Lab – 5

Week of Feb. 11, 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.

The following submission file is *required*:

1. `MyList.ml`

2 Learn on your own

1. You will implement a module named `MyList` which will perform several useful operations on lists. The functions will mirror functions of the same name in the List module of OCaml. But first, **on your own:**

   (a) Figure out how to write a module. Following is a sample way to write a module (some important key words are highlighted):

```
module MyList = struct
   let rec length l =
      match l with
      | [] -> 0
      | h::t -> 1 + length t
end
```

   It contains a single function `length` only. You can use this function in OCaml Toplevel as follows:

```
# MyList.length [2; 3; 4];;
- :  int = 3
```

   (b) Learn how to define exceptions and raise exceptions here (note that an exception has type `exn`, built-in type in OCaml). Here is an example of defining and raising an exception:

```
# exception Invalid_age of int;;
exception Invalid_age of int
# let is_teenager age =
   if age < 0
   then raise (Invalid_age age)
   else 12 < age && age < 20
val is_teenager :  int -> bool = <fun>
# is_teenager (-2);;
Exception:  Invalid_age (-2).
# is_teenager 14;;
- :  bool = true
```

3 Functions that you should implement

(*submission file:* `MyList.ml`)

Your module, named `MyList`, should contain the following functions. For each `List` function below, go through the description of that function here: `https://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html`. An example of how to use each function is given. Try a few more on your own to ensure that you know what output is expected (including errors and exceptions that are raised). Then, implement each function, *exactly as specified* in the `List` module (note that this *includes* raising specific exceptions) using the basic techniques that we have already learnt in class.

1. `nth: 'a list -> int -> 'a`

```
# let a = [1; 2; 3];;
val a : int list = [1; 2; 3]
# MyList.nth a 2;;
- : int = 3
```

2. `flatten: 'a list list -> 'a list`

```
# let b = [4; 5];;
val b : int list = [4; 5]
# MyList.flatten [a; b; a];;
- : int list = [1; 2; 3; 4; 5; 1; 2; 3]
```

3. `map: ('a -> 'b) -> 'a list -> 'b list`

```
# let succ x = x + 1;;
val succ : int -> int = <fun>
# MyList.map succ a;;
- : int list = [2; 3; 4]
```

4. `rev_map: ('a -> 'b) -> 'a list -> 'b list`

```
# MyList.rev_map succ a;;
- : int list = [4; 3; 2]
```

5. `fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`

```
(* following lines calculate the sum of elements of an int list *)
# let sum x y = x + y;;
val sum : int -> int -> int = <fun>
# MyList.fold_left sum 0 a;;
- : int = 6

# let mul_str x s = x * int_of_string(s);;
val mul_str : int -> string -> int = <fun>
# MyList.fold_left mul_str 0 ["5"; "2"; "4"];;
- : int = 40
```

6. `fold_right: ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`

```
# let cat_str x s = string_of_int(x) ^ s;;
val cat_str : int -> string -> string = <fun>
# MyList.fold_right cat_str [1; 2; 3] "\n";;
- : string = "123\n"
```

7. `map2: ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list`

```
# let zip x y = (x,y);;
val zip : 'a -> 'b -> 'a * 'b = <fun>
# MyList.map2 zip [1; 2; 3] ["one"; "two"; "three"];;
- : (int * string) list = [(1, "one"); (2, "two"); (3, "three")]
```

8. `fold_left2: ('a -> 'b -> 'c -> 'a) -> 'a -> 'b list -> 'c list -> 'a`

```
(* following lines calculate the dot product of 2 lists *)
# let mul_add a x y = a + x * y;;
val mul_add : int -> int -> int -> int = <fun>
# MyList.fold_left2 mul_add 0 [1; 2; 3] [4; 5; 6];;
- : int = 32
```

9. `for_all: ('a -> bool) -> 'a list -> bool`

```
# let even x = (x mod 2 = 0);;
val even : int -> bool = <fun>
# MyList.for_all even [2; 4; 6];;
- : bool = true
# MyList.for_all even [2; 3; 6];;
- : bool = false
```

10. `exists: ('a -> bool) -> 'a list -> bool`

```
# MyList.exists even [2; 3; 6];;
- : bool = true
# MyList.exists even [1; 3; 5];;
- : bool = false
```

11. `filter: ('a -> bool) -> 'a list -> 'a list`

```
# MyList.filter even [1; 2; 3; 6];;
- : int list = [2; 6]
```