Lab # 1

Due: 2024-09-27

By: Omar Abdul, Anas Taimah, Yusuf Khan
300228700, 300228842, 300293842

Course: CSI3120

Group: 40

Code explanation:

Question 1:
The function map2 takes the input lst1 and lst2 as well as a function f (+, -, %, etc.) that applies to the items in the list. In the base case, if both lists are empty, it returns an empty list. In the second case if both lists are greater than size 0, it takes the head of each list and defines the tail of each list "(x::xs, y::ys)" and applies the function to it "(f x y)" to create the head of a new list being returned. The tail of this new list is then recursively defined by calling the map2 function on the tail of the list until it is empty. The third case of this function happens when the lists are of two different lengths.

Question 2:

This function filter_even takes a list as an input and filters out all the odd numbers from it. It uses List.filler which is a built in function that takes a function and the list. It goes through each element x in the list and applies the predicate (fun x -> x mod 2 = 0) to check if x is divisible by 2. If the predicate returns true, x is kept in the list and if the predicate returns false, x is removed from the list.

Question 3:

The first part is the function compose_functions which takes 2 functions as an argument. The function then defines a new function "f (g(x))" which applies function g on input x and then applies function f to that result.

The second part is creating a variable called composed which calls the compose_functions on an input with two defined functions f (fun x -> x * 2) & g (fun y -> y + 3). The defined function f multiplies by 2 and the defined function g adds 3.

When combined it means the input is first incremented by 3 and then multiplied by 2 to give us our final result. Composed 5 = 2(5 + 3) = 16

Question 4:
A recursive function "reduce" is defined which takes 3 inputs (a function, an accumulator, a list). The base case checks if the list input is empty which returns the accumulator value. The second case when the list is not empty splits the list into head x and tail xs, and then calls the reduce function recursively with three new inputs; 1) same function 2) a new accumulator which is calculated by applying the function on the accumulator and the head 3) the tail of the list.

Once all the recursive calls are done and the tail list is empty, the accumulator is returned.

Test Cases:

Question 1:

Test 1:

```
# map2 (fun x y -> x + y) [1; 2; 3] [4; 5; 6];;
- : int list = [5; 7; 9]
```

Test 2:

```
# map2 (fun x y -> x + y) [3; 8; 1] [2; 1; 4];;
- : int list = [5; 9; 5]
```

Question 2:

Test 1:

```
# let even_numbers = filter_even [1; 2; 3; 4; 5; 6];;
val even_numbers : int list = [2; 4; 6]
```

Test 2:

```
# let even_numbers = filter_even [2; 2; 7; 1; 6; 8];;
val even_numbers : int list = [2; 2; 6; 8]
```

Question 3:

Test 1:

```
# composed 5;;
- : int = 16
```

Test 2:

```
# composed 6;;
- : int = 18
```

Question 4:

Test 1:

```
# let result = reduce (fun x y -> x + y) 0 [1; 2; 3; 4];;
val result : int = 10
```

Test 2:

```
# let result = reduce (fun x y -> x + y) 0 [4; 1; 7; 2];;
val result : int = 14
```

All Test Cases Screenshot:

```
omarabdul@Omars-Laptop CSI3120-Labs % ocamlc -o lab1 lab1.ml
omarabdul@Omars-Laptop CSI3120-Labs % ocaml

  OCaml version 4.14.0
  Enter #help;; for help.

  # #use "lab1.ml";;
  val map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list = <fun>
  val filter_even : int list -> int list = <fun>
  val compose_functions : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
  val composed : int -> int = <fun>
  val reduce : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
  # map2 (fun x y -> x + y) [1; 2; 3] [4; 5; 6];;
  - : int list = [5; 7; 9]
  #
    map2 (fun x y -> x + y) [3; 8; 1] [2; 1; 4];;
  - : int list = [5; 9; 5]
  # let even_numbers = filter_even [1; 2; 3; 4; 5; 6];;
  val even_numbers : int list = [2; 4; 6]
  # let even_numbers = filter_even [2; 2; 7; 1; 6; 8];;
  val even_numbers : int list = [2; 2; 6; 8]
  # composed 5;;
  - : int = 16
  # composed 6;;
  - : int = 18
  # let result = reduce (fun x y -> x + y) 0 [1; 2; 3; 4];;
  val result : int = 10
  # let result = reduce (fun x y -> x + y) 0 [4; 1; 7; 2];;
  val result : int = 14
  #
```

ChatGPT declaration:

Our group used ChatGPT throughout the lab to better understand the program application so that we are able to develop our answers. ChatGPT was used only for assistance to learn and understand how it worked and not to generate any answers or results. All solutions were implemented in our own words and approach.