

# CMSC330 Spring 2024 Quiz 1

## Problem 1: Basics

[Total 4 pts]

|                                                                      | True                             | False                            |
|----------------------------------------------------------------------|----------------------------------|----------------------------------|
| OCaml uses type inference to determine the type of variables         | <input checked="" type="radio"/> | <input type="radio"/>            |
| Functional Programming Languages favor mutable data                  | <input type="radio"/>            | <input checked="" type="radio"/> |
| Functional Programming aims to decrease the amount of side affects   | <input checked="" type="radio"/> | <input type="radio"/>            |
| Functions are expressions in OCaml                                   | <input checked="" type="radio"/> | <input type="radio"/>            |
| OCaml does not use type inference to determine the type of variables | <input type="radio"/>            | <input checked="" type="radio"/> |
| Functional Programming Languages don't favor mutable data            | <input checked="" type="radio"/> | <input type="radio"/>            |
| Functional Programming aims to increase the amount of side effects   | <input type="radio"/>            | <input checked="" type="radio"/> |

## 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]::[] ;;
```

Type:

'a list -> 'a list list -> 'a list list

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

Evaluation:

[[4]]

(b) [2 pts]

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

Type:

bool -> bool -> bool

```
f 2.0 false;;
```

Evaluation:

ERROR

(c) [2 pts]

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

Type:

('a -> 'b) -> 'a list -> ('a \* 'b) list

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

Evaluation:

[(1, true); (2, false); (3, true)]

|                                                                                                                        |                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (d)                                                                                                                    | [2 pts]                                                                                                                                                       |
| <pre>let f x = match x with   [] -&gt; [[3]]    x::xs -&gt; [x]::[] ;;  f [4];;</pre>                                  | <div>Type: <input type="text" value="int list -&gt; int list list"/></div> <div>Evaluation: <input type="text" value="[[4]]"/></div>                          |
| (e)                                                                                                                    | [2 pts]                                                                                                                                                       |
| <pre>let f a b c =   if b &gt; a then c   else a &lt; true ;;  f true false (fun x y -&gt; x &gt; y);;</pre>           | <div>Type: <input type="text" value="bool -&gt; bool -&gt; bool -&gt; bool"/></div> <div>Evaluation: <input type="text" value="ERROR"/></div>                 |
| (f)                                                                                                                    | [2 pts]                                                                                                                                                       |
| <pre>let f x y = match x with   [] -&gt; y    _::xs -&gt; [2]::[]    4::xs -&gt; [4]::[];;  f [4] [[6]] ;;</pre>       | <div>Type: <input type="text" value="int list -&gt; int list list -&gt; int list list"/></div> <div>Evaluation: <input type="text" value="[[2]]"/></div>      |
| (g)                                                                                                                    | [2 pts]                                                                                                                                                       |
| <pre>let f a b =   if b &gt; a then (1.3 &lt; 4.6)   else a &lt; true ;;  f true 1.3;;</pre>                           | <div>Type: <input type="text" value="bool -&gt; bool -&gt; bool"/></div> <div>Evaluation: <input type="text" value="ERROR"/></div>                            |
| (h)                                                                                                                    | [2 pts]                                                                                                                                                       |
| <pre>let f x y = match x with   [] -&gt; y    x::xs -&gt; [x]::[] ;;  f [(1,2);(3,4)] [[(6,7)]] ;;</pre>               | <div>Type: <input type="text" value="'a list -&gt; 'a list list -&gt; 'a list list"/></div> <div>Evaluation: <input type="text" value="[[ (1, 2) ]]]"/></div> |
| (i)                                                                                                                    | [2 pts]                                                                                                                                                       |
| <pre>let f a b =   if b &gt; a then ("hello" &lt; "bye")   else a &lt; true ;;  f (fun x -&gt; x &lt; 1) false;;</pre> | <div>Type: <input type="text" value="bool -&gt; bool -&gt; bool"/></div> <div>Evaluation: <input type="text" value="ERROR"/></div>                            |

### 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 ints, and the result of AND'ing the bools.

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)
*)

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

(* Write your code below for calc lst *)

(* 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;;
```