Quiz 1 Solutions

Q1 Give the type of the following OCaml expression. If there is a compile error, explain it.

1. `1::[2,3,4]`
2. `fun a b -> 1 + if a b > 0 then b else (a b)`
3. `fun a b -> (fun x -> let _ = b <> x in a)`

Solutions:

1. Type Error, cannot cons `int` and `('a * 'b * 'c) list`
2. `(int -> int) -> int -> int`
3. `'a -> 'b -> 'b -> 'a`

Q2 Give an OCaml expression of the following type without using type annotations.

1. `int -> (int -> 'a) -> 'a`
2. `'a * 'b -> 'b -> 'a`
3. `int * ('a -> int) list`

Solutions:

1. `fun i f -> f (i + 1)`
2. `fun (a, b) x -> let _ = b <> x in a`
3. `1, [fun a -> 1]`

Q3 OCaml Programming

Given functions

For the below questions, you may use the following functions:

- `let rec map f l = match l with [ ] -> [ ] | h :: t -> (f h) :: (map f t)`
- `let rec fold_left f ac l = match l with [ ] -> ac | h :: t -> fold_left f (f ac h) t`
- `let rec fold_right f l ac = match l with [ ] -> ac | h :: t -> f h (fold_right f t ac)`

`sum_even`

Write a function `sum_even`, which takes in an `int list` and returns the sum of all even elements in the list.

you **may not** define the following function as recursive. you also **may not** define a recursive helper function, but you can define as many non-recursive functions as you would like.\ example:

```
sum_even [1;2;3;4;5;6] = 12
sum_even [1;3] = 0
```

```
let sum_even (lst : int list) : (int) =
```

Solution:

```
let sum_even (lst : int list) : int =
  let f (acc : int) (x : int) : int =
    if x mod 2 = 0 then acc + x
    else acc
  in foldl f 0 lst
;;
```

`parity_splitter`

Write a function, `parity_splitter` which takes in an `int list` and returns a tuple with the even numbers as the first tuple element and odd numbers as the second element.

This function **is allowed** to be called recursively.

Example:

```
parity_splitter [1;2;3;4] = [2;4], [1;3]
parity_splitter [1;3] = ([], [1;3])
```

```
let rec parity_splitter (lst : int list) : (int list * int list) =
```

Solution:

```
let pairity_splitter (lst : int list) : ((int list) * (int list)) =
  let f (x:int) (l, r)  : ((int list) * (int list)) =
    if x mod 2 = 0 then (x :: l, r)
    else (l, x :: r)
  in
  foldr f lst ([], [])
```

You can also implement it recursively

```
let rec pairity_splitters_rec lst =
  match lst with
    [] -> ([], [])
  | h :: t ->
    let (l, r) = pairity_splitters_rec t in
    if h mod 2 = 0 then (h :: l, r)
    else (l, h :: r);;
```

Quiz 1 - Redo

Q1[6pts] OCaml types

1. `[[1,2,3]; [4,5,6]; [7,8,9]]`
2. `fun a b -> 1.0 + match (a,b) with (_, _) -> a (a b)`
3. `fun x y -> ( ^ )`

Solution:

1. `(int * int * int) list list`
2. TypeError: `+` expected a float but got an `int`
3. `'a -> 'b -> string -> string -> string`

Q2[6pts] OCaml Expressions

1. `'a -> ('a -> 'b) -> 'b`
2. `('a * 'b * 'c) -> 'a -> 'b -> 'c`
3. `'a -> int * 'b list`

Solution:

1. `|>`
2. `fun (a,b,c) x y -> let _ = (a,b,c) <> (x,y) in c`
3. `(fun _ -> 1, [])`

Q3 OCaml Programming

`sub_evens`

Write a function `sub_evens`, which takes in an `int list` and returns numbers subtracted in the order they appear in the list

You MAY NOT define the following function as recursive. You MAY NOT define a recursive helper, but you may define as many non-recursive helpers as you would like.

Example:

```
sub_evens [16; 1; 2; 3; 4; 6] = 4
sub_evens [1; 3;] = 0
```

```
let sub_evens (lst : int list) =
```

```
let sub_even (lst : int list) : int =
  let f (acc : int) (x : int) : int =
    if x mod 2 = 0 then acc - x
    else acc
  in
  let first_even =
    foldr (fun x acc -> if x mod 2 = 0 then x else acc) lst 0
  in
  foldl f (first_even * 2) lst
;;
```

div_splitter

Write a function `div_splitter` which takes in a non zero `int` and `int list`. The function returns a tuple with an `int list` of the numbers divisible by the `int` as the first element and an `int list` the non divisible numbers as the second element.

This function MAY be called recursively. Example:

```
div_splitter 2 [1;2;3;4;5] = ([2;4], [1;3;5])
div_splitter 5 [1;2;3;4] = ([], [1; 2; 3; 4])
div_splitter 7 [] = ([], [])
```

```
let div_splitter (d: int) (lst: int list) : int list * int list =
```

Solution:

```
let div_splitter (d : int) (lst : int list) : ((int list) * (int list)) =
  let f (x:int) (l, r)  : ((int list) * (int list)) =
    if x mod d = 0 then (x :: l, r)
    else (l, x :: r)
  in
  foldr f lst ([], [])
;;
```

```
let rec div_splitters_rec (d : int) (lst : int list) : int list * int list =
  match lst with
    [] -> ([], [])
  | h :: t ->
    let (l, r) = div_splitters_rec d t in
    if h mod d = 0 then (h :: l, r)
    else (l, h :: r);;
```