Homework 2
ENE4014 Programming Languages, Spring 2026
due: 5/6(Wed), 23:59

- Submit one file per problem via the submission system in the course website. Make sure that your files are compiled and run without errors. Any file that does not compile or run will receive zero points. You should include all auxiliary or helper functions in the same file.
- Do not use any external libraries. Any code that uses external libraries (including the standard library (`https://ocaml.org/api/index.html`)) will receive zero points.

Exercise 1 (5 points) Write a function

```
revrev:  'a list list -> 'a list list
```

such that `revrev` t returns the result of reversing the order of the elements of t and then reversing the order of the elements of each element of t. For example,

```
revrev [[1;2;3]; [4;5;6]; [7;8;9]] = [[9;8;7]; [6;5;4]; [3;2;1]]
```

□

Exercise 2 (5 points) Write a function

```
union:  'a list -> 'a list -> 'a list
```

such that `union` t_1 t_2 returns the union of the elements of t_1 and t_2 without duplicates. You can use `=` to compare elements of the list. The order of elements in the return value does not matter. For example,

`union` $[1;2;3;4]$ $[3;4;5;6] = [1;2;3;4;5;6]$.

□

Exercise 3 (5 points) Write a function

```
alterSum:  int list -> int
```

such that `alterSum` t returns an integer that is the result of applying addition and subtraction to the elements of the list t alternately. The first operation, if applicable, is addition. For an empty list, `alterSum` returns 0. For example,

$$\texttt{alterSum}\ [] = 0$$

$$\texttt{alterSum}\ [1;2;3;4;5] = 1+2-3+4-5 = -1$$

$$\texttt{alterSum}\ [1;2;3;4;5;6] = 1+2-3+4-5+6 = 5$$

□

Exercise 4 (5 points) Write a function

```
dsort:  int list -> int list
```

such that `dsort` t returns a list that is the result of sorting the elements of t in descending order. For example,

$$\texttt{dsort}\ [3;2;1;4;5] = [5;4;3;2;1]$$

$$\texttt{dsort}\ [2;1;4;3;6;5] = [6;5;4;3;2;1].$$

Do not use any built-in sorting functions (e.g., List.sort, List.stable_sort, List.fast_sort, etc.). □

Exercise 5 (5 points) Write a higher-order function

$$\texttt{iter}\ n\ f = f^n$$

where f^n is the function that is the result of applying f to itself n times (in other words, $f^n(x) = \underbrace{f(\cdots(f}_{n}(x)))$. When $n = 0$, the function returns the identity function (`fun x -> x`).

For example,

$$\texttt{(iter}\ n\ f\texttt{)}\ \texttt{0}$$

when $f =$ `(fun x -> x + 2)` and $n = 3$ returns 6 because

$$\texttt{(iter}\ n\ f\texttt{)}\ \texttt{0} = f^3(0) = f(f(f(0))) = f(f(2)) = f(4) = 6.$$

□

Exercise 6 (5 points) Write a function

```
sigma :  int * int * (int -> int) -> int.
```

such that `sigma(a,b,f)` returns $\Sigma_{n=a}^{b} f(n)$.

Exercise 7 (5 points) Write a function

```
mapn:  ('a -> 'a) -> int -> 'a list -> 'a list
```

such that `mapn` f n l returns a list that is the result of applying f to each element of l n times. More precisely,

$$\texttt{mapn}\ f\ n\ [x_1; x_2; \cdots, x_m] = [f^n(x_1); f^n(x_2); \cdots, f^n(x_m)]$$

where f^n is defined as in the previous exercise and x_i is the i-th element of l. For example,

```
mapn (fun x -> x + 1) 3 [1; 2; 3; 4; 5] = [4; 5; 6; 7; 8]

mapn (fun x -> x + 1) 0 [1; 2; 3; 4; 5] = [1; 2; 3; 4; 5]
```

□

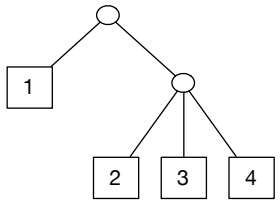
Exercise 8 (5 points) N-ary tree is a tree in which each node has at most N children. The following is the definition of an N-ary tree whose elements are of type `'a`.

```
type 'a ntree = Leaf of 'a | Node of ('a ntree list)
```

For example,

```
Node [Leaf 1; Node [Leaf 2; Leaf 3; Leaf 4]]
```

is a tree represented as follows:



Write a function

```
findn:  'a ntree -> int
```

such that `findn` t returns the maximum number of children of any node in the tree t. For example,

```
findn (Node [Leaf 1; Node [Leaf 2; Leaf 3; Leaf 4]]) = 3
```

□

Exercise 9 (5 points) Write a function

```
flatten:  'a ntree -> 'a list
```

such that `flatten` t returns a list containing all the `'a` type-elements in the tree t. The order of elements in the return value does not matter. For example,

```
flatten (Node [Leaf 1; Node [Leaf 2; Leaf 2; Leaf 4]]) = [1; 2; 2; 4]
```

□

Exercise 10 (5 points) Consider the following OCaml data type for propositional formulas

```
type formula = TRUE | FALSE
  | NOT of formula
  | ANDALSO of formula * formula
  | ORELSE of formula * formula
  | IMPLY of formula * formula
  | LESS of expr * expr
and expr = NUM of int
  | PLUS of expr * expr
  | MINUS of expr * expr
```

Considering the above definition, write a function

$$\texttt{eval} : \texttt{formula} \rightarrow \texttt{bool}$$

that computes the truth value of a given formula. For example,

```
eval (IMPLY (IMPLY (TRUE, FALSE), TRUE))
```

evaluates to *true*, and

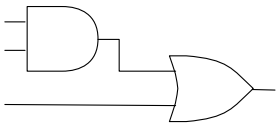
```
eval (LESS (NUM 5, PLUS (NUM 1, NUM 2)))
```

evaluates to *false*.

