université de BORDEAUX	**Année universitaire :** 2021/2022 **Parcours :** Licence Informatique 2e année UE 4TINA01U **Épreuve :** Examen de Programmation fonctionnelle **Date :** Mardi 4 janvier 2022 9 :00 – 10 :30 **Durée :** 1h30 **Documents interdits.**	**Collège Sciences et Technologies**

Exercice 1 *(2pts)*

Pour chacun des types suivants, donner une expression ayant ce type.

1. `('a -> bool) -> 'a -> int`
2. `'a -> 'a list -> bool`

Pour chacune des expressions suivantes, donner sa **valeur** et son **type** si elle est correcte, sinon expliquer pourquoi elle est incorrecte.

3. `let x = 10 in x + (let x = 5 in let y = 2 * x in x + y)`
4. `let f x y = 3 * x + y in f 3`

Exercice 2 *(6pts)* Soit la fonction `mystere` suivante :

```
let rec mystere l1 l2 =
  match l1 with
    [] -> l2
  | h1 :: t1 -> h1 :: mystere t1 l2
```

5. Donner son type.
6. Que retourne l'appel suivant (valeur et type) ?

   ```
   mystere [1; 2; 3] [4; 5];;
   ```

7. Que fait la fonction `mystere` ?
8. Quelle est sa complexité ?[1].
9. Donner une version récursive terminale de `mystere` qui soit de même complexité.

Exercice 3 *(4pts)*

10. Écrire la fonction `compose f g` qui retourne la fonction composée $f \circ g$ de deux fonctions d'un argument f et g.
11. Écrire la fonction `compose_gen functions` qui s'applique à une liste de fonctions $f_1, f_2, \ldots f_n$ et qui retourne $f_1 \circ f_2 \ldots \circ f_n$.[2]

```
utop[30]> compose;;
- : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
utop[31]> compose (fun x -> 2 * x) (fun x -> x + 10);;
- : int -> int = <fun>
utop[32]> compose (fun x -> 2 * x) (fun x -> x + 10) 3;;
- : int = 26
utop[33]> compose_gen;;
- : ('a -> 'a) list -> 'a -> 'a = <fun>
utop[34]> compose_gen [(fun x -> x - 2); (fun x -> x * x); (fun x -> x + 2)];;
- : int -> int = <fun>
utop[35]> compose_gen [(fun x -> x - 2); (fun x -> x * x); (fun x -> x + 2)] 3;;
- : int = 23
```

[1] On pourra exprimer la complexité avec la notation $\mathcal{O}$ ou simplement compter le nombre d'appels récursifs.
[2] On pourra utiliser la fonction `compose` et la fonction `List.fold_left`.

Exercice 4 *(4pts)* On représente un couple de coordonnées (x, y) par le type `point` qui servira à représenter un point du plan.

```
type point = P of float * float
```

12. Écrire la fonction constructeur `make_point x y` qui construit un objet de type `point` à partir des coordonnées `x` et `y`.
13. Écrire les fonctions accesseur `cx point` (resp. `cy point`) qui permettent de récupérer la coordonnée `x` (resp. la coordonnée `y`) de l'objet `point` de type `point`.

```
utop[40]> let point = make_point 3. 1.5;;
val point : point = P (3., 1.5)
utop[41]> cx point;;
- : float = 3.
utop[42]> cy point;;
- : float = 1.5
```

Une transformation du plan transforme un point de coordonnées (x, y) en un point de coordonnées (x', y') et sera représentée par le type `point -> point`

14. Écrire une fonction `next_point point transformation` qui prend en paramètre `point` (un point de coordonnées (x, y)) et retourne le point de coordonnées (x', y') obtenu après application de la transformation `transformation` au point `point`.

```
utop[43]> point;;
- : point = P (3., 1.5)
utop[44]> let transformation = fun point -> make_point (cx point *. 2.) (cy point -. 3.);;
val transformation : point -> point = <fun>
utop[45]> next_point point transformation;;
- : point = P (6., -1.5)
```

15. Écrire une fonction `translation dx dy` qui retourne la transformation correpondant à une translation de vecteur (dx, dy).
16. Écrire une fonction `next_point_gen point transformations` qui prend en paramètre `point` (un point de coordonnées (x, y)) et retourne le point de coordonnées (x', y') obtenu après application de la composée des transformations de la liste `transformations`.[3].

```
utop[46]> let translation2_3 = translation 2. 3.;;
val translation2_3 : point -> point = <fun>
utop[47]> let homothetie3 = fun point -> make_point (3. *. cx point) (3. *. cy point);;
val homothetie3 : point -> point = <fun>
utop[48]> next_point_gen (make_point 1. 3.) [homothetie3; translation2_3; homothetie3];;
- : point = P (15., 36.)
```

Exercice 5 *(4pts)*

17. Écrire une fonction `count_sequences l n` qui compte le nombre de séquences d'éléments consécutifs identiques de longueur au moins `n` dans la liste `l`. À noter, qu'une séquence `l = [e;e;,...,e]` (ne contenant que des `e`), ne compte que pour 1 quand on compte le nombre de séquences de longueur plus petite que `l`. Exemples :

```
utop[50]> count_sequences;;
- : 'a list -> int -> int = <fun>
utop[51]> count_sequences [1;1;2;3;3;3;3;5;2;2;2;1;1;1] 3;;
- : int = 3
utop[52]> count_sequences [1;1;2;3;3;3;3;5;2;2;2;1;1;1] 2;;
- : int = 4
utop[53]> count_sequences [1;1;2;3;3;3;3;5;2;2;2;1;1;1] 4;;
- : int = 1
utop[54]> count_sequences [1;1;1;1;1;1] 3;;
- : int = 1
```

[3] On pourra utiliser la fonction `next_point` vue précédemment et la fonction `compose_gen` vue à l'exercice précédent.