CSI 3120

Lab 1 Report

Group #6

Carl Li 300235679
Ruoyu Liu 300176134
Roy Rui 300176548

Question 1:

The program uses map2() to apply the sum of the corresponding elements from two lists, adds the number into a new list named result, and then outputs the new list.

The function map2() is defined in the program as taking two lists as input and returning a new list. The function adds the first elements from l1 and l2, adds them to the new list, and then recurses the rest of l1 and l2 until the two lists are empty. If l1 and l2 have unequal length, the function will raise an exception called Unequal _lengths and shut the program

```
exception Unequal_lengths

let rec map2 f l1 l2 =
  match (l1, l2) with
  | ([], []) -> []
  | (head::tail, head2::tail2) -> (f head head2)  :: (map2 f tail tail2)
  | _ -> raise Unequal_lengths
```

Reference: Line 1. “exception Unequal_lengths” was assisted by ChatGPT as knowledge.

Test Case:

1.
```
let () =
  let result = map2 (fun x y -> x + y) [1;2;3] [4;5;6] in
  Printf.printf "int list = [";
  Printf.printf "%s" (String.concat ";" (List.map string_of_int result));
  Printf.printf "]\n"
```

```
PS A:\GitHub\CSI3120_Labs\Lab1\lab101> opam exec -- dune exec lab101
int list = [5;7;9]
```

2.
```
let () =
  let result = map2 (fun x y -> x + y) [10;20;30] [47;58;69] in
  Printf.printf "int list = [";
  Printf.printf "%s" (String.concat ";" (List.map string_of_int result));
  Printf.printf "]\n"
```

```
PS A:\GitHub\CSI3120_Labs\Lab1\lab101> opam exec -- dune exec lab101
int list = [57;78;99]
```

3.

```
let () =
  let result = map2 (fun x y -> x + y) [15;25;35] [15;25;35] in
  Printf.printf "int list = [";
  Printf.printf "%s" (String.concat ";" (List.map string_of_int result));
  Printf.printf "]\n"
```

```
PS A:\GitHub\CSI3120_Labs\Lab1\lab101> opam exec -- dune exec lab101
int list = [30;50;70]
```

Question 2:

The function filter_even accepts a list named l1, and the program iterates l1 from the beginning to the end. It checks every number in l1 and adds it to a new list if it's even. Then it reruns itself with the rest of l1 until l1 is empty, and finally returns the latest list.

```
let rec filter_even l1 =
  match l1 with
  | [] -> []
  | head :: tail ->
      if head mod 2 = 0 then head :: filter_even tail
      else filter_even tail
```

Test Case:

1.
```
let () =
  let result = filter_even [1; 2; 3; 4; 5; 6; 7; 8] in
  Printf.printf "The int list = [";
  Printf.printf "%s" (String.concat ";" (List.map string_of_int result));
  Printf.printf "]";
  Printf.printf "\n"
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab102> opam exec -- dune exec lab102
The int list = [2;4;6;8]
```

2.
```
let () =
  let result = filter_even [3; 7; 12; 1; 9; 5; 14; 6] in
  Printf.printf "The int list = [";
  Printf.printf "%s" (String.concat ";" (List.map string_of_int result));
  Printf.printf "]";
  Printf.printf "\n"
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab102> opam exec -- dune exec lab102
The int list = [12;14;6]
```

3.
```
let () =
  let result = filter_even [20; 4; 17; 8; 15; 2; 11; 19] in
  Printf.printf "The int list = [";
  Printf.printf "%s" (String.concat ";" (List.map string_of_int result));
  Printf.printf "]";
  Printf.printf "\n"
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab102> opam exec -- dune exec lab102
The int list = [20;4;8;2]
```

Question 3:

The function compose_functions accepts 2 functions f and x, and a variable x as parameters. It calculates g(x) first and uses the output of g(x) as the input of f(x), then the program creates an instance of compose_functions named composed and assigns f(x) = 2x, g(x) = x + 3. Then the program calls the function composed with an input, and the function will calculate the result of f(g(x)).

```
let compose_functions f g x = f (g x)
let composed = compose_functions (fun x -> x * 2) (fun y -> y + 3);;
```

Test Case:

1.
```
let result = composed 5 in
Printf.printf "int = %d\n" result
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab103> opam exec -- dune exec lab103
int = 16
```

2.
```
let result = composed 17 in
Printf.printf "int = %d\n" result
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab103> opam exec -- dune exec lab103
int = 40
```

3.
```
let result = composed 189 in
Printf.printf "int = %d\n" result
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab103> opam exec -- dune exec lab103
int = 384
```

Question 4:

The function accepts a function f and two numbers as parameters. It recurses from beginning to end until it's empty. In each recursion, the function does f(acc, head), which should add them together, and uses the number as the new acc. The function then adds the acc with the rest of the list until the end of the list.

```
let rec reduce f acc lst =
  match lst with
  |[] -> acc
  |head :: tail->
    reduce f (f acc head) tail
```

Test Case:

1.
```
let () =
  let result = reduce (fun x y -> x + y) 0 [1;2;3;4] in
  Printf.printf "int = %d\n" result
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab104> opam exec -- dune exec lab104
int = 10
```

2.
```
let () =
  let result = reduce (fun x y -> x + y) 0 [2;2;3;4] in
  Printf.printf "int = %d\n" result
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab104> opam exec -- dune exec lab104
int = 11
```

3.
```
let () =
  let result = reduce (fun x y -> x + y) 14 [3;2;3;4] in
  Printf.printf "int = %d\n" result
```
```
PS A:\GitHub\CSI3120_Labs\Lab1\lab104> opam exec -- dune exec lab104
int = 26
```