COL 100: Lab Assignment № 5

March 21, 2018

Model solution

```
module MyList = struct

let rec nth li n = match li with
    [] -> raise (Failure "List.nth")
    | h::t -> if n=0 then h
        else if n<0 then raise (Invalid_argument "List.nth")
        else nth t (n-1)

let rec flatten l = match l with
    [] -> []
    | h::t -> h@(flatten t)

let rec map f l = match l with
    [] -> []
    | h::t -> (f h)::(map f t)

let rec rev_map f l = match l with
    [] -> []
    | h::t -> (rev_map f t)@[(f h)]

let rec fold_left f e l = match l with
    [] -> e
    | h::t -> fold_left f (f e h) t

let rec fold_right f l e = match l with
    [] -> e
    | h::t -> f h (fold_right f t e)

let rec map2 f l1 l2 = match (l1,l2) with
    ([],[]) -> []
    | ([],_) -> raise (Invalid_argument "MyList.map2")
    | (_,[]) -> raise (Invalid_argument "MyList.map2")
    | (h1::t1,h2::t2) -> (f h1 h2)::(map2 f t1 t2)
```

```

let rec fold_left2 f e l1 l2 = match (l1,l2) with
    ([],[]) -> e
    | ([],_) -> raise (Invalid_argument "MyList.fold_left2")
    | (_,[]) -> raise (Invalid_argument "MyList.fold_left2")
    | (h1::t1,h2::t2) -> fold_left2 f (f e h1 h2) t1 t2

let rec for_all f l = match l with
    [] -> true
    | h::t -> (f h) && (for_all f t)

let rec exists f l = match l with
    [] -> false
    | h::t -> (f h) || (exists f t)

let rec filter f l = match l with
    [] -> []
    | h::t -> if (f h) then h::(filter f t) else (filter f t)
end;;
```

Test cases

```
(*exists*)
let check1 x = (x>=18) in MyList.exists check1 [1;2;3;4;5;15;20] ;;
let check1 x = (x>=1000) in MyList.exists check1 [1;2;3;4;5;15;20] ;;
let check1 x = (x mod 2 = 0) in MyList.exists check1 [1;3;7;9;1;3;7;9] ;;
let check1 x = (x mod 5 = 0) in MyList.exists check1 ↩
    [10;20;30;40;100;200;300;400] ;;
let check1 x = (x mod 5 = 0) in MyList.exists check1 [] ;;
let check1 x = (x && false) in MyList.exists check1 [true;false;true;false↩
    ] ;;
let check1 x = ((x="\n") || (x="\t")) in MyList.exists check1 ["a";"b";"\n↩
    ";"\t"] ;;
let check1 x = (x || true) in MyList.exists check1 [false;false;false] ;;
let check1 x = (x=2) in MyList.exists check1 [2] ;;
(*filter*)
let check1 x = (x mod 2 = 0) in MyList.filter check1 [1;2;3;4;5;15;20] ;;
let check1 x = (x>=1000) in MyList.filter check1 [1;2;3;4;5;15;20] ;;
let check1 x = (x mod 2 = 0) in MyList.filter check1 [1;3;7;9;1;3;7;9] ;;
let check1 x = (x mod 5 = 0) in MyList.filter check1 ↩
    [10;20;30;40;100;200;300;400] ;;
let check1 x = (x mod 5 = 0) in MyList.filter check1 [] ;;
let check1 x = (x && false) in MyList.filter check1 [true;false;true;false↩
    ] ;;
```

```
let check1 x = ((x="\n") || (x="\t")) in MyList.filter check1 ["a";"b";"\n";"\t"] ;;
