\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LAB 06 QUESTIONS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Name: (FILL THIS in)

- NetID: (THE kauf0095 IN kauf0095@umn.edu)

Answer the questions below according to the lab specification. Write

your answers directly in this text file and submit it to complete the

lab.

PROBLEM 1: Choose a Higher-Order Function

=========================================

SOLUTION :solution:

--------

,----

| (\* which\_higher\_func.ml: fill in short definitions of each function

| below using a higher-order List function like filter, iter, map, or

| fold\_left. \*)

|

| (\* val totlen : string list -> int

| Given a list of strings, compute the sum of their lengths

|

| # totlen ["abba"; "queen"; "def leopard"];;

| - : int = 20

| # totlen ["miles"; "cannonball"; "coltrane"; "ella"];;

| - : int = 27

| \*)

**| let totlen list =**

**| List.fold\_left (fun tot str -> tot + (String.length str)) 0 list**

**| ;;**

|

|

| (\* val run\_thunks : (unit -> unit) list -> unit

|

| Given a list of functions which take unit and return unit

| ("thunks") call each function with unit input.

|

| # run\_thunks [(fun ()->for i=1 to 4 do printf "Thunk!\n"; done);

| (fun ()->printf "What was that?\n")];;

| Thunk!

| Thunk!

| Thunk!

| Thunk!

| What was that?

| - : unit = ()

| # let xr = ref 5;;

| val xr : int ref = {contents = 5}

| # run\_thunks [(fun ()-> xr := 7);

| (fun ()-> xr := !xr \* 2);

| (fun ()-> printf "done\n")];;

| done

| - : unit = ()

| # xr;;

| - : int ref = {contents = 14}

| \*)

**| let run\_thunks list =**

**| List.iter (fun f-> f ()) list**

**| ;;**

|

| (\* val func\_results : (unit -> 'a) list -> 'a list

|

| Given a list of functions which take unit and return something,

| create list with the return value given by calling each function

| with unit parameter .

|

| # func\_results [(fun ()-> 7);

| (fun ()-> 16);

| (fun ()-> 8\*5+2);];;

| - : int list = [7; 16; 42]

|

| # func\_results [(fun ()-> "hello there");

| (fun ()-> sprintf "%s %s" "goodbye" "now");];;

| - : string list = ["hello there"; "goodbye now"]

| \*)

**| let func\_results list =**

**| List.map (fun f -> f ()) list**

**| ;;**

|

| (\* val keepers : ('a \* string \* 'b) list -> ('a \* string \* 'b) list

|

| Given a list of triples (3-tuples), create a list with elements

| that have string "keep" as the 2nd element of the triple

|

| # keepers [(1,"nope",4); (7,"keep",2); (3,"nada",12); (11,"keep",11)];;

| - : (int \* string \* int) list = [(7, "keep", 2); (11, "keep", 11)]

|

| # keepers [("a","keep","b"); ("keep","nope",""); ("b","keep","c"); ("d","keep","ee")];;

| - : (string \* string \* string) list =

| [("a", "keep", "b"); ("b", "keep", "c"); ("d", "keep", "ee")]

| \*)

| let keepers list =

| List.filter (fun (a,b,c)-> b = "keep") list

| ;;

|

| (\* val strlens : string list -> int list

|

| Given a list of strings, create a list of the lengths of those

| strings

|

| # strlens ["abba"; "queen"; "def leopard"];;

| - : int list = [4; 5; 11]

| # strlens ["miles"; "cannonball"; "coltrane"; "ella"];;

| - : int list = [5; 10; 8; 4]

| \*)

**| let strlens list =**

**| List.map String.length list**

**| ;;**

|

| (\* val div57 : int list -> int list

|

| Given a list of integers, create a list with eleemnts that are

| divisible by 5 or 7

|

| # div57 [5;7;11];;

| - : int list = [5; 7]

|

| # div57 [1;15;21;14;35;36];;

| - : int list = [15; 21; 14; 35]

| \*)

