Université de Bordeaux **Licence S&TS Mention Informatique, S5**
4TIN514U **2023/2024**

Programmation fonctionnelle
TP noté du mardi 5 Décembre
Durée : 1h20.

Le barème est donné à titre **indicatif**. Le sujet comporte 4 pages.

---

a) Télécharger le fichier `student.ml` depuis Moodle. Le travail doit être effectué dans ce fichier `student.ml` lequel doit être déposé sur Moodle à l'issue du TP. Ce fichier contient un barème indicatif.

b) Renseignez vos nom, prénom et numéro de groupe.

c) Pensez à sauver régulièrement votre travail.

d) Il est recommandé d'utiliser les fonctions du module `List` chaque fois que c'est approprié.

e) Il y a des exemples pour chaque fonction demandée.

---

1 Préléminaires

1. Écrire une fonction `union` de préférence **récursive terminale** de type
   `'a list -> 'a list -> 'a list`
   qui appliquée à deux listes sans doublons retourne l'union de ces deux listes sans doublons. L'ordre des éléments n'a pas d'importance. Exemple :

```
# union [2; 1; 5; 3] [2; 3; 4; 1];;
- : int list = [5; 2; 3; 4; 1]
```

2. Écrire une fonction `intersection`, de préférence **récursive terminale** de type
   `'a list -> 'a list -> 'a list`
   qui appliquée à deux listes sans doublons retourne l'intersection de ces deux listes sans doublons. L'ordre des éléments n'a pas d'importance. Exemple :

```
# intersection [2; 1; 5; 3] [2; 3; 4; 1];;
- : int list = [3; 1; 2]
```

2 Nombres rationnels positifs

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). 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 $\frac{n}{1}$.

Pour représenter un nombre rationnel, on utilise le type `ratio` suivant :

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

Le constructeur `Int` servira pour les rationnels qui sont des entiers tandis que le constructeur `Ratio` servira pour les rationnels dont le dénominateur est différent de `1` (une fois la fraction réduite).

3. 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 son choix.

4. Écrire les accesseurs `numerateur r` et `denominateur r` qui retournent respectivement le numérateur et le dénominateur d'un nombre rationnel `r`.

Exemples :

```
# 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
# let i = make_ratio 9 3
val i : ratio = Int 3
# denominateur i;;
- : int = 1
```

5. Écrire le prédicat `ratio_sup r1 r2` qui retourne `true` si $r1 > r2$ 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_sup r1 r2;;
- : bool = false
# ratio_sup r2 r1;;
- : bool = true
```

6. Écrire la fonction `ratio_prod r1 r2` de type `ratio -> ratio -> ratio` qui retourne le rationnel produit des rationnels `r1` et `r2`.

```
# ratio_prod (make_ratio 3 5) (make_ratio 4 3);;
- : ratio = Ratio (4, 5)
```

3 Ensembles de nombres rationnels positifs

On souhaite représenter des sous-ensembles de $\mathbb{Q}^+$. On va considérer deux implémentations.

La première consiste à représenter le sous-ensemble 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).

```
type rset_fun = ratio -> bool
```

La deuxième consiste à représenter l'ensemble par une liste de rationnels.

```
type rset_list = ratio list
```

Soit l'implémentation avec le type `rset_fun` donnée en Annexe Page 4.

7. Parmi les neuf opérations, quelles sont celles qu'il n'est pas possible d'implémenter avec le type `rset_list` et pourquoi ?

8. Écrire les opérations (neuf au maximum) qu'il est possible d'implémenter avec le type `rset_list` .

9. Écrire une fonction `rset_of_couples` de préférence récursive terminale de type `(int * int) list -> rset_list`
   qui prend une liste de couples (n, d) et fabrique l'ensemble contenant les rationnels n/d. L'ordre n'a pas d'importance. Exemple :

```
# rset_of_couples [(1,2); (3,4); (5,3); (6,2); (5,3); (6,8)];;
- : rset_list = [Int 3; Ratio (5, 3); Ratio (3, 4); Ratio (1, 2)]
```

10. Écrire une fonction `rset_product` de préférence récursive terminale de type `rset_list -> ratio`
    qui prend en paramètre un ensemble de rationnels et retourne le rationnel produit de tous les rationnels de l'ensemble. Exemple :

```
utop[13]> rs;;
- : ratio list = [Int 3; Ratio (5, 3); Ratio (3, 4); Ratio (1, 2)]
utop[14]> rset_product rs;;
- : ratio = Ratio (15, 8)
```

Annexe

```
type rset_fun = ratio -> bool

let rset_empty : rset_fun = fun r -> false

let rset_member r (rset : rset_fun) = rset r

let rset_complement rset = fun r -> not (rset_member r rset)

let rec rset_make_interval r1 r2 : rset_fun = (* [r1, r2] *)
  fun r -> ratio_infeg r1 r && ratio_infeg r r2

let rset_make_singleton r = rset_make_interval r r

let rset_adjoin rat (rset : rset_fun) : rset_fun =
  fun r -> r = rat || rset_member r rset

let rset_intersection rset1 rset2 : rset_fun =
  fun r -> rset_member r rset1 && rset_member r rset2

let rset_union rset1 rset2 : rset_fun =
  fun r -> rset_member r rset1 || rset_member r rset2

let rset_remove_if p rset : rset_fun =
  fun r -> p r && rset_member r rset
```

FIG. 1 : Implémentation par fonction caractéristique