CS 320 Theory Homework 2

Due: October 11th @ 11:59 PM EST on Gradescope

There is no late due date for this assignment

Read the following information below for this assignment

Krumbl Kookies

This sensational cookie brand is working on making new flavors for its next week special! A client has proposed the following cookie flavors:

```
type cookie_details = float * float

type cookie_flavor =
   | CaramelPumpkin of cookie_details
   | LaddooLemon of cookie_details
   | Nevadito of cookie_details
```

Where `cookie_details` contains the cookie's diameter (cm) and price ($) respectively.

Question 1.

Instructions

Your objective is to exhaustively match all patterns of the following expressions using only the `match` keyword. You should match *until there are only base types* (`int, float, bool, string`). A wildcard (`_`) should be used to match the right-hand side of the cons (`_::_`) pattern.

Below are three examples that attempt to illustrate what satisfies our requirements.

Wrong Example 1

```
e: (int * bool) list

match e with
| [] -> ...
| a :: _ -> ...
```

This is *wrong* because `a` is a `tuple` which is *not* a base type, so it must be simplified further.

Correct Example 1

```
e: (int * bool) list
match e with
| [] -> ...
| (a, b) :: _ -> ...
```

Correct, because `a` and `b` in the tuple's pattern `(a, b)` are both base types – `a` is an `int` and `b` is a `bool` respectively.

Correct Example 2

```
x: (int * bool) list option

match x with
| None -> ...
| Some l -> (
    match l with
    | [] -> ...
    | (a, b) :: _ -> ...)
```

Correct, because both constructors of `x` have been matched. In the case of the `(_::_)` pattern, the head element of type `(int * bool)` has been matched against its left and right components. The components `a` has type `int` and `b` has type `bool`, which are base types.

Match the following variables

1.1

```
n: cookie_flavor
    match n with
```

1.2

```
w: int list option list
    match w with
```

1.3

```
x: (int * (bool list * string)) option
    match x with
```

Question 2.

For this question, you may include the types `cookie_flavor` *and* `cookie_details` *in your answer* ***if appropriate****.*

Consider a function with polymorphic type `f : ‘a -> ‘a * ’a -> (‘a * ’a) list`

2.1. What is the type of `f (CaramelPumpkin(1., 1.))` ? **Justify your answer.**

2.2. What is the type of the following statement

```
let cd: cookie_details = (0., 0.) in
   fun x -> f x cd
```

Justify your answer.

Question 3.

For this question, you may include the types `cookie_flavor` *and* `cookie_details` *in your answer* ***if appropriate****.*

Consider a function with polymorphic type f : (‘a * ‘b) -> (‘b -> ‘a) -> ‘a

3.1. What is the type of `f (Nevadito(1., 2.), true)` **? Justify your answer.**

3.2. What is the type of `fun coo kie -> f (coo, kie) ?` **Does it have the same type as** `f`**? Justify both answers.**

Question 4.

Consider a function with polymorphic type `g : (‘a -> ‘b -> ‘c) -> ‘d`

4.1. What is the type of `g (List.fold_right)` **? Justify your answer.**
Hint: this is the type signature: `List.fold_right: (’a -> ’b -> ’b) -> ’a list -> ’b -> ’b`

4.2. What is the type of `g (fun acc (e: cookie_flavor) -> List.cons e acc)`**? Briefly explain your reasoning.**

Question 5.

Is the following function well-typed? Briefly explain your reasoning.

```
let mystery (x : cookie_flavor list) (y: cookie_details * bool): cookie_flavor list =
   match x, y with
   | [], ((diam, price), b) -> List.fold_right (List.cons) [] x
   | h::t, (_, true) -> []
   | _::_, (_, false) -> [Nevadito(y)]
```

Question 6.

Instructions
Krumbl actually finds these flavors quite lovely and wants to implement them in a factory! To do so, they need to implement the following functions:

```
reverse: cookie_flavor list -> cookie_flavor list

append: cookie_flavor list -> cookie_flavor list -> cookie_flavor list

concat : (cookie_flavor list list) -> cookie_flavor list

filter: (cookie_flavor -> bool) -> cookie_flavor list -> cookie_flavor list

fold_right : (cookie_flavor -> 'b -> 'b) -> cookie_flavor list -> 'b -> 'b
```

You may wonder *"why can't they use the already-made* `List` *library functions?"* Well, unfortunately, Krumbl is a low-budget Crumbl, so they were only able to afford `List.fold_left`

```
List.fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
```

(You can only use this library function in addition to basic OCaml features including `let, fun, match, if, else, ...`)

However, you, as a beloved student of CS320, are strongly determined to help Krumbl conduct these factory operations.

You are welcome to use your created functions to implement others – for example, you could use `filter` when implementing `fold_right`, **but not vice-versa** because `filter` is implemented before `fold_right`.

Implement the following standard list functions. When given the same input, they should have the same output as their standard library counterparts.

6.1. reverse : cookie_flavor list -> cookie_flavor list

6.2. append : cookie_flavor list -> cookie_flavor list -> cookie_flavor list

6.3. concat : (cookie_flavor list list) -> cookie_flavor list

6.4. filter : (‘a -> bool) -> ‘a list -> ‘a list

6.5. fold_right : (cookie_flavor -> ’b -> ’b) -> cookie_flavor list -> ’b -> ’b

Question 7.

Instructions
Krumbl now wants you to make a `create_cookie_boxes` function that will take three parameters: a cookie box, the diameter of the cookie (cm) and the price of a cookie in dollars (Each cookie will have the same diameter and price)

```
fix_cookie_box: cookie_flavor list -> float -> float -> cookie_flavor list list
```

For each element in the `cookie_flavor list`, create a new list of 2 cookies consisting of the same flavor, but using the new diameter and new price provided.

For example,

```
fix_cookie_box [LaddooLemon (55.4, 66.9); CaramelPumpkin (77.3, 88.3)] 1.2 3.4 =
[[LaddooLemon (1.2, 3.4); LaddooLemon (1.2, 3.4)];[CaramelPumpkin (1.2, 3.4); CaramelPumpkin (1.2, 3.4)]]
```

You can use the functions declared above in 6.1 - 6.5 and still have access to `List.fold_left`

7.1 Create the function

Question 8.

8.a) Given the following mystery1 function, what is the type of the mystery1 function?. You must also explain your answer. Failure to explain your answer or giving incorrect explanation will result in zero credits

```
let rec mystery1 (f, l1, l2) =
   match (l1, l2) with
   | ([], []) -> []
   | (_, []) -> []
   | ([], _) -> []
   | (h1::t1, h2::t2) -> [f (Some (LaddooLemon(h1, h2)))] :: mystery1 (f, t1, t2)
```

8.b) Given the following mystery2 function, what is the type of the mystery2 function? You must also explain your answer. Failure to explain your answer or giving incorrect explanation will result in zero credits

```
   let rec mystery2 f l1 l2 =
      match l1, l2 with
      | [], [] -> []
      | _, [] -> []
      | [], _ -> []
      | Some h1::t1, LaddooLemon h2::t2 ->
        (
            Some (h1 (Some (LaddooLemon(4.1, 5.3)))) :: mystery2 f t1 t2
        )
      | _, _ -> []
```