let check1 x = (x || true) in MyList.filter check1 [false;false;false] ;;
let check1 x = (x=2) in MyList.filter check1 [2] ;;
(*flatten*)
MyList.flatten [ [1;2] ;[3;4]; [5;6]] ;;
MyList.flatten [ [1;2] ;[3;4];[]; [5;6];[];[];[7]] ;;
MyList.flatten [ [];[];[];[];[];[] ] ;;
MyList.flatten [["A.";"G.";"P.="];["As"];["Good";"As"];["Possible"]] ;;
MyList.flatten [ [1;2;3;1;2;3] ;  [4;5;6]; [1;2;3] ;  [4;5;6;4;5;6]];;
MyList.flatten [ [];[1];[1;2]] ;;
MyList.flatten [ [1;2] ] ;;
MyList.flatten [ [1];[2] ] ;;
MyList.flatten [] ;;
(*fold_left*)
let  op1 x y = (x - y ) in  MyList.fold_left op1 109 [1;2;3;4;5;15;20] ;;
let  op1 x y = (x + y ) in  MyList.fold_left op1 0  [1;2;3;4;5;15;20] ;;
let  op1 x y = (x *  y ) in  MyList.fold_left op1 1  [1;2;3;4;5] ;;
let  op1 x y = (x *  y ) in  MyList.fold_left op1 0  [] ;;
let  op1 x y = (x + y* y ) in  MyList.fold_left op1 0  [1;2;3;4;5] ;;
let  op1 x y = (x && y ) in  MyList.fold_left op1 true  [true;false;true] ;;
let  op1 x y = (x || y ) in  MyList.fold_left op1 false  [false;false;false;false;true] ;;
let  op1 x y = (x^y ) in  MyList.fold_left op1 "a"  ["1";"";"\n";"2"] ;;
let  op1 x y = y in  MyList.fold_left op1 (-1)  [1;2;3;4;5] ;;
(*fold_left2*)
let  op1 t x y = t+ (x - y ) in  MyList.fold_left2 op1 1000 [1;2;3;4;5;15;20] [1;2;3;4;5;15;20] ;;
let  op1 t x y = t + (x *  y ) in  MyList.fold_left2 op1 1000  [1] [5;4;3] ;;
let  op1 t x y = (x * x  + t ) in  MyList.fold_left2 op1 0  [1;2;3;4;5] [11;12;13;14;15] ;;
let  op1 t x y =  t+1 in  MyList.fold_left2 op1 1000  [1;2;3;4;5] [11;12;13;14;15] ;;
let  op1 t x y = t + (x + y ) * (x + y) in  MyList.fold_left2 op1 1000  [1;2;3;4;5] [1;2;3;4;5] ;;
let  op1 t x y = t + (x + y ) * (x + y) in  MyList.fold_left2 op1 1000  [1] [1;2;3;4;5] ;;
let  op1 t x y = t^x^y in  MyList.fold_left2 op1 ""  ["a";"b"] ["\n";"\n"] ;;
let  op1 t x y = t && (x || y) in  MyList.fold_left2 op1 true  [false;false;true] [true;false;false] ;;
let  op1 t x y = t in  MyList.fold_left2 op1 1000  [] [] ;;
(*fold_right*)
let  op1 x y = (x - y ) in  MyList.fold_right op1 [1;2;3;4;5;15;20] 100 ;;
let  op1 x y = (x + y ) in  MyList.fold_right op1 [1;2;3;4;5;15;20] 0 ;;
let  op1 x y = (x *  y ) in  MyList.fold_right op1 [1;2;3;4;5] 1 ;;
let  op1 x y = (x *  y ) in  MyList.fold_right op1 [] 10000 ;;
let  op1 x y = (x*x + y ) in  MyList.fold_right op1 [1;2;3;4;5] 1000 ;;
let  op1 x y = (x && y ) in  MyList.fold_right op1 [true;false;true] true ;;
let  op1 x y = (x || y ) in  MyList.fold_right op1 [false;false;false;false;true] false ;;
let  op1 x y = (x^y ) in  MyList.fold_right op1 ["1";"";"\n";"2"] "a" ;;
let  op1 x y = x in  MyList.fold_right op1 [1;2;3;4;5] (-1) ;;
(*for_all*)
