université de BORDEAUX	**Année Universitaire 2018/2019** Licence Informatique & MI L2, UE 4TINA01U, Programmation fonctionnelle. Devoir surveillé — Vendredi 16 novembre 2018. 8h–9h30 Durée: 1h30. **Documents interdits.**	**Collège Sciences et Technologies**

L'évaluation attachera une grande importance à la clarté des justifications.
Vous pouvez utiliser les fonctions du module `List`.
Le barème est indicatif.

Exercice 1 — 5pts. Soit la fonction `mystere e l` ci-dessous.

```
let rec mystere e l =
  match l with
    []    -> 0
  | x::tl -> let r = mystere e tl in
             if e = x
             then 1 + r
             else r
```

1) Quel est le type de cette fonction ?
2) Donner un exemple d'appel à cette fonction et sa valeur de retour.
3) Que fait cette fonction (la réponse doit être justifiée) ?
4) Écrire une version récursive terminale de cette fonction.

Solution. Cette fonction compte le nombre d'occurrences de `x` dans la liste `l`.

```
utop[125]> mystere 1 [1; 2; 3; 1; 4; 5; 1];;
- : int = 3

let mystere x l =
  let rec aux l c =
    match l with
      [] -> c
    | e::tl -> aux tl (if x = e then c + 1 else c) in
  aux l 0
```

Exercice 2 — 4pts.

1) Écrire une fonction `remove_n n x l` de type

```
int -> 'a -> 'a list -> 'a list
```

qui retourne la liste des éléments de `l` de laquelle :

- on a supprimé exactement `n` éléments de `l` égaux à `x`, si `l` contient au moins `n` éléments égaux à `x`,
- on a supprimé ***tous*** les éléments égaux à `x` dans la liste `l`, sinon.

Seul le nombre d'occurrences est important ; l'ordre des éléments ne l'est pas.

Exemples :

```
remove_n 0 5 [1; 2; 5; 3; 5; 4; 6];;
- : int list = [1; 2; 5; 3; 5; 4; 6]

remove_n 1 5 [1; 2; 5; 3; 5; 4; 6];;
- : int list = [1; 2; 3; 5; 4; 6]
```

```
remove_n 2 5 [1; 2; 5; 3; 5; 4; 6];;
- : int list = [1; 2; 3; 4; 6]

remove_n 3 5 [1; 2; 5; 3; 5; 4; 6];;
- : int list = [1; 2; 3; 4; 6]
```

2) Évaluer le nombre d'appels récursifs en fonction de `n` et de la taille de `l`, dans le meilleur des cas et dans le pire des cas (selon la place des éléments à supprimer). Une réponse non justifiée à cette question ne rapporte pas de point.

Solution.

```
let remove_n n x l =
  let rec aux n l nl =
  if n = 0 then l @ nl
  else
    match l with
      [] -> nl
    | e::tl -> if e = x then aux (n - 1) tl nl else aux n tl (e::nl)
in aux n l []
```

Dans le cas le pire, on examine chacun des éléments de la liste et on fait donc $len(l)$ d'appels récursifs. Dans le meilleur des cas, les éléments à supprimer sont en début de liste et on fait $min(n\,, len(l))$ appels récursifs.

Exercice 3 — 6pts.

1) Écrire une fonction `trinome a0 a1 a2` prenant comme paramètres 3 entiers a_0, a_1 et a_2 et qui retourne la ***fonction*** qui à tout entier x associe $a_0 + a_1x + a_2x^2$.
2) Quel est le type de la fonction `trinome` ? Expliquer votre réponse.
3) On veut écrire une fonction `polynome` prenant en paramètre une liste d'entiers `[a0; a1; ... ; an]` et qui retourne la ***fonction*** qui à tout entier x associe $a_0 + a_1x + \cdots a_nx^n$. Par exemple, l'appel `polynome [1;2;5]` retournera la fonction $x \longmapsto 1 + 2x + 5x^2$.
   Remarque. `polynome []` est la fonction qui à tout entier x associe la valeur 0.
   a. On note f la fonction retournée par l'appel `polynome [a0; a1; ... ; an]` et g la fonction retournée par l'appel `polynome [a1; ... ; an]`. Calculer $f(x)$ en fonction de $g(x)$.
   b. Écrire la fonction `polynome`.
4) Écrire une version récursive terminale de la fonction `polynome`.

Solution.

```
let trinome a0 a1 a2 =
  fun x ->  a2 * x * x + a1 * x + a0

- : int -> int -> int -> int -> int = <fun>
```

```
let rec polynome coefficients =
  fun x ->
    match coefficients with
      [] -> 0
    | a0::ais -> a0 + x * (polynome ais x)

let polynome_rt coefficients =
  let rec aux l f =
    match l with
      [] -> f
    | an::ais -> aux ais (fun x -> an + x * (f x))
in aux (List.rev coefficients) (fun x -> 0)
```

Exercice 4 — 5pts.

1) Définir un type `carburant` ayant trois constructeurs `Diesel`, `Essence` ou `Electrique`.
2) Un *véhicule* est caractérisé par son carburant et son nombre de roues. Définir un type `vehicule` répondant à ces critères.
3) Lors des pics de pollution, les véhicules diesel à 4 roues au moins sont interdits. Écrire une fonction `peut_rouler : vehicule -> bool` qui teste si un véhicule est autorisé.
4) Pour rouler 100km, un véhicule électrique consomme environ 10kWh, un véhicule diesel consomme environ 6L de carburant, et un véhicule essence consomme environ 8L. Sachant qu'1kWh coûte 0.25€ et qu'un litre de carburant coûte 1.5€, écrire une fonction `consommation : vehicule -> int -> float` telle que `consommation v n` renvoie le coût d'utilisation du véhicule `v` sur `n` kilomètres.
5) Ajouter un constructeur au type `carburant` de façon à prendre en compte les véhicules hybrides (pouvant fonctionner avec deux types de carburants), et donner un exemple de véhicule hybride.

Solution.

```
type carburant = Diesel | Essence | Electrique

type vehicule = carburant * int

let peut_rouler carburant nb_roues =
  match carburant with
    Diesel -> nb_roues < 4
  | _ -> true

let calcul_consommation nb_km nb_litres prix_litre =
  (float_of_int nb_km) *. nb_litres *. prix_litre /. 100.

let consommation vehicule nb_km =
  match vehicule with
    Electrique -> calcul_consommation nb_km 10.0 0.25
  | Diesel -> calcul_consommation nb_km 6. 1.5
  | Essence -> calcul_consommation nb_km 8. 1.5

type carb = Diesel | Essence | Electrique | Hybride of carb * carb
```