**| let div57 list =**

**| List.filter (fun n-> n mod 5 = 0 || n mod 7 = 0) list**

**| ;;**

|

| (\* val set\_sum : float ref -> float list -> unit

|

| Given a reference to a float and a float list, initialize the

| reference to 0.0 then set the sum of floats in the list

|

| # let mysum = ref 2.0;;

| val mysum : float ref = {contents = 2.}

|

| # set\_sum mysum [1.0; 6.0; 9.5];;

| - : unit = ()

| # mysum;;

| - : float ref = {contents = 16.5}

|

| # set\_sum mysum [];;

| - : unit = ()

| # mysum;;

| - : float ref = {contents = 0.}

|

| # set\_sum mysum [6.2; 3.1];;

| - : unit = ()

| # mysum;;

| - : float ref = {contents = 9.3}

| \*)

**| let set\_sum fref list =**

**| fref := 0.0;**

**| List.iter (fun f-> fref := !fref +. f) list;**

**| ;;**

|

| (\* val first\_elems : 'a list list -> 'a list

|

| Given a list of lists, create a list of the first elements. If any

| of the lists are empty, some kind of exception will result.

|

| # first\_elems [[1;2]; [3]; [4;5;6]];;

| - : int list = [1; 3; 4]

|

| # first\_elems [["a"]; ["b";"c";"d"]; ["f";"g"]; ["h";"i"]];;

| - : string list = ["a"; "b"; "f"; "h"]

|

| # first\_elems [];;

| first\_elems [];;

| - : 'a list = []

|

| # first\_elems [[1]; []; [3]];;

| Exception: Failure "hd".

| \*)

**| let first\_elems list =**

**| List.map List.hd list**

**| ;;**

|

| (\* val find\_min : 'a list -> 'a -> 'a

| Given a list of any kind of value and an absolute maximum value,

| return the minimum value in the list or the absolute max if the

| list is empty

|

| # max\_float;;

| - : float = 1.79769313486231571e+308

| # find\_min [7.5; 9.5; 6.3; 2.7; 8.1] max\_float;;

| - : float = 2.7

| # find\_min [7.5] max\_float;;

| - : float = 7.5

| # find\_min [] max\_float;;

| - : float = 1.79769313486231571e+308

|

| # max\_int;;

| - : int = 4611686018427387903

| # find\_min [7; 2; 4; 8; 11; 5] max\_int;;

| - : int = 2

| # find\_min [] max\_int;;

| - : int = 4611686018427387903

| \*)

**| let find\_min list absmax =**

**| List.fold\_left min absmax list**

**| ;;**

|

| (\* Alternate version \*)

| (\* let find\_min list absmax =

| \* List.fold\_left (fun mn x->if x<mn then x else mn) absmax list

| \* ;; \*)

|

| (\* val transforms : ('a -> 'b) list -> 'a -> 'b list

|

| Given a list of functions and a data element, apply each function

| to the data and create a list of the results.

|

|

| # let int\_funcs = [(fun x-> 2\*x);

| (fun x-> x+7);

| (fun x-> 0);];;

| val int\_funcs : (int -> int) list = [<fun>; <fun>; <fun>]

|

| # transforms int\_funcs 8;;

| - : int list = [16; 15; 0]

|

| # transforms int\_funcs 20;;

| - : int list = [40; 27; 0]

|

| # let string\_funcs = [(fun x-> x="indeed");

| (fun x-> (String.length x) > 4)];;

| val string\_funcs : (string -> bool) list = [<fun>; <fun>]

|

| # transforms string\_funcs "no";;

| - : bool list = [false; false]

|

| # transforms string\_funcs "indeed";;

| - : bool list = [true; true]

|

| # transforms string\_funcs "indubitably";;

| - : bool list = [false; true]

| \*)

**| let transforms transf\_list x =**

**| List.map (fun transf -> transf x) transf\_list**

**| ;;**

**`----**

PROBLEM 2: Higher-Order Function Alternatives

=============================================

SOLUTION :solution:

--------

`List.iteri' works identically to `iter' except that the function

passed in takes two parameters:

- the index in the list of an element (not present in `iter')

- the element itself

,----

| # let slist = ["a"; "b"; "c"; "d"];;

| val slist : string list = ["a"; "b"; "c"; "d"]

| # List.iteri (fun i s-> printf "Elem %d is '%s'\n" i s) slist;;

| Elem 0 is 'a'

| Elem 1 is 'b'

| Elem 2 is 'c'

| Elem 3 is 'd'

`----

`List.mapi' works identically to `map' but also passes two parameters

to its transform function, the index and the element.