let check1 x = (x>=10) in MyList.for_all check1 [1;2;3;4;5;15;20] ;;
let check1 x = (x>=0) in MyList.for_all check1 [1;2;3;4;5;15;20] ;;
let check1 x = (x mod 2 = 0) in MyList.for_all check1 [1;2;3;4;5;15;20] ;;
let check1 x = (x mod 5 = 0) in MyList.for_all check1 [10;20;30;40;100;200;300;400] ;;
let check1 x = (x mod 5 = 0) in MyList.for_all check1 [] ;;
let check1 x = (x && false) in MyList.for_all check1 [true;false;true;false] ;;
let check1 x = ((x="\n") || (x="\t")) in MyList.for_all check1 ["a";"b";"\n";"\t"] ;;
let check1 x = (x || true) in MyList.for_all check1 [false;false;false] ;;
let check1 x = (x=2) in MyList.for_all check1 [2] ;;
(*map*)
let map1 x = x * x in MyList.map map1 [1;5;10;100;4] ;;
let map1 x = ("Hello",x) in MyList.map map1 [1;2;3;4;5;6] ;;
let map1 x = (x+1)::(x::[]) in MyList.map map1 [20;10;0] ;;
let map1 x = [] in MyList.map map1 [1;5;10;100;4] ;;
let map1 x = string_of_int x in MyList.map map1 [1;2;3;4;5;6] ;;
let map1 x = x-x*x in MyList.map map1 [] ;;
let map1 x = x*5 in MyList.map map1 [1;2] ;;
let map1 x = x>0 in MyList.map map1 [1;2;3;-1;-2;0] ;;
let map1 x = x mod 2 in MyList.map map1 [1;2;3;4;5;6] ;;
(*map2*)
let map1 x y = x in MyList.map2 map1 [] [] ;;
let map1 x y = x+y in MyList.map2 map1 [1;2;3;4;5;6] [1;2;3;4;5;6] ;;
let map1 x y = (x,y) in MyList.map2 map1 [1;2;3;4;5;6] [1;2;3;4;5;6] ;;
let map1 x y= (x)::(y::[]) in MyList.map2 map1 [1;2;3;4;5;6] [1;2;3;4;5;6] ;;
let map1 x y = "Lab 5"  in MyList.map2 map1 [1;2;3;4;5;6] [1;2;3;4;5;6] ;;
let map1 x y = (x,y,x)  in MyList.map2 map1 [1] [1;2] ;;
let map1 x y = x^y  in MyList.map2 map1 ["a";"b"] ["\n";"\t"] ;;
let map1 x y = [x;y]  in MyList.map2 map1 [1;1;1;1] [2;2;2;2] ;;
let map1 x y = y  in MyList.map2 map1 [] [1] ;;
(*nth*)
MyList.nth [10;20;30;40;50;60;70;80] 0 ;;
MyList.nth [10;20;30;40;50;60;70;80] 7 ;;
MyList.nth [10;20;30;40;50;60;70;80] 9 ;;
MyList.nth [10;20;30;40;50;60;70;80] (-1);;
MyList.nth [[10];[20;30;40];[50;60];[70;80]] 2 ;;
MyList.nth [] 4 ;;
MyList.nth [-1;-2;-3;-4;-5] 4 ;;
MyList.nth [[1];[2];[3];[4];[5]] 2 ;;
MyList.nth [] (-1);;
MyList.nth [10] 0 ;;
(*rev_map*)
let rev_map1 x = x * x in MyList.rev_map rev_map1 [1;5;10;100;4] ;;
let rev_map1 x = ("Hello") in MyList.rev_map rev_map1 [1;2;3;4;5;6] ;;
let rev_map1 x = (x+1)::(x::[]) in MyList.rev_map rev_map1 [20;10;0] ;;
let rev_map1 x = 2 * x in MyList.rev_map rev_map1 [1;5;10;100;4] ;;
let rev_map1 x = ("Hello",x) in MyList.rev_map rev_map1 [1;2;3;4;5;6] ;;
let map1 x = x-x*x in MyList.rev_map map1 [] ;;
let map1 x = x*5 in MyList.rev_map map1 [1;2] ;;
let map1 x = x>0 in MyList.rev_map map1 [1;2;3;-1;-2;0] ;;
let map1 x = x mod 2 in MyList.rev_map map1 [1;2;3;4;5;6] ;;
```