université de BORDEAUX	**Année universitaire :** 2020/2021 **Parcours :** Licence Informatique 2e année UE 4TINA01U **Épreuve :** Examen de Programmation fonctionnelle **Date :** Lundi 4 janvier 2021 11 :30 – 13 :00 **Durée :** 1h30 **Documents interdits.**	**Collège Sciences et Technologies**

Exercice 1 *(4pts)* Soit la fonction `mystere` suivante :

```
let remove e l =
  let rec aux l nl =
    match l with
      [] -> List.rev nl
    | h :: tl -> if h = e then aux tl nl
                 else aux tl (h :: nl)
  in aux l []

let rec mystere l =
  match l with
    [] -> []
  | x :: t ->  x :: (mystere (remove x t))
```

1. Donner son type.
2. Que retourne l'appel suivant (valeur et type) ?

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

3. Que fait la fonction `mystere` ?
4. Donner une version récursive terminale de `mystere`.

Exercice 2 *(3pts)*

5. Écrire une fonction `couples_elements couples` qui retourne la liste des éléments apparaissant dans les couples de la liste `couples` sans doublons. On pourra utiliser la fonction `remove_duplicates l` qui retourne la liste `l` sans doublon (sans l'écrire) ainsi que la fonction `List.fold_right`.

   Exemples :

```
utop[16]> List.fold_right;;
- : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b = <fun>
utop[17]> couples_elements;;
- : ('a * 'a) list -> 'a list = <fun>
utop[18]> couples_elements [];;
- : 'a list = []
utop[19]> couples_elements [(1, 2); (2, 3); (4, 2)];;
- : int list = [1; 2; 3; 4]
```

Relations binaires

On souhaite manipuler des *relations binaires* sur l'ensemble des entiers naturels autrement dit des sous-ensembles de $\mathbb{N} \times \mathbb{N}$. On considère l'ensemble des fonctions de l'API (Application Program Interface) donné en Annexe page 3 permettant de construire et de manipuler des relations. On trouvera aussi en Annexe, une utilisation de cette API. Le typage explicite des fonctions n'est pas obligatoire.

Exercice 3 *(4pts)*

6. En supposant la fonction `rel_add` connue, implémenter la fonction `rel_adds`.
7. En utilisant la fonction `rel_adds`, implémenter la fonction `add_reflexive_couples elements relation` qui retourne la relation qui contient tous les couples de `relation` plus les couples (i, i) pour tout i appartenant à la liste `elements`. On pourra utiliser `List.map`.
8. En utilisant, la fonction `rel_elements couples` et la fonction `add_reflexive_couples elements relation` du point précédent, implémenter la fonction `rel_make_reflexive relation`.

Exercice 4 *(2pts)* Dans un premier temps, on va implémenter les relations à l'aide de listes. Ainsi une relation sera représentée par le type

```
type 'a relation = ('a * 'a) list
```

La relation vide est représentée par la liste vide et on peut définir, en utilisant la fonction `couples_elements couples` vue précédemment, la variable `rel_empty` et la fonction `rel_elements relation` par le code suivant :

```
utop[21]> let rel_empty = ([] : relation);;
val rel_empty : 'a relation = []
utop[22]> let rel_elements (relation : relation) = couples_elements relation;;
val rel_elements : 'a relation -> 'a list = <fun>
```

9. En utilisant la représentation par liste, implémenter la fonction `rel_add couple relation`. On pourra utiliser la fonction `List.mem`.

```
utop[26]> let r1 = rel_adds [(1,2); (2,3); (4,1)] rel_empty;;
val r1 : int relation = [(4,1); (2,3); (1,2)]
utop[27]> rel_add (2,3) r1;;
- : int relation = [(4,1); (2,3); (1,2)]
utop[28]> rel_add (3,2) r1;;
- : int relation = [(3,2); (4,1); (2,3); (1,2)]
```

Exercice 5 *(7pts)* On propose une deuxième implémentation dans laquelle une relation est représentée par la liste des éléments et une fonction booléenne s'appliquant à un couple et qui retourne vrai si le couple appartient à la relation et faux sinon.

On utilisera le type suivant :

```
type 'a relation = { rel_elts : 'a list; rel_fun: 'a * 'a -> bool }
```

- le champ `rel_elts` contient la liste des éléments de la relation.
- le champ `rel_fun` est un prédicat s'appliquant à un couple et qui retourne `true` si le couple appartient à la relation et `false` sinon.

Ainsi, la variable `rel_empty` et la fonction `rel_elements` s'écriront :

```
let rel_empty = { rel_elts = []; rel_fun = fun c -> false}
let rel_elements relation = relation.rel_elts
```

En utilisant la représentation fonctionnelle, implémenter les fonctions suivantes :

10. `rel_member`,
11. `rel_add`,
12. `rel_make_symmetric`.

FIN

Annexe

API pour les relations binaires

`rel_empty`	la relation vide.
`rel_add couple relation`	la relation contenant tous les couples de la relation `relation` ainsi que le couple `couple`.
`rel_adds couples relation`	la relation contenant tous les couples de `relation` ainsi que les couples de la liste `couples`.
`rel_member couple relation`	vrai si `couple` appartient à `relation`, faux sinon.
`rel_elements relation`	ensemble (sous forme de liste **sans contrainte sur l'ordre**) des entiers apparaissant dans `relation`.
`rel_make_reflexive relation`	relation contenant tous les couples de `relation` ainsi que tous les couples (i, i) pour tout $i \in$ `rel_elements relation`.
`rel_make_symmetric relation`	relation contenant tous les couples (i, j) de `relation` ainsi que le couple symétrique (j, i).

Exemple d'utilisation de l'API (représentation avec liste)

```
utop[84]> let r1 = rel_adds [(1,2); (2,1); (3,3); (3,1)] rel_empty;;
val r1 : int relation = [(3,1); (3,3); (2,1); (1,2)]
```

Exemple d'utilisation de l'API (représentation avec liste d'éléments et fonction)

```
utop[84]> let r1 = rel_adds [(1,2); (2,1); (3,3); (3,1)] rel_empty;;
val r1 : int relation = {rel_elements = [3; 2; 1]; rel_fun = <fun>}
```

Exemple d'utilisation de l'API valables dans les deux implémentations

```
utop[85]> rel_elements r1;;
- : int list = [3; 2; 1] (* l'ordre n'a pas d'importance *)
utop[86]> rel_member (2,1) r1;;
- : bool = true
utop[87]> rel_member (1,3) r1;;
- : bool = false
utop[91]> rel_member (1,1) (rel_make_reflexive r1);;
- : bool = true
utop[92]> rel_member (4,4) (rel_make_reflexive r1);;
- : bool = false
utop[93]> rel_member (1,3) (rel_make_symmetric r1);;
- : bool = true
utop[94]> rel_member (1,4) (rel_make_symmetric r1);;
- : bool = false
```