COMP 3958: Lab 2

Submit a file named `lab2.ml` containing your source code. Unless explicitly specified what you must do, you may use the `hd`, `tl`, `length`, `rev`, `filter`, `map` and `sort` from the `List` module. Your file must compile without warnings or errors. If not, you may receive no credit for this lab exercise. As before, comment each function. Write at least 3 tests for each function in question 3. Maximum score: 15.

1. Implement the following list functions with the given signatures using recursion and without calling functions from other modules except for `List.rev`.

   (a) `val drop_while :  ('a -> bool) -> 'a list -> 'a list`
   `drop_while f lst` returns `lst` with leading elements satifying `f` (i.e., element `e` where `f e` is true) dropped. For example,

   `drop_while (fun x -> x mod 2 = 0) [4;2;6;7;6;8;1]` returns `[7;6;8;1]`

   Note that in the example, leading even integers are dropped.

   (b) `val zip_with :  ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list` `zip_with f lst1 lst2` returns a list whose elements are obtained by applying `f` to corresponding elements of `lst1` and `lst2`. If the two lists have different lengths, it stops at the end of the shorter list. For example,

   `zip_with ( + ) [4;2;6;7;6;8] [3;2;-1;1]` returns `[7;4;5;8]`

   Provide a tail-recursive implementation.

2. (a) Recall that `map` applies a function to each element of a list to get a new list. Its signature is

   `val map : ('a -> 'b) -> 'a list -> 'b list`

   Implement from basics a tail-recursive version of the function `mapi` with signature

   `val mapi : (int -> 'a -> 'b) -> 'a list -> 'b list`

   that is similar to `map`, except that the function (of type `int -> 'a -> 'b`) passed to `mapi` is applied to the index (starting from 0) as well as the value of each element of a list to get a new list.
   For example, `mapi (fun i x -> (i, x)) ["homer"; "ned"; "monty"] returns [(0, "homer"); (1, "ned"); (2, "monty")]`

   (b) Using `mapi` (and some `List` functions), implement a function `every` with signature

   `val every : int -> 'a list -> 'a list`

   so that `every n lst` returns a list consisting of every `n`-th element of `lst`. (It is a precondition that `n` be positive.) For example, `every 3 [1;2;3;4;5;6;7]` returns [3;6] (every third element).

3. (a) Using either `List.fold_left` or `List.fold_right`, implement the `dedup` function from lab 1.

   (b) i. Using either `List.fold_left` or `List.fold_right`, implement a function `group` with signature

   `val group : 'a list -> 'a list list`

   that groups consecutive identical elements in a list into a sub-list. For example,

   `group [12; 34; 34; 34; 5; 12; 12; 6; 78; 90; 90]`

   returns:

   `[[12]; [34; 34; 34]; [5]; [12; 12]; [6]; [78]; [90; 90]].`

   ii. Using `group` (and some `List` functions), implement a function `frequencies` with signature

   `val frequencies : 'a list -> ('a * int) list`

   that, given a list of elements, counts how many times each element occurs in the list. `frequencies` returns a list of pairs, where each pair is of the form `(elt, count)`, where `count` is the number of times the element `elt` occurs in the list. The order of the pairs in the returned list is unspecified, i.e, they can be in any order. For Example,

   `frequencies [23; 12; 15; 12; 45; 15; 13; 45; 15; 12; 15; 15]`

   returns something like

   `[(12, 3); (13, 1); (15, 5); (23, 1); (45, 2)]`