exercise07-p

December 1, 2018

0.0.1 Assignment 7.1 (P) The List Module (part 2)

In order to write short and easy to understand programs, it is crucial to use existing library functions for regularly appearing problem patterns. The most frequently used library functions are clearly those of the `List` Module and the following in particular. Check the documentation for their types and functionality:

- `List.map`
- `List.fold_left`
- `List.fold_right`
- `List.find_opt`
- `List.filter`

Implement the following functions without using any recursive functions:

1) `squaresum : int list -> int` computes $\sum_{i=1}^{n} x_i^2$ for a list $[x_1, \dots, x_n]$.

2) `float_list : int list -> float list` converts all ints in the list to floats.

3) `to_string : int list -> string` builds a string representation of the given list.

4) `part_even : int list -> int list` partitions all even values to the front of the list.

```
In [ ]: let squaresum l = (* TODO *)

        let float_list l = (* TODO *)

        let to_string l = (* TODO *)

        let part_even l = (* TODO *)
```

0.0.2 Assignment 7.2 (P) Mappings

There is a need to represent and modify a mapping from one set of values to another set of values in many applications. Remember how we stored the grades in the student record. Storing a list of pairs `'a * 'b` is a simple way to represent a mapping from type `'a` to type `'b`. This kind of list is typically referred to as an associative list:

1) Implement these functions to work with mappings based on associative lists:

- `is_empty : ('k * 'v) list -> bool`
- `get : 'k -> ('k * 'v) list -> 'v option`
- `put : 'k -> 'v -> ('k * 'v) list -> ('k * 'v) list`
- `contains_key : 'k -> ('k * 'v) list -> bool`
- `remove : 'k -> ('k * 'v) list -> ('k * 'v) list`
- `keys : ('k * 'v) list -> 'k list`
- `values : ('k * 'v) list -> 'v list`

```
In [ ]: let is_empty m = (* TODO *)

        let rec get k = (* TODO *)

        let put k v m = (* TODO *)

        let contains_key k m = (* TODO *)

        let rec remove k = (* TODO *)

        let keys m = (* TODO *)

        let values m = (* TODO *)
```

2) Check the `List` module for functions that already provide (some of) these functionalities.

An alternative to associative lists is to use functions of type `'k -> 'v option` directly. So for example, the function `fun x -> x * x + 1` respresents a very efficient mapping from any number to the successor of its square.

3) Implement the above functions again for mappings based on functions. Some of these functions cannot be implemented, however:

```
In [ ]: let is_empty m = (* TODO *)

        let get k m = (* TODO *)

        let put k v m = (* TODO *)

        let contains_key k m = (* TODO *)

        let remove k m = (* TODO *)

        let keys m = (* TODO *)

        let values m = (* TODO *)
```

4) Discuss: What are the advantages of either approach? When would you use which?

0.0.3 Assignment 7.3 (P) Operator Functions

In OCaml, infix notation of operators is just syntactic sugar for a call to the corresponding function. The binary addition + merely calls the function `(+) : int -> int -> int`.

1) Discuss why this is a very useful feature.