Exercise 11 (5 points) Consider the following OCaml data type for Boolean circuits

```
type circuit = IN
  | AND of circuit * circuit
  | OR of circuit * circuit
```

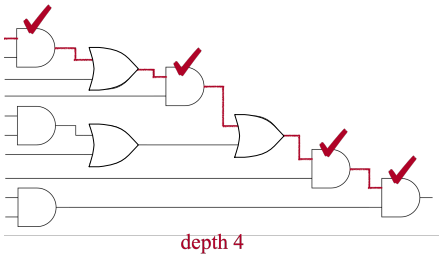
where `IN` denotes input. For example, the following circuit



can be described as

```
OR (AND (IN, IN), IN)
```

The AND depth of a circuit is the maximum number of sequential AND gates from input to output. For example, the following circuit has the AND depth of 4.

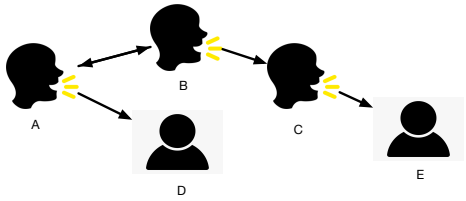


Write a function

```
and_depth :  circuit -> int
```

that takes a circuit and returns its AND depth.

Exercise 12 (5 points) Suppose we are interested in if someone gets COVID-19. The following image describes people who are talking to someone else in a party at a moment.



The type `talkingto` is for representing whom each person is talking to:

```
type talkingto = (string * string) list
```

For example, the above situation can be represented as

```
let party = [("A", "B"); ("B", "A"); ("A", "D"); ("B", "C"); ("C", "E")]
```

Suppose no one is wearing a mask, and if person A talks to B and A gets COVID-19, B also gets infected immediately. For example, in the above situation, E gets infected if A got COVID because A talks to B, who talks to C, who talks to E.

Write a function

```
infected :  talkingto -> string -> string -> bool
```

that determines if a person (3rd argument) gets infected when another person (2nd argument) got COVID. For example, the function should behave as follows:

```
infected party "A" "E" = true
infected party "B" "D" = true
infected party "C" "D" = false
infected party "C" "B" = false
```

Exercise 13 (5 points) As an extension of the previous exercise, suppose people who get vaccinated never get COVID. Write a function

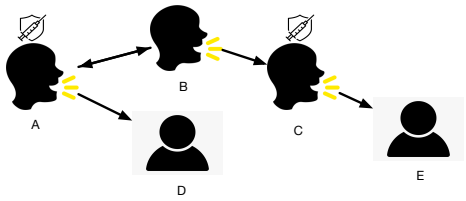
```
infected_vaccine :  talkingto -> string list -> string -> string -> bool
```

that additionally gets a list of people who get vaccinated as the second argument.

For example,

```
infected_vaccine party ["A"; "C"] "B" "D" = false
```

as if A and C get vaccinated, E is free from COVID even if B got infected because C is blocking the way from B to E.



The followings are other example behaviors.

```
infected_vaccine party ["A"; "C"] "B" "E" = false
infected_vaccine party ["A"] "B" "E" = true
infected_vaccine party ["C"] "A" "E" = false
```

Exercise 14 (5 points) Write a function

$$\texttt{diff} : \texttt{ae} * \texttt{string} \to \texttt{ae}$$

that differentiates the given algebraic expression with respect to the variable given as the second argument. The `ae` type is defined as follows:

```
type ae = CONST of int
  | VAR of string
  | POWER of string * int
  | TIMES of ae list
  | SUM of ae list
```

For example, $x^2 + 2x + 1$ is represented by

```
SUM [POWER ("x", 2); TIMES [CONST 2; VAR "x"]; CONST 1]
```

and differentiating it (w.r.t. "x") gives $2x + 2$, which can be represented by

```
SUM [TIMES [CONST 2; VAR "x"]; CONST 2]
```

Exercise 15 (5 points) Write a function

$$\texttt{calculate} : \texttt{exp} \to \texttt{float}$$

that returns a result of a given arithmetic formula. The `exp` type is defined as follows:

```
type exp = X | INT of int
  | REAL of float
  | ADD of exp * exp
  | SUB of exp * exp
  | MUL of exp * exp
  | DIV of exp * exp
  | SIGMA of exp * exp * exp
  | INTEGRAL of exp * exp * exp
```

For example, the following arithmetic formulas can be written in the exp type:

$\sum_{x=1}^{10}(x \times x - 1)$	`SIGMA(INT 1, INT 10, SUB(MUL(X, X), INT 1))`
$\int_{x=1.0}^{10.0}(x \times x - 1)\,dx$	`INTEGRAL(REAL 1.0, REAL 10.0, SUB(MUL(X, X), INT 1))`

When you compute integrals, dx should be 0.1.