université de BORDEAUX	**Année universitaire :** 2020/2021 **Parcours :** Licence Informatique 2e année UE 4TINA01U **Épreuve :** Devoir surveillé de Programmation fonctionnelle **Date :** vendredi 23 octobre 2020 09 :00 – 10 :30 **Durée :** 1h30 **Documents interdits.**	**Collège Sciences et Technologies**

Exercice 1 *(6pts)* Pour chacune des expressions suivantes, donner sa **valeur et le type** si elle est correcte, sinon expliquer pourquoi elle est incorrecte.

1. `3. +. 4.`
2. `3.14 + 1`
3. `let x = 10 in x + (let x = 5 and y = 3 in x + y)`
4. `let x = 10 in x + (let x = 5 and y = 3 * x in x + y)`
5. `let x = 10 in x + (let x = 5 in let y = 2 * x in x + y)`
6. `fun x y -> (x + y, y)`
7. `let f x y = 3 * x + 2 * y`
8. `f 5`

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

9. `'a -> 'a`
10. `'a -> 'a -> ('a -> 'b) -> bool`
11. `('a -> 'b) -> ('c -> 'a) -> 'c -> 'b`
12. `'a -> ('a -> bool) -> int`

Exercice 2 *(4pts)* On considère la suite récurrente d'ordre 2 suivante :

$$\begin{cases} u_0 &= 0 \\ u_1 &= 3 \\ u_n &= -u_{n-1} + 2u_{n-2}, \forall n \geqslant 2 \end{cases}$$

13. Écrire une fonction récursive `seq_aux` de type `int -> int * int` qui à tout entier naturel n associe le couple (u_n, u_{n+1}).
14. En déduire une fonction `seq` qui à tout entier naturel n associe u_n.
15. Quel est le type de `seq` ?
16. Quelle est la complexité de `seq` en fonction de son paramètre n ?

Exemples :

```
# seq_aux;;
- : int -> int * int = <fun>
# seq_aux 0;;
- : int * int = (0, 3)
# seq_aux 1;;
- : int * int = (3, -3)
# seq_aux 2;;
- : int * int = (-3, 9)
# seq 3;;
- : int = 9
```

Exercice 3 *(5pts)* On souhaite manipuler des couples d'entiers. Pour cela on utilise le type :

```
let type couple = C of int * int
```

17. Écrire le constructeur `make_couple` qui permet de construire un couple à partir de deux entiers.

18. Écrire les accesseurs `couple_fst` (resp. `couple_snd`) permettant d'accéder au premier (resp. deuxième) élément d'un couple de type `couple`.
19. Écrire la fonction `couple_sum` qui retourne la somme des deux composantes d'un couple.
20. Écrire la fonction `couple_add` qui prend en paramètre deux couples et retourne un couple dont la première (resp. deuxième) composante est la somme des premières (resp. deuxièmes) composantes des deux couples.

```
# make_couple 1 2;;
- : couple = C (1, 2)
# couple_fst (make_couple 1 2);;
- : int = 1
# couple_snd (make_couple 1 2);;
- : int = 2
# couple_sum (make_couple 1 2);;
- : int = 3
# couple_add (make_couple 1 2) (make_couple 3 4);;
- : couple = C (4, 6)
```

On décide de changer le type `couple` pour

```
type couple = bool -> int
```

Un couple est représenté par une fonction qui s'applique à un booléen. Appliquée à `true` (resp. `false`), la fonction retourne la première (resp. deuxième) composante du couple.

21. Réécrire le constructeur `make_couple` et les deux accesseurs `couple_fst` et `couple_snd` avec ce nouveau type.
22. Les fonctions `couple_sum` et `couple_add` que vous avez écrites restent-elles valables avec le nouveau type ?

Exemples :

```
# make_couple 1 2;;
- : bool -> int = <fun>
# couple_fst (make_couple 1 2);;
- : int = 1
# couple_snd (make_couple 1 2);;
- : int = 2
# couple_sum (make_couple 1 2);;
- : int = 3
# couple_add (make_couple 1 2) (make_couple 3 4);;
- : bool -> int = <fun>
```

Exercice 4 *(5pts)* Soit f une fonction de $\mathbb{R}$ dans $\mathbb{R}$.

