1 <3 ocaml

# CMSC330 Spring 2025 Quiz

**Name:** _____ **UID:** _____

**Section Number:** _____ **Proctoring TA:** _____

## Problem 1: Basics

[Total 4 pts]

| | True | False |
|---|---|---|
| In OCaml, the terms values and expressions can be used interchangeably | T | F |
| Using mutable variables can cause side effects | T | F |
| Due to OCaml's type constraints, you cannot make a list of functions | T | F |
| `fold_left`'s accumulator cannot be a tuple | T | F |

## Problem 2: OCaml Typing and Evaluating

[Total 6 pts]

Give the type for the following functions `foo` and give what the following function call evaluates to. **If there is a type error in the function**, put "TYPE ERROR" for the type, and put "ERROR" for the evaluation. If the function call causes an error for any reason, put "ERROR" for the evaluation.

(a) [3 pts]

```
let foo x y =
  match x,y with
    x::xs,y::ys -> y :: xs
  | _ -> [] ;;

foo [] [1;2;3] ;;
```

**Type of foo:**

**Evaluation:**

(b) [3 pts]

```
let foo lst = let total, _ =
    fold_left
    (fun (tot, idx) ele ->
        (tot + ele * idx, idx + 1))
    (0, 0)
    lst in total;;

foo [3;6;9] ;;
```

**Type of foo:**

**Evaluation:**

## Problem 3: Filter

[Total 4 pts]

`filter` is another common higher order function that has type `('a -> bool) -> 'a list -> 'a list`. It applies a function to every item in a list and returns a list of the items that caused the function to return true. **using only `fold` (left or right, given below)**, write a function called `my_filter` which has the same functionality as `filter`. `f` will be the function that returns true or false, and `l` will be the list. **Note:** the original order must be maintained.

```
(* example: my_filter (fun x -> x < 4) [2;3;4;6] = [2;3] *)
let my_filter f l =
```

## Problem 4: Coding

[Total 6 pts]

Write a function call `last_sum` which takes in a `int list list` and returns the sum of the last elements in each `int list`. If list is empty, it adds nothing to the total.

You can write helper functions, you may use the `rec` keyword, **you do not have to use map/fold** (however they are still given). You may not use any List module functions, except those provided. (cons and @ are fine). You may also not use any imperative OCaml.

```
(* last_sum has type int list list -> int *)
(* Examples
    last_sum [] = 0
    (* 3 + 6 + 9 *)
    last_sum [[1;2;3];[4;5;6];[7;8;9]] = 18
    (* 0 + 1 + 3 *)
    last_sum [[];[1];[2;3]] = 4
*)

(* Write your code below *)


 let rec last_sum mtx =
```

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

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

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