Problems

Section 05

Problem 1

Module Implementations: Consider the Ocaml code below that defines a module signature for a simple list library

```
module type LIST_LIB = sig

  (*
    map f lst applies function f to each element of lst, returning a
list of the results.
  *)
  val map: ('a -> 'b) -> 'a list -> 'b list

  (*
    filter f lst returns a list of elements in lst for which f returns
true.
  *)
  val filter: ('a -> bool) -> 'a list -> 'a list

end
```

For each implementation of this signature below, state whether it type checks and if it implements the module correctly.

```
module Llib1: LIST_LIB = struct

  let rec map f l =
    match l with
    | [] -> []
    | hd :: tl -> f hd :: map f tl

  let rec filter f l =
    match l with
    | [] -> []
    | hd :: tl -> if f hd then hd :: filter f tl else filter f tl

end
```

Typechecks (T / F): _____ Correct (T / F): _____

```
module Llib2: LIST_LIB = struct

  let rev l =
    let rec loop l acc =
      match l with
      | [] -> acc
      | hd :: tl -> loop tl (hd :: acc)
    in loop l []

let rec map f l =
    match l with
    | [] -> []
    | hd :: tl -> f hd :: map f tl

  let rec filter f l =
    match l with
    | [] -> []
    | hd :: tl -> if f hd then hd :: filter f tl else filter f tl

end
```

Typechecks (T / F): _____ Correct (T / F): _____

```
module Llib3: LIST_LIB = struct

 let rec map f l =
    match l with
    | [] -> []
    | hd :: tl -> f hd :: map f tl

 let rec filter f l =
   match l with
   | [] -> []
   | hd :: tl -> if f hd then filter f tl else hd :: filter f tl

end
```

Typechecks (T / F): _____ Correct (T / F): _____

```
module Llib4: LIST_LIB = struct

  let rev l =
    let rec loop l acc =
      match l with
      | [] -> acc
      | hd :: tl -> loop tl (hd :: acc)
    in loop l []

  let map f l =
    let rec loop l acc =
      match l with
      | [] -> rev acc
      | hd :: tl -> loop tl (f hd :: acc)
  in loop l []

end
```

Typechecks (T / F): _____ Correct (T / F): _____

Problem 2

Module Representation Invariants: Consider the OCaml code below that defines a signature for a module that implements a list where all elements are positive integers.

```
module type POSITIVE_LIST = sig
  (* This is a signature for a module that provides an implementation
     for a list of ints where each int is positive. It should be
     impossible for the user to have a list with any non-positive
     elements using this module (this is our representation invariant)
  *)

  type t

  val empty_posList: t

  val cons: int -> t -> t

  val map: (int -> int) -> t -> t

end
```

For each implementation of this signature below, state whether it implements the module correctly.

```
module PL1: POSITIVE_LIST = struct

  type t = int list

  let empty_posList = []

  let cons a l = a :: l

  let map = List.map

end
```

Correct (T / F): _____

```
module PL2: POSITIVE_LIST = struct

  type t = int list

  let empty_posList = []

  let cons a l = if a > 0 then a :: l
                 else failwith "Non-positive input"

  let rec map f l =
    match l with
    | [] -> []
    | hd :: tl -> (f hd) :: map f tl

end
```

Correct (T / F): _____

```
module PL3: POSITIVE_LIST = struct

  type t = int list

  let empty_posList = []

  let cons a l = if a > 0 then a :: l
                 else failwith "Non-positive input"

  let rec map f l =
    match l with
    | [] -> []
    | hd :: tl -> cons (f hd) (map f tl)

end
```

Correct (T / F): _____