Lab – 6

Week of Feb. 18, 2018

1 Instructions

1. Use the OCaml top-level to develop and debug your code.
2. You may assume that all inputs are valid unless otherwise stated in the problem.
3. In questions that require string outputs, be careful not to include leading or trailing whitespace.
4. You may submit and evaluate your code a *maximum* of 15 times without penalty. Subsequently, you will lose 2 marks per additional evaluation. Therefore, please ensure that you have thoroughly debugged your code before submitting.

Hint - Using OCaml's List package, specifically the 'map' and 'fold' functions for parts 1-5, should result in significantly shorter code

The following submission files are *required*:

1. `all_even.ml`
2. `even_odd.ml`
3. `add_num.ml`
4. `merge_list.ml`
5. `num_distinct.ml`
6. `matrix_add.ml`
7. `matrix_mult.ml`

2 Assignments

1. `all_even: int list -> int list`. Given a list of non-negative integers, write a function to return a list of even numbers in the given list (in the same order). Example:
   (a) `all_even [1;2;3;4]` should return `[2;4]` (and not `[4;2]`).
   (b) `all_even [1;3;5]` should return `[]`.

2. `even_odd: int list -> bool`. Given a list containing non-negative integers, write a function to return true only if the sum of even numbers is equal to the sum of odd numbers. Otherwise, return false.

   (a) `even_odd [1;2;3;2]` returns `true`.

   (b) `even_odd [1;2]` returns `false`.

   (c) `even_odd []` returns `true`.

3. `add_num: int list -> int -> int list`. Given a list of integers and an integer k, write a function to add multiples of k as follows. Add `1` $\times$ `k` to the first element, `2` $\times$ `k` to the second element, etc. Examples:

   (a) `add_num [1;1;1] 2` should return `[3;5;7]`

   (b) `add_num [1;2;3] 3` should return `[4;8;12]`

4. `merge_list: int list * int list -> int list`. Given two lists of integers, the function should first check if each of the lists are sorted in *descending* order. If either list is not sorted, then it should raise the error `List_Sorting_Error` with the string `"Input lists are not sorted!"`. If both lists are correctly sorted, then the function should return a *single* new list in which the integers are sorted in *ascending* order. Examples:

   (a) `merge_list ([3;2;1], [8;2;1])` should return `[1;1;2;2;3;8]`

   (b) `merge_list ([6;1], [4;3;2])` should return `[1;2;3;4;6]`

   (c) `merge_list ([3;5;4], [4;3;2;1])` should raise the `List_Sorting_Error`

5. `num_distinct: int list -> (int * int) list`. Given a list of integers which is sorted in *descending* order and potentially contain duplicates, write a function to count the number of *distinct values* that are in the list and the number of times they occur in the form `(value, #occurrences)`. Use the `List.sort` function to sort the output list according to `#occurrences` in *descending* order. If the number of occurrences of two values are the same, then sort according to the value in *ascending* order (**Note:** You will need to implement your own sort function). Examples:

   (a) `num_distinct [5;4;4;3;2;2;2;1]` returns `[(2,3);(4,2);(1,1);(3,1);(5,1)]`

   (b) `num_distinct [5;4;3]` returns `[(3,1);(4,1);(5,1)]`

6. `matrix_add: int list list -> int list list -> int list list`. A matrix can be represented as a list of lists. Suppose we assume that a matrix has been represented in *row major order* (that is, row-at-a-time), write a function that adds two matrices and returns their sum (another matrix in row major order). Ensure that you do error checking for dimensions(should be positive), both *within* the matrix, as well as *between* the matrices and raise one of two errors `Not_a_Matrix` and `Non_matching_dimensions`.

   (a) `matrix_add [[1;2];[3;4]] [[0;0];[0;0]]` returns `[[1;2];[3;4]]`

   (b) `matrix_add [[1;1];[2;2]] [[1;5];[7;8]]` returns `[[2;6];[9;10]]`

(c) `matrix_add [[1;1;2];[2;2]] [[0;0];[0;0]]` raises the error `Not_a_Matrix` with the message `"You have not given a valid matrix"`.

(d) `matrix_add [[1;5];[7;8];[9;10]] [[1;1];[2;2]]` raises the error `Non_matching_dimensions` with the message `"Dimensions of the two matrices do not match"`

7. `matrix_mult: int list list -> int list list -> int list list` Implement the matrix multiplication operation with matrices represented as above. Your dimension checking will change, but the errors raised will remain the same.

   (a) `matrix_mult [[1;5];[7;8];[9;10]] [[1;1];[2;2]]` returns `[[11;11];[23;23];[29;29]]`

   (b) `matrix_mult [[1;1];[2;2]] [[1;5];[7;8];[9;10]]` raises the error `Non_matching_dimensions` with the message `"Dimensions of the two matrices not suitable for multiplication"`