CMSC330 Spring 2024 Quiz 1

Problem 1: Basics

[Total 4 pts]

	True	False
OCaml uses type inference to determine the type of variables	T	F
Functional Programming Languages favor mutable data	T	F
Functional Programming aims to decrease the amount of side affects	T	F
Functions are expressions in OCaml	T	F
OCaml does not use type inference to determine the type of variables	T	F
Functional Programming Languages don't favor mutable data	T	F
Functional Programming aims to increase the amount of side effects	T	F

Problem 2: OCaml Typing

[Total 5 pts]

Give the type for the following expressions and what they evaluate to. If there is an error, **either in evaluation OR typing**, put "ERROR".

(a) [2 pts]

```
let f x y = match x with
   [] -> y
  |x::xs -> [x]::[] ;;

f [4] [[6]] ;;
```

Type: `'a list -> 'a list list -> 'a list list`

Evaluation: `[[4]]`

(b) [2 pts]

```
let f a b =
    if b > a then b
    else a < true ;;

f 2.0 false;;
```

Type: `bool -> bool -> bool`

Evaluation: `ERROR`

(c) [2 pts]

```
let rec f g lst = match lst with
   [] -> []
  |x::xs -> (x, g x)::(f g xs) ;;

f (fun x -> x mod 2 = 1) [1;2;3] ;;
```

Type: `('a -> 'b) -> 'a list -> ('a * 'b) list`

Evaluation: `[(1, true); (2, false); (3, true)]`

(d) [2 pts]

```
let f x = match x with
   [] -> [[3]]
  |x::xs -> [x]::[] ;;

f [4];;
```

Type: `int list -> int list list`

Evaluation: `[[4]]`

(e) [2 pts]

```
let f a b c =
    if b > a then c
    else a < true ;;

f true false (fun x y -> x > y);;
```

Type: `bool -> bool -> bool -> bool`

Evaluation: `ERROR`

(f) [2 pts]

```
let f x y = match x with
   [] -> y
  |_::xs -> [2]::[]
  |4::xs -> [4]::[];;

f [4] [[6]] ;;
```

Type: `int list -> int list list -> int list list`

Evaluation: `[[2]]`

(g) [2 pts]

```
let f a b =
    if b > a then (1.3 < 4.6)
    else a < true ;;

f true 1.3;;
```

Type: `bool -> bool -> bool`

Evaluation: `ERROR`

(h) [2 pts]

```
let f x y = match x with
   [] -> y
  |x::xs -> [x]::[] ;;

f [(1,2);(3,4)] [[(6,7)]] ;;
```

Type: `'a list -> 'a list list -> 'a list list`

Evaluation: `[[(1, 2)]]`

(i) [2 pts]

```
let f a b =
    if b > a then ("hello" < "bye")
    else a < true ;;

f (fun x -> x < 1) false;;
```

Type: `bool -> bool -> bool`

Evaluation: `ERROR`

Problem 3: OCaml Expressions

[Total 4 pts]

Write an expression that would have the following types.

The following are examples of correct answers.

(a) [2 pts]

`int list -> 'a -> 'a -> bool`

```
fun x y z -> match x with |h::t -> (if h = 0 then y = z else false) |[] -> true
```

(b) [2 pts]

`(int -> 'a) -> int -> int * 'a list`

```
fun f a -> (a + 1, [f a])
```

(c) [2 pts]

`('a -> 'b) -> 'a -> 'b -> bool`

```
fun x y z -> (x y) = z
```

(d) [2 pts]

`float -> float -> bool -> float list`

```
fun x y z -> if z then [x +.  1.0] else [y]
```

(e) [2 pts]

`(int * 'a) -> (bool -> 'a) -> 'a`

```
fun (a, b) c -> if a + 1 = 2 then b else c true
```

(f) [2 pts]

`int list -> int -> bool list`

```
fun x y -> match x with |h::t -> ([h + 1 = y]) |[] -> [false]
```

(g) [2 pts]

`'a -> 'b list -> 'a -> 'a * 'a`

```
fun x y z -> if y = [] && x = z then (x, z) else (x, z)
```

Problem 4: Coding

[Total 6 pts]

Write a function `calc` that takes a `(int * bool) list` and returns a `(int * bool)`, which consists of the sum of the `int`s, and the result of AND'ing the `bool`s.

You **do NOT have to use `map` or `fold`**, but their definitions are given if you want to use them.
You can write helper methods. **Make sure your function header matches the arguments that `calc` takes in.**

```
(* Examples
    calc [(1,true); (2,false)] = (3,false)
    calc [(3,true); (4,true)] = (7,true)
*)


(* Write your code below for calc lst *)
```

```
let rec map f l = match l with
   [] -> []
  |x::xs -> (f x)::(map f xs)

let rec fold f a l = match l with
   [] -> a
  |x::xs -> fold f (f a x) xs
```

```
(* recursive version *)
let rec func lst = match lst with
[] -> (0,true)
|(i,b)::xs -> let (s,a) = func xs in (s + i, b&&a);;

(* fold version *)
let func lst = fold (fun (a,b) (c,d) -> (a+c, b&&d)) (0, true) lst;;
```