

# COL 100: Lab Assignment № 5

March 21, 2018

## Model solution

---

```
1 module MyList = struct
2
3 let rec nth li n = match li with
4   [] -> raise (Failure "List.nth")
5   | h::t -> if n=0 then h
6             else if n<0 then raise (Invalid_argument "List.nth")
7             else nth t (n-1)
8
9 let rec flatten l = match l with
10  [] -> []
11  | h::t -> h@(flatten t)
12
13 let rec map f l = match l with
14  [] -> []
15  | h::t -> (f h)::(map f t)
16
17 let rec rev_map f l = match l with
18  [] -> []
19  | h::t -> (rev_map f t)@[(f h)]
20
21 let rec fold_left f e l = match l with
22  [] -> e
23  | h::t -> fold_left f (f e h) t
24
25 let rec fold_right f l e = match l with
26  [] -> e
27  | h::t -> f h (fold_right f t e)
28
29 let rec map2 f l1 l2 = match (l1,l2) with
30  ([],[]) -> []
31  | ([],_) -> raise (Invalid_argument "MyList.map2")
32  | (_,[]) -> raise (Invalid_argument "MyList.map2")
33  | (h1::t1,h2::t2) -> (f h1 h2)::(map2 f t1 t2)
```

```

34
35 let rec fold_left2 f e l1 l2 = match (l1,l2) with
36   ([],[]) -> e
37   | ([],_) -> raise (Invalid_argument "MyList.fold_left2")
38   | (_,[]) -> raise (Invalid_argument "MyList.fold_left2")
39   | (h1::t1,h2::t2) -> fold_left2 f (f e h1 h2) t1 t2
40
41 let rec for_all f l = match l with
42   [] -> true
43   | h::t -> (f h) && (for_all f t)
44
45 let rec exists f l = match l with
46   [] -> false
47   | h::t -> (f h) || (exists f t)
48
49 let rec filter f l = match l with
50   [] -> []
51   | h::t -> if (f h) then h::(filter f t) else (filter f t)
52 end;;

```

---

## Test cases

---

```

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

```

```

18 let check1 x = ((x=="\n") || (x=="\t")) in MyList.filter check1 ["a";"b";"\n↵
    ";"\t"] ;;
19 let check1 x = (x || true) in MyList.filter check1 [false;false;false] ;;
20 let check1 x = (x=2) in MyList.filter check1 [2] ;;
21 (*flatten*)
22 MyList.flatten [ [1;2] ;[3;4]; [5;6]] ;;
23 MyList.flatten [ [1;2] ;[3;4];[]; [5;6];[];[];[7]] ;;
24 MyList.flatten [ [];[];[];[];[];[] ] ;;
25 MyList.flatten [ ["A."; "G."; "P.="]; ["As"]; ["Good"; "As"]; ["Possible"] ] ;;
26 MyList.flatten [ [1;2;3;1;2;3] ; [4;5;6]; [1;2;3] ; [4;5;6;4;5;6]];;
27 MyList.flatten [ [];[1];[1;2]] ;;
28 MyList.flatten [ [1;2] ] ;;
29 MyList.flatten [ [1];[2] ] ;;
30 MyList.flatten [] ;;
31 (*fold_left*)
32 let op1 x y = (x - y ) in MyList.fold_left op1 109 [1;2;3;4;5;15;20] ;;
33 let op1 x y = (x + y ) in MyList.fold_left op1 0 [1;2;3;4;5;15;20] ;;
34 let op1 x y = (x * y ) in MyList.fold_left op1 1 [1;2;3;4;5] ;;
35 let op1 x y = (x * y ) in MyList.fold_left op1 0 [] ;;
36 let op1 x y = (x + y * y ) in MyList.fold_left op1 0 [1;2;3;4;5] ;;
37 let op1 x y = (x && y ) in MyList.fold_left op1 true [true;false;true] ↵
    ;;
38 let op1 x y = (x || y ) in MyList.fold_left op1 false [false;false;↵
    false;false;true] ;;
39 let op1 x y = (x^y ) in MyList.fold_left op1 "a" ["1";"";"\n";"2"] ;;
40 let op1 x y = y in MyList.fold_left op1 (-1) [1;2;3;4;5] ;;
41 (*fold_left2*)
42 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] ;;
43 let op1 t x y = t + (x * y ) in MyList.fold_left2 op1 1000 [1] [5;4;3]↵
    ;;
44 let op1 t x y = (x * x + t ) in MyList.fold_left2 op1 0 [1;2;3;4;5] ↵
    [11;12;13;14;15] ;;
45 let op1 t x y = t+1 in MyList.fold_left2 op1 1000 [1;2;3;4;5] ↵
    [11;12;13;14;15] ;;
46 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] ;;
47 let op1 t x y = t + (x + y ) * (x + y) in MyList.fold_left2 op1 1000 ↵
    [1] [1;2;3;4;5] ;;
48 let op1 t x y = t^x^y in MyList.fold_left2 op1 "" ["a";"b"] ["\n";"\n"]↵
    ;;
49 let op1 t x y = t && (x || y) in MyList.fold_left2 op1 true [false;↵
    false;true] [true;false;false] ;;
50 let op1 t x y = t in MyList.fold_left2 op1 1000 [] [] ;;
51 (*fold_right*)
52 let op1 x y = (x - y ) in MyList.fold_right op1 [1;2;3;4;5;15;20] 100 ;;
53 let op1 x y = (x + y ) in MyList.fold_right op1 [1;2;3;4;5;15;20] 0 ;;

