université de BORDEAUX	**Année universitaire :** 2023-2024 **Parcours :** Licence Informatique 3e année **UE :** Programmation fonctionnelle UE 4TIN514U **Épreuve :** Examen de Programmation fonctionnelle **Date :** Mardi 19 décembre 2023 11 :30 – 13 :00 **Durée :** 1h30 **Documents interdits.**	**Collège Sciences et Technologies**

Exercice 1 *(2pts)* Soit un programme `OCaML` constitué de deux modules indépendants `f1.ml` et `f2.ml` et d'un programme principal `main.ml`.

1. Donner une suite de commandes permettant d'obtenir un exécutable nommé `main`.

Exercice 2 *(6pts)*

Soient les deux fonctions `mystere1` et `mystere2` données Fig. 2 en annexe page 4.

2. Que retourne l'appel `mystere1 [1; 2; 3; 4; 5] (fun n -> n mod 2 = 0)` ?
3. Que retourne l'appel `mystere2 [1; 2; 3; 4; 5] (fun n -> n mod 2 != 0)` ?
4. Donner le type de la fonction `mystere2`.
5. Les fonctions `mystere1` et `mystere2` sont-elles équivalentes (même effets pour mêmes arguments) ?
6. Décrire en une phrase ce que fait la fonction `mystere2` (ce qu'elle prend en entrée et ce quelle retourne en sortie).

Soient les deux appels à `time` présentés à la fin de la figure Fig. 3 en annexe page 4.

7. Expliquer la différence de comportement entre ces deux appels.
8. Écrire une version récursive terminale de la fonction `mystere2` **sans utiliser de fonction du module** `List`.
9. Donner une fonction du module `List` de `OCaML` équivalente à `mystere2`.

À partir de maintenant, il est autorisé (et recommandé) d'utiliser les fonctions du module `List` chaque fois que c'est approprié.

Exercice 3 *(4pts)*

10. Écrire une fonction `count` de type `'a -> 'a list -> int` (récursive ou pas) qui prend en paramètre un objet o de type `'a` et une liste l d'objets de type `'a` et qui retourne le nombre d'occurrences de o dans l.
    Si la fonction écrite est **récursive**, elle devra être **récursive terminale**. Exemples :

```
# count 1 [1; 1; 2; 1; 4; 1];;
- : int = 4
# count 3 [1; 1; 2; 1; 4; 1];;
- : int = 0
```

Un *2-ensemble* est un ensemble où les éléments peuvent apparaître avec une multiplicité d'au plus 2, c'est-à-dire 0, 1 ou 2 fois.

11. Écrire une fonction `list_to_2set` de type `'a list -> 'a list` qui transforme une liste d'objets en un 2-ensemble en supprimant les occurrences excédentaires des éléments qui apparaissent plus de deux fois. L'ordre des éléments n'est pas important. On pourra utiliser la fonction `count` définie précédemment. Exemple :

```
# list_to_2set [1; 1; 1; 2; 6; 3; 1; 2; 5; 2];;
- : int list = [1; 6; 3; 1; 2; 5; 2]
# list_to_2set [1; 1; 2; 6; 3; 5; 2];;
- : int list = [1; 1; 2; 6; 3; 5; 2]
```

Exercice 4 *(5pts)* On utilise le type récursif `monster` suivant pour représenter des *monstres* à plusieurs têtes (au moins une).

```
type monster = M of (monster list)
```

Chaque constructeur `M` correspond à une *tête* du monstre et prend comme argument la liste des *enfants* du monstre qui sont aussi des monstres.

On utilise le constructeur `make_monster` et l'accesseur `monster_children` associés au type `monster`.

```
let make_monster (children : monster list)  = M children
let monster_children m = let M children = m in children
```

On peut ainsi construire des monstres :

```
# let m1 = make_monster [];;
val m1 : monster = M []
# let m2 = make_monster [m1; m1; m1; m1];;
val m2 : monster = M [M []; M []; M []; M []]
# let m3 = make_monster [m1; m2; m1];;
val m3 : monster = M [M []; M [M []; M []; M []; M []]; M []]
```

Une représentation graphique des monstres est présentée Fig. 1 page 2.

L'*arité* d'un monstre est le nombre de ses enfants autrement dit la longueur de la liste de ses enfants.

12. Écrire une fonction `monster_arity` de type `monster -> int` qui retourne l'arité d'un monstre.

La *taille* d'un monstre est définie par son nombre de têtes c'est-à-dire le nombre de constructeurs `M` qui la constituent.

13. Écrire une fonction `monster_size` de type `monster -> int` qui retourne la taille d'un monstre.

L'*arité maximum* d'un monstre est le maximum de son arité et de celle de tous ses descendants.

14. Écrire une fonction `monster_max_arity` de type `monster -> int` qui retourne l'arité maximum d'un monstre.

```
# monster_arity m2;;
- : int = 4
# monster_arity m1;;
- : int = 0
# monster_arity m3;;
- : int = 3
# monster_size m1;;
- : int = 1
# monster_size m2;;
- : int = 5
# monster_max_arity m3;;
- : int = 4
```

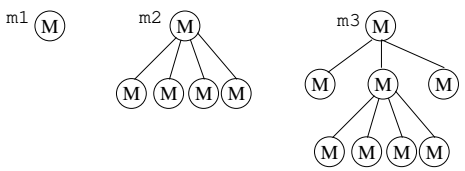

FIG. 1 : Représentation graphique des monstres

Exercice 5 *(3pts)*

15. Écrire une fonction `dict_of_keys_and_values` de type `'a -> list 'b -> list -> 'a -> 'b option` qui prend en paramètre une liste de clés $keys = k_1, \ldots, k_n$ et une liste de valeurs $values = v_1, \ldots, v_m$ et qui retourne une **fonction** qui
    - appliquée à une des clés k_i retourne `Some` v_i si v_i existe (soit si $i \leqslant m$) et `None` sinon,
    - et appliquée à toute clé n'appartenant pas à la liste des clés $keys$ retourne `None`.

Exemples :

```
# let dico = dict_of_keys_and_values [1; 2; 3; 4] ["un"; "deux"; "trois"];;
val dico : int -> string option = <fun>
# dico 2;;
- : string option = Some "deux"
# dico 4;;
- : string option = None
# dico 5;;
- : string option = None
```

Annexe

```
let rec mystere1 l f =
  match l with
    [] -> l
  | e :: t ->
     let r = mystere1 t f in
     if f e then r @ [e]
     else r

let  mystere2 l f =
  let rec aux l =
    match l with
      [] -> []
    | e :: t -> let r = aux t in
                if f e then e :: r
                else r
  in List.rev (aux l)
```

FIG. 2 : Fonctions `mystere1` et `mystere2`

```
# let time f =
  let start = Sys.time() in
  let _ = f () in
  Sys.time() -. start;;
val time : (unit -> 'a) -> float = <fun>
# let l1000 = List.init 1000 (fun i -> i);;
val l1000 : int list =
[0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17; 18; 19; 20; ...]
# let time f =
  let start = Sys.time() in
  let _ = f () in
  Sys.time() -. start
;;
val time : (unit -> 'a) -> float = <fun>
# time (fun _ -> mystere1 l1000 (fun e -> e mod 2 = 0));;
- : float = 0.00241699999999944737
# time (fun _ -> mystere2 l1000 (fun e -> e mod 2 = 0));;
- : float = 4.50000000000727596e-05
```

FIG. 3 : Test des fonctions `mystere1` et `mystere2`

FIN