CMSC330 Spring 2024 Quiz 1

Proctoring TA: _____ **Name:** _____

Section Number: _____ **UID:** _____

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 effects	(T)	(F)
Functions are expressions in OCaml	(T)	(F)

Problem 2: OCaml Typing and Evaluating

[Total 6 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:

Evaluation:

(b) [2 pts]

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

f 2.0 false;;
```

Type:

Evaluation:

(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:

Evaluation:

Problem 3: OCaml Expressions [Total 4 pts]

Write an expression that would have the following types.

(a) [2 pts]

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

(b) [2 pts]

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

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
```

CMSC330 Spring 2024 Quiz 1

Proctoring TA: _____ **Name:** _____

Section Number: _____ **UID:** _____

Problem 1: Basics

[Total 4 pts]

	True	False
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)
Functions are expressions in OCaml	(T)	(F)

Problem 2: OCaml Typing and Evaluating

[Total 6 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 = match x with
   [] -> [[3]]
  |x::xs -> [x]::[] ;;

f [4];;
```

Type: _____

Evaluation: _____

(b) [2 pts]

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

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

Type: _____

Evaluation: _____

(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: _____

Evaluation: _____

Problem 3: OCaml Expressions [Total 4 pts]

Write an expression that would have the following types.

(a) [2 pts]

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

(b) [2 pts]

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

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
```

CMSC330 Spring 2024 Quiż 1

Proctoring TA: _____ **Name:** _____

Section Number: _____ **UID:** _____

Problem 1: Basics

[Total 4 pts]

	True	False
Functions are not expressions in OCaml	(T)	(F)
OCaml uses type inference to determine the type of variables	(T)	(F)
Functional Programming aims to decrease the amount of side effects	(T)	(F)
Functional Programming Languages not favor mutable data	(T)	(F)

Problem 2: OCaml Typing and Evaluating

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

Type:

Evaluation:

(b) [2 pts]

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

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

Type:

Evaluation:

(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:

Evaluation:

Problem 3: OCaml Expressions [Total 4 pts]

Write an expression that would have the following types.

(a) [2 pts]

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

(b) [2 pts]

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

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
```

CMSC330 Spring 2024 Quiz 1

Proctoring TA: _____ **Name:** _____

Section Number: _____ **UID:** _____

Problem 1: Basics

[Total 4 pts]

	True	False
Functions are expressions in OCaml	(T)	(F)
Functional Programming aims to decrease the amount of side effects	(T)	(F)
OCaml uses type inference to determine the type of variables	(T)	(F)
Functional Programming Languages favor mutable data	(T)	(F)

Problem 2: OCaml Typing and Evaluating

[Total 6 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
  |_::xs -> [2]::[]
  |4::xs -> [4]::[];;

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

Type:

Evaluation:

(b) [2 pts]

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

f true 1.3;;
```

Type:

Evaluation:

(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:

Evaluation:

Problem 3: OCaml Expressions [Total 4 pts]

Write an expression that would have the following types.

(a) [2 pts]

```
int list -> int -> bool list
```

(b) [2 pts]

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

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
```