université de BORDEAUX	**Année universitaire :** 2023/2024 **Parcours :** Licence Informatique 3e année UE 4TIN514U **Épreuve :** Devoir surveillé de Programmation fonctionnelle **Date :** mardi 24 octobre 2023 9h00 – 10h30 **Durée :** 1h20 **Documents interdits.**	**Collège Sciences et Technologies**

Le sujet comporte 4 pages.

Exercice 1 *(3.5pts)*

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

1. `3.14 + 0`
2. `let x = 4 in x + (let x = 3 and y = 2 in x + y)`
3. `let x = 4 in x + (let x = 3 and y = 2 * x in x + y)`
4. `let x = 4 in x + (let x = 3 in let y = 2 * x in x + y)`
5. `fun x y -> (x, x + y)`
6. `let f x y = 3 * x + 2 * int_of_float y`
7. `f 5.` où `f` est la fonction définie précédement.

Exercice 2 *(1.5pts)*

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

8. `int -> float -> float`
9. `'a * 'a -> bool`
10. `'a -> ('a -> bool) -> int`

On trouvera le type de chacune des fonctions demandées en dernière page.

Exercice 3 *(2pts)*

Le plus grand commun diviseur (pgcd) de deux nombres entiers non nuls peut se calculer en utilisant la méthode d'Euclide décrite ci-dessous :

$$pgcd(a,b) = \begin{cases} \text{si } b = 0 & \text{alors} & a \\ & \text{sinon} & pgcd(b, a \ mod \ b). \end{cases}$$

11. Implémenter la fonction `pgcd a b` en utilisant la méthode d'Euclide. Dans le cas où les deux nombres sont nuls, déclencher une erreur avec `failwith "pgcd(0,0)"`.

Exemples :

```
# pgcd 12 30;;
- : int = 6
# pgcd 0 5;;
- : int = 5
# pgcd 0 0;;
Exception: Failure "pgcd(0,0)".
```

À noter qu'en `OCaML`, l'opérateur **infixe** pour le modulo est `mod`. Exemples :

```
# 10 mod 3;;
- : int = 1
# (mod);;
- : int -> int -> int = <fun>
```

Exercice 4 *(5.5pts)*

Soit $\mathbb{Q}^+$ l'ensemble des nombres rationnels positifs ou nuls, c'est-à-dire l'ensemble des nombres qui peuvent s'écrire sous la forme $\frac{n}{d}$ avec $n \in \mathbb{N}$ et $d \in \mathbb{N}^* = \mathbb{N} - \{0\}$ (ensemble de entiers naturels strictement positifs). Pour représenter un nombre rationnel, on utilise le type `ratio` suivant :

```
type ratio = Ratio of int * int
```

On n'utilisera que la forme *normale* d'un nombre rationnel, c'est-à-dire la forme telle que $pgcd(n, d) = 1$. Ainsi, on n'utilisera pas $\frac{12}{30}$ mais $\frac{2}{5}$. De manière générale, on utilisera les formes $\frac{n/p}{d/p}$ où $p = pgcd(n, d)$. Un entier n sera représenté par $\frac{n}{1}$.

12. Écrire une fonction constructeur `make_ratio n d` qui construit le rationnel $\frac{n}{d}$ dans sa forme normale. On pourra utiliser la fonction `pgcd` de l'exercice précédent. On déclenchera une erreur avec `failwith «Division by zero»` dans le cas où le dénominateur d est nul.
13. Écrire les accesseurs `numerateur r` et `denominateur r` qui retournent respectivement le numérateur et le dénominateur d'un nombre rationnel `r`.

    Exemples :

```
# make_ratio 12 30;;
- : ratio = Ratio (2, 5)
# make_ratio 0 4;;
- : ratio = Ratio (0, 1)
# make_ratio 5 0;;
- : ratio = Ratio (1, 0)
# make_ratio 0 0;;
Exception: Failure "pgcd(0,0)".
# let r = make_ratio 12 30;;
val ratio : r = Ratio (2, 5)
# numerateur r;;
- : int = 2
# denominateur r;;
- : int = 5
```

14. Écrire la fonction `float_of_ratio r` qui donne la valeur approchée du rationnel `r`.

```
# float_of_ratio (make_ratio 12 30);;
- : float = 0.4
# float_of_ratio (make_ratio 8 6);;
- : float = 1.33333333333333326
```

15. Écrire le prédicat `ratio_infeg r1 r2` qui retourne `true` si $r1 \leqslant r2$ et `false` sinon.

```
# let r1 = make_ratio 12 30;;
val r1 : ratio = Ratio (2, 5)
# let r2 = make_ratio 8 6;;
val r2 : ratio = Ratio (4, 3)
# ratio_infeg r1 r2;;
- : bool = true
# ratio_infeg r2 r1;;
- : bool = false
```

