches e_1 and p_2 matches e_2, and gives the list of bindings achieved by matching p_1 with e_1 and p_2 with e_2.
4. Any matching of a pattern to an expression is obtained by repeated application of the rules 1, 2 and 3.

Declare a function `match`:

```
match: pat * expr -> bindingList option
```

where `match(`p`,`e`)` = `NONE`, if p does not match e, whereas `match(`p`,`e`)` = `SOME` bs, if p matches e, and the list bs contains the bindings from that match.