Exercises Week 8

Study Sections 1.3, 1.4, 1.6 — 1.8 of OCaml reference manual.

1. [1+1 POINTS] Use `List.fold_left` or `List.fold_right` to implement (a) `append` and (b) `rev`:

```
# append [1; 2; 3] [4; 5; 6];;
- : int list = [1; 2; 3; 4; 5; 6]
# rev [1; 2; 3];;
- : int list = [3; 2; 1]
```

2. [2 POINTS] Use `List.map` (twice) and `List.concat` to implement the following `product`:

```
# product (fun x y -> (x, y)) [1; 2] [10; 20; 30];;
- : (int * int) list =
     [(1, 10); (1, 20); (1, 30); (2, 10); (2, 20); (2, 30)]
# product ( + ) [1; 2] [10; 20; 30];;
- : int list = [11; 21; 31; 12; 22; 32]
```

3. [1+1+1 POINTS] The merge sort `msort` is based on a divide (`even_odd`) and conquer (`merge`). Implement (a) `even_odd`, (b) `merge`, and (c) `msort`.

```
# even_odd [1;2;3;3;4;5];;
- : int list * int list = ([1; 3; 4], [2; 3; 5])
# merge [1;4;5] [2;3;3];;
- : int list = [1; 2; 3; 3; 4; 5]
# msort [3;2;4;1;3];;
- : int list = [1; 2; 3; 3; 4]
```

4. [1+2 POINTS] Consider the following code:

```
type expr =
  | Const of int
  | Add of expr * expr
  | Mul of expr * expr
type instr = Push | Addint | Mulint | Value of int
exception Bad_code
```

   (a) Implement a bytecode compiler `compile`, which translates `expr` to `instr list`.

   (b) Implement `interpret`, a bytecode interpreter for `instr list`.

```
# let code = compile (Mul (Const 10, Add (Const 20, Const 30)));;
val code : instr list =
  [Push; Value 10; Push; Value 20; Push; Value 30; Addint; Mulint]
# interpret code;;
- : int = 500
```

   Hint: `List.nth` $[x_0;x_1;\ldots;x_m]$ $n = x_n$

Submit `yourMatrNr.ml` before 23:59 on **December 7**.