16. Écrire la fonction `ratio_sum r1 r2` qui retourne le rationnel somme des rationnels `r1` et `r2`.

```
# ratio_sum (make_ratio 3 5) (make_ratio 4 3);;
- : ratio = Ratio (29, 15)
```

Exercice 5 *(7.5pts)*

On souhaite représenter des sous-ensembles potentiellement infinis de $\mathbb{Q}^+$. Pour gérer l'infinitude, un sous-ensemble est représenté par sa **fonction caractéristique** qui s'applique à un rationnel et retourne un booléen (vrai si le rationnel appartient à l'ensemble et faux sinon). On utilise le type

```
type rset = ratio -> bool
```

Ainsi, on peut définir la variable `rset_empty` qui contient le sous-ensemble vide s'écrit :

```
let rset_empty : rset = fun r -> false
```

et la fonction `rset_member r rset` qui retourne `true` si un rationnel `r` appartient à l'ensemble `rset` et `false` sinon :

```
let  rset_member r rset = rset r
```

17. Définir la variable `rset_full` contenant l'ensemble de tous les rationnels.
18. Écrire la fonction `rset_make_interval r1 r2` qui retourne l'ensemble des rationnels compris entre les deux rationnels `r1` et `r2`. On supposera `r1` $\leqslant$ `r2`.
19. Écrire la fonction `rset_make_singleton r` qui retourne l'ensemble contenant uniquement le rationnel `r`.

Exemples :

```
# rset_member (make_ratio 3 4) rset_full;;
- : bool = true
# let interval = rset_make_interval (make_ratio 2 3) (make_ratio 25 4);;
val interval : ratio -> bool = <fun>
# rset_member (make_ratio 7 2) interval;;
- : bool = true
# rset_member (make_ratio 0 2) interval;;
- : bool = false
# rset_member (make_ratio 3 4) (rset_make_singleton (make_ratio 3 4));;
- : bool = true
# rset_member (make_ratio 4 3) (rset_make_singleton (make_ratio 3 4));;
- : bool = false
```

20. Écrire la fonction `rset_complement rset` qui retourne le complémentaire de l'ensemble `rset` dans $\mathbb{Q}$, c'est-à-dire l'ensemble des rationnels qui ne sont pas dans `rset`.
21. Écrire la fonction `rset_intersection rset1 rset2` qui retourne l'intersection des deux ensembles de rationnels `rset1` et `rset2`.

Exemples :

```
# let interval = rset_make_interval (make_ratio 2 3) (make_ratio 25 4);;
val interval : ratio -> bool = <fun>
# rset_member (make_ratio 8 1) (rset_complement interval);;
- : bool = true
# rset_member
    (make_ratio 7 3)
    (rset_intersection
      (rset_make_interval (make_ratio 0 1) (make_ratio 4 1))
      (rset_make_interval (make_ratio 2 1) (make_ratio 13 2)));;
- : bool = true
```

22. Écrire la fonction `rset_remove_if pred rset` qui retourne l'ensemble de rationnels `rset` duquel on a "supprimé" les éléments qui vérifient le prédicat `pred`.
23. Écrire une expression qui retourne l'ensemble des rationnels privé des entiers[1].
24. Écrire la fonction `rset_find_smaller_int rset n` pour `rset` un ensemble de rationnels positifs et `n` un entier qui retourne le plus grand entier de `rset` strictement plus petit que l'entier `n` s'il existe et `n` sinon.

Exemples :

```
# let interval =  rset_make_interval (make_ratio 2 1) (make_ratio 9 2);;
val interval : ratio -> bool = <fun>
# rset_member
    (make_ratio 7 2)
    (rset_remove_if (fun r -> numerateur r = 7) interval);;
- : bool = false
```

[1] Dans notre représentation, un entier est un rationnel dont le dénominateur est 1.

```
# rset_find_smaller_int interval 10;;
- : int = 4
# rset_find_smaller_int interval 1;;
- : int = 1
```

FIN

```
val pgcd : int -> int -> int
type ratio = Ratio of int * int
val make_ratio : int -> int -> ratio
val numerateur : ratio -> int
val denominateur : ratio -> int
val float_of_ratio : ratio -> float
val ratio_infeg : ratio -> ratio -> bool
val ratio_sum : ratio -> ratio -> ratio
type rset = ratio -> bool
val rset_empty : rset
val rset_full : rset
val rset_member : ratio -> rset -> bool
val rset_complement : rset -> ratio -> bool
val rset_make_interval : ratio -> ratio -> rset
val rset_make_singleton : ratio -> rset
val rset_union : rset -> rset -> rset
val rset_intersection : rset -> rset -> rset
val rset_difference : rset -> rset -> rset
val rset_adjoin : ratio -> rset -> rset
val rset_remove_if : (ratio -> bool) -> rset -> rset
val rset_find_smaller_int : rset -> int -> int
```