

CIS 120 - Lab 4

Assume that you have access to ML implementations (Set1 and Map1) of the following interfaces.

```
module type Set = sig
  type 'a set
  val empty : 'a set
  val is_empty : 'a set -> bool
  val mem : 'a -> 'a set -> bool
  val add : 'a -> 'a set -> 'a set
  val union : 'a set -> 'a set -> 'a set
  val remove : 'a -> 'a set -> 'a set
  val list_to_set : 'a list -> 'a set
  val equal : 'a set -> 'a set -> bool
  val elements : 'a set -> 'a list
end

module Set1 : Set = ...

module type Map = sig
  type ('k,'v) map
  val empty : ('k,'v) map
  val is_empty : ('k,'v) map -> bool
  val mem : 'k -> ('k,'v) map -> bool
  val find : 'k -> ('k,'v) map -> 'v
  val add : 'k -> 'v -> ('k,'v) map -> ('k,'v) map
  val remove : 'k -> ('k,'v) map -> ('k,'v) map
  val keys : ('k,'v) map -> 'k list
  val values : ('k,'v) map -> 'v list
  val bindings : ('k,'v) map -> ('k*'v) list
end

module Map1 : Map = ...
```

1. Consider the following tests for the Set1 module. Which ones pass? Which ones cause errors? What is the cause of the errors?

```
;; open Assert
;; open Set1

(* a *)
let test () : bool =
  let set1 = add 1 empty in
  let set2 = add 2 set1 in
  let set3 = add "x" set2 in
  mem "x" set3
;; run_test "a" test

(* b *)

let test () : bool =
  let set1 = add 1 empty in
  let set2 = add 2 set1 in
  let x = 1 in
    let set3 = add x set2 in
    mem 1 (remove 1 set3)
;; run_test "b" test

(* c *)
let test () : bool =
  let set1 = add 1 empty in
  let set2 = add 2 set1 in
  let x = 1 in
    mem x (add "x" set2)
;; run_test "c" test

(* d *)
let test () : bool =
  let set1 = add 1 empty in
  let set2 = add 2 set1 in
  mem 2 set1
;; run_test "d" test
```

2. Now suppose that you have opened an implementation of the Map interface. Which of these tests pass?

```
;; open Map2

let test () : bool =
  let map1 = add 1 2 empty in
  let map2 = add 1 3 map1 in
  find map1 1 = 3
;; run_test "a" test

(* b *)
let test () : bool =
  let map1 = add 1 2 empty in
  mem 2 (add 2 2 map1)
;; run_test "b" test

(* c *)
let test () : bool =
  let map1 = add 1 2 empty in
  let map2 = remove 1 map1 in
  mem 1 (add 1 2 map2)
;; run_test "c" test

(* d *)
let test () : bool =
  let map1 = add 1 2 empty in
  let map2 = add 2 3 map1 in
  find 3 map2 = 2
;; run_test "d" test
```

3. Write a function which, given a string list, converts it into a (int, string) map where the key represents the index of the element in the list and the value represents the value of the element in the list. For example, if passed {"A"; "B"; "C"; "D"}, this function should return [1 -> "A", 2 -> "B", 3 -> "C", 4 -> "D"]. **Don't forget to write test cases.** Feel free to write helper functions.

```
let rec index_to_value_map (x : string list) : (int, string) map =
```

4. Write a function which, given an int list and an int set, returns an int list containing only those values in the original int list that AREN'T in the int set. For example, given [1;2;3;4] and {2,3}, this function should return [1;4].

```
let rec filter (x : int list) (y : int set) : int list =
```

5. Write a function which, given a (string, int) map, finds all the strings bound to even values and returns them as a list. For example, given the map [{"a" -> 2, "y" -> 3, "ab" -> 4}], it should return {"a"; "ab"}

```
let rec bound_to_even_values (x: (string, int) map) : string list =
```

6. Write a function which, given a ('k, 'v) map, returns a ('v, 'k set) map. For examples, given the map [{"A" -> 1, "B" -> 2, "C" -> 2}], it should return [1 -> {"A"}, 2 -> {"B", "C"}].

```
let rec reverse_map (x: ('k, 'v) map) : ('v, 'k set) map =
```