COMP 3958: Lab 2

Submit a file named `lab2.ml` containing your source code. For the standard library `List` module, you are only allowed to use `hd`, `tl`, `filter`, `map`, `rev`, `length`, `fold_left` and `fold_right`. Your file must compile without warnings or errors. If not, you may receive no credit for this lab exercise. Maximum score: 15.

1. (a) Implement a function `min_elt` with signature

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

   that returns the smallest element in a non-empty list. (It is unspecified what happens if the list is empty.)

   (b) Implement a function `remove` with signature

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

   so that `remove x l` removes the first occurrence of `x` in the list `l`, i.e., it returns a list that is the same as `l` except with the first occurrence of `x`, if any, removed.

   (c) Using (a) and (b), implement a form of "selection sort" to sort a list (in ascending order). The function has signature

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

   The idea of selection sort is to select the smallest element of the list and move it to the front, then repeat the step for the rest of the list. (Clearly, you are not allowed to use `List.sort`.)

2. (a) Recall that `map` has signature

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

   and that it applies a function to each element of a list to get a new list.
   Implement from basics a function `mapi` with signature

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

   that is essentially the same as `map`, except that the function `mapi` takes (type `int -> 'a -> 'b`) 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` together with some other functions, implement a function `every` with signature

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

   so that `every n lst` returns a list consisting of the elements of `lst` whose index is a multiple of `n`. Note that `n` must be positive — it is unspecified what happens if it is not. For example, `every 3 [1;2;3;4;5;6;7]` returns [1;4;7].

3. (a) Recall that fold_left has signature

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

   Implement from basics a function `fold_until` that has signature

   ```
   val fold_until : ('a -> 'b -> 'a option) -> 'a -> 'b list -> 'a
   ```

   `fold_until f init lst` is a short-circuiting version of `fold_left`. If `f a b` returns `None`, the computation stops and returns `a`; however, if `f a b` returns `Some c`, then `c` is used as the first argument to `f` in the recursive call.

   (b) Use `fold_until` to implement a function `sum_until_nonpos` that sums a list of integers until a non-positive integer is encountered (or until the end of the list). The signature of `sum_until_nonpos` is

   ```
   val sum_until_nonpos : int list -> int
   ```