```

```

54 let op1 x y = (x * y) in MyList.fold_right op1 [1;2;3;4;5] 1 ;;
55 let op1 x y = (x * y) in MyList.fold_right op1 [] 10000 ;;
56 let op1 x y = (x*x + y) in MyList.fold_right op1 [1;2;3;4;5] 1000 ;;
57 let op1 x y = (x && y) in MyList.fold_right op1 [true;false;true] true <-
    ;;
58 let op1 x y = (x || y) in MyList.fold_right op1 [false;false;false;<-
    false;true] false ;;
59 let op1 x y = (x^y) in MyList.fold_right op1 ["1";"";"\n";"2"] "a" ;;
60 let op1 x y = x in MyList.fold_right op1 [1;2;3;4;5] (-1) ;;
61 (*for_all*)
62 let check1 x = (x>=10) in MyList.for_all check1 [1;2;3;4;5;15;20] ;;
63 let check1 x = (x>=0) in MyList.for_all check1 [1;2;3;4;5;15;20] ;;
64 let check1 x = (x mod 2 = 0) in MyList.for_all check1 [1;2;3;4;5;15;20] ;;
65 let check1 x = (x mod 5 = 0) in MyList.for_all check1 <-
    [10;20;30;40;100;200;300;400] ;;
66 let check1 x = (x mod 5 = 0) in MyList.for_all check1 [] ;;
67 let check1 x = (x && false) in MyList.for_all check1 [true;false;true;<-
    false] ;;
68 let check1 x = ((x="\n") || (x="\t")) in MyList.for_all check1 ["a";"b";"\<-
    n";"\t"] ;;
69 let check1 x = (x || true) in MyList.for_all check1 [false;false;false] ;;
70 let check1 x = (x=2) in MyList.for_all check1 [2] ;;
71 (*map*)
72 let map1 x = x * x in MyList.map map1 [1;5;10;100;4] ;;
73 let map1 x = ("Hello",x) in MyList.map map1 [1;2;3;4;5;6] ;;
74 let map1 x = (x+1)::(x::[]) in MyList.map map1 [20;10;0] ;;
75 let map1 x = [] in MyList.map map1 [1;5;10;100;4] ;;
76 let map1 x = string_of_int x in MyList.map map1 [1;2;3;4;5;6] ;;
77 let map1 x = x-x*x in MyList.map map1 [] ;;
78 let map1 x = x*5 in MyList.map map1 [1;2] ;;
79 let map1 x = x>0 in MyList.map map1 [1;2;3;-1;-2;0] ;;
80 let map1 x = x mod 2 in MyList.map map1 [1;2;3;4;5;6] ;;
81 (*map2*)
82 let map1 x y = x in MyList.map2 map1 [] [] ;;
83 let map1 x y = x+y in MyList.map2 map1 [1;2;3;4;5;6] [1;2;3;4;5;6] ;;
84 let map1 x y = (x,y) in MyList.map2 map1 [1;2;3;4;5;6] [1;2;3;4;5;6] ;;
85 let map1 x y = (x)::(y::[]) in MyList.map2 map1 [1;2;3;4;5;6] [1;2;3;4;5;6]<-
    ;;
86 let map1 x y = "Lab 5" in MyList.map2 map1 [1;2;3;4;5;6] [1;2;3;4;5;6] ;;
87 let map1 x y = (x,y,x) in MyList.map2 map1 [1] [1;2] ;;
88 let map1 x y = x^y in MyList.map2 map1 ["a";"b"] ["\n";"\t"] ;;
89 let map1 x y = [x;y] in MyList.map2 map1 [1;1;1;1] [2;2;2;2] ;;
90 let map1 x y = y in MyList.map2 map1 [] [1] ;;
91 (*nth*)
92 MyList.nth [10;20;30;40;50;60;70;80] 0 ;;
93 MyList.nth [10;20;30;40;50;60;70;80] 7 ;;
94 MyList.nth [10;20;30;40;50;60;70;80] 9 ;;

```

```
95 MyList.nth [10;20;30;40;50;60;70;80] (-1);;
96 MyList.nth [[10];[20;30;40];[50;60];[70;80]] 2 ;;
97 MyList.nth [] 4 ;;
98 MyList.nth [-1;-2;-3;-4;-5] 4 ;;
99 MyList.nth [[1];[2];[3];[4];[5]] 2 ;;
100 MyList.nth [] (-1);;
101 MyList.nth [10] 0 ;;
102 (*rev_map*)
103 let rev_map1 x = x * x in MyList.rev_map rev_map1 [1;5;10;100;4] ;;
104 let rev_map1 x = ("Hello") in MyList.rev_map rev_map1 [1;2;3;4;5;6] ;;
105 let rev_map1 x = (x+1)::(x::[]) in MyList.rev_map rev_map1 [20;10;0] ;;
106 let rev_map1 x = 2 * x in MyList.rev_map rev_map1 [1;5;10;100;4] ;;
107 let rev_map1 x = ("Hello",x) in MyList.rev_map rev_map1 [1;2;3;4;5;6] ;;
108 let map1 x = x-x*x in MyList.rev_map map1 [] ;;
109 let map1 x = x*5 in MyList.rev_map map1 [1;2] ;;
110 let map1 x = x>0 in MyList.rev_map map1 [1;2;3;-1;-2;0] ;;
111 let map1 x = x mod 2 in MyList.rev_map map1 [1;2;3;4;5;6] ;;
```

---