,----

| (\* Make progressively longer strings with the given character \*)

| # List.mapi (fun i c-> String.make i c) ['a'; 'b'; 'c'; 'd'; 'e'; 'f'];;

| - : string list = [""; "b"; "cc"; "ddd"; "eeee"; "fffff"]

`----

(B) `partition'

~~~~~~~~~~~~~~~

SOLUTION :solution:

--------

Like `filter', `partition' takes a predicate function which returns

true/false for list elements. It forms two list results, one for true

elements, the other for false, and returns these two lists as a pair.

,----

| # List.partition (fun n->n mod 2 = 0) [1;2;3;4;5;6];;

| - : int list \* int list = ([2; 4; 6], [1; 3; 5])

| # List.partition (fun n-> n > 3) [1;2;3;4;5;6];;

| - : int list \* int list = ([4; 5; 6], [1; 2; 3])

| # List.partition (fun s-> s.[0]='c') ["cool"; "sweet"; "crap"; "clam"; "awesome"];;

| - : string list \* string list =

| (["cool"; "crap"; "clam"], ["sweet"; "awesome"])

`----

(C) `iter2' and `map2'

~~~~~~~~~~~~~~~~~~~~~~

SOLUTION :solution:

--------

Both these functions operate on two lists. They take two list

arguments and call their parameter function on corresponding elements

from each list.

,----

| # List.iter2 (fun c n-> printf "char '%c' and int %d\n" c n) ['a';'b';'c'] [1;2;3];;

| char 'a' and int 1

| char 'b' and int 2

| char 'c' and int 3

| - : unit = ()

|

| # List.map2 (fun a b-> (a,b)) ['a';'b';'c'] [1;2;3];;

| - : (char \* int) list = [('a', 1); ('b', 2); ('c', 3)]

|

| # List.map2 (fun c n-> String.make n c) ['a';'b';'c'] [1;2;3];;

| - : string list = ["a"; "bb"; "ccc"]

`----

They raise exceptions if the lists differ in length.

,----

| # List.iter2 (fun c n-> printf "char '%c' and int %d\n" c n) [] [1;2;3];;

| Exception: Invalid\_argument "List.iter2".

|

| # List.map2 (fun c n-> (c,n)) ['a';'b';'c'] [1;2];;

| Exception: Invalid\_argument "List.map2".

`----

PROBLEM 3: Folding and Reducing

===============================

(A)

~~~

In a REPL, show several uses of both `fold\_left' and `reduce\_left'

which produce equivalent results.

SOLUTION :solution:

--------

,----

| # reduce\_left (+) [1;2;3;4];;

| - : int = 10

| # fold\_left (+) 0 [1;2;3;4];;

| - : int = 10

| # reduce\_left (^) ["a";"bbb";"cc"];;

| - : string = "abbbcc"

| # fold\_left (^) "" ["a";"bbb";"cc"];;

| - : string = "abbbcc"

`----

(B)

~~~

Show some cases in which `reduce\_left' raises some exceptions while

`fold\_left' does not. Outline the conditions that cause `reduce\_left'

to raise exceptions and describe why `fold\_left' will never raise any

exceptions.

SOLUTION :solution:

--------

,----

| # fold\_left (^) "" [];;

| - : string = ""

| # reduce\_left (^) [];;

| Exception: Failure "Empty list".

| # fold\_left (+) 0 [];;

| - : int = 0

| # reduce\_left (+) [];;

| Exception: Failure "Empty list".

`----

`reduce\_left' will always fail on an empty list. This doesn't happen

with `fold\_left' as it simply returns the "initial value" that is

passed as a parameter.