Si f est dérivable, la fonction dérivée de f, f' peut être définie par

$$\begin{array}{rccc} f' : & \mathbb{R} & \to & \mathbb{R} \\ & x & \mapsto & \lim_{h\to 0} \tau(f,h,x) \end{array}$$

où

$$\tau(f,h,x) = \frac{f(x+h) - f(x-h)}{2h}$$

En prenant un h petit (proche de 0), on obtient la fonction f'_h, dérivée approchée de f, définie par

$$\begin{array}{rccc} f'_h : & \mathbb{R} & \to & \mathbb{R} \\ & x & \mapsto & \tau(f,h,x) \end{array}$$

23. Écrire la fonction `derivee` qui prend en paramètre f, h et retourne f'_h.

Exemples :

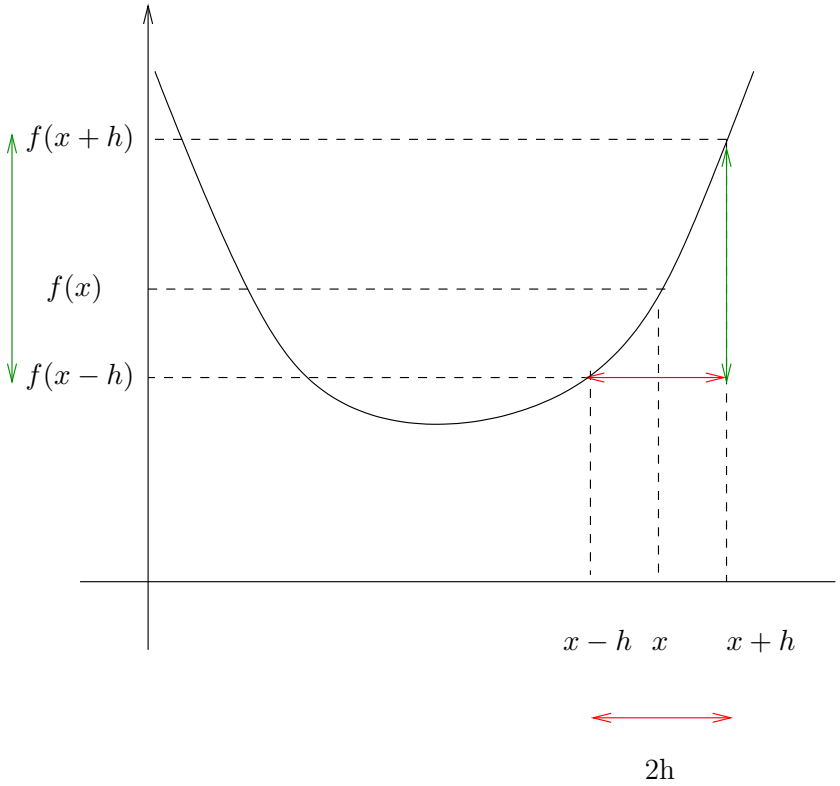

FIG. 1 : Taux d'accroissement

```
# epsilon;;
- : float = 1e-06
# derivee;;
- : (float -> float) -> float -> float -> float = <fun>
# derivee (fun x -> 4. *. x +. 3.) epsilon 10.;;
- : float = 3.99999999700639819
# derivee (fun x -> x *. x *. x +. 5.) epsilon 2.;;
- : float = 11.9999999999009788
```

On s'intéresse maintenant à la dérivée $n$ème $f^{(n)}$ d'une fonction f qui peut être définie par

$$\begin{cases} f^{(0)} = f \\ f^{(n)} = (f')^{(n-1)} \end{cases}$$

24. Écrire la fonction `derivee_n` qui prend en paramètre un entier n, f et h et retourne la fonction dérivée $n$ème (approchée) de f.

Exemples :

```
# derivee_n;;
- : int -> (float -> float) -> float -> float -> float = <fun>
# derivee_n 2 (fun x -> x *. x *. x +. 5.) epsilon 2.;;
- : float = 12.0015108961979422
```

FIN