| université de BORDEAUX | **Année Universitaire 2018/2019**<br>Licence Informatique & MI L2, UE 4TINA01U, Programmation fonctionnelle.<br>Examen Session 1 — Lundi 7 janvier 2019. 9h–10h30 Durée: 1h30.<br>**Documents interdits.** | **Collège Sciences et Technologies** |
|---|---|---|

**La notation tiendra compte de la clarté du code, des commentaires et des justifications.**

**Vous pouvez utiliser les fonctions du module `List`, ainsi que toute fonction d'une question précédente, même si vous ne l'avez pas traitée.**

**Le barème est indicatif.**

### Exercice 1 — 4pts.

1. Écrire un type `couleur` ayant trois valeurs : `Blanc`, `Rouge` et `Rose`.
2. On définit le type `region` comme suit :

   ```
   type region = Medoc | Graves | Alsace | Beaujolais | Touraine | Bourgogne
   ```

   En utilisant les types `couleur` et `region`, définir un type `vin` permettant de représenter un vin par
   - sa région,
   - sa couleur, **et**
   - son millésime (c'est-à-dire, une année, de type `int`).
3. Écrire une fonction `bordeaux` qui prend en paramètre une liste de vins et renvoie la sous-liste des vins de Bordeaux (c'est-à-dire produits dans le Médoc ou dans les Graves) qu'elle contient.
4. Écrire une fonction `millesimes` qui prend en paramètres une liste de vins `l` et un prédicat `p` de type `vin -> bool`, et qui renvoie la liste des millésimes des vins de `l` qui satisfont `p`. Donner un exemple d'appel de votre fonction, ainsi que le type et la valeur de son retour.

### Exercice 2 — 7pts.

1. Écrire une fonction `distribute` de type `'a -> 'b list -> ('a * 'b) list` telle que l'appel

   `distribute e [`$e_1$`; ...;`$e_n$`]`

   retourne la liste des couples `[(e,` $e_1$`); (e,` $e_2$`); ...; (e,` $e_n$`)]`. Par exemple :

   ```
   distribute 0 [];;
   - : (int * 'a) list = []
   distribute 0 [1; 2; 3];;
   - : (int * int) list = [(0, 1); (0, 2); (0, 3)]
   ```

2. Soit maintenant la fonction `cartesian_product` définie ci-dessous :

   ```
   let rec cartesian_product l1 l2 =
     match l1 with
       []     -> []
     | hd::tl ->  (distribute hd l2) @ (cartesian_product tl l2)
   ```

   *a.* Quel est le type de `cartesian_product` ?

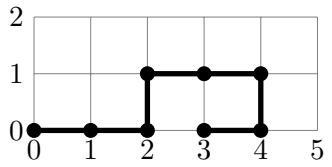
   *b.* Quelle est la valeur de `cartesian_product [] []` ?

   *c.* Quels sont le type et la valeur de `cartesian_product [1; 2; 3] [4.5; 7.0]` ?
3. De manière générale, si `l1 = [`$x_1$`;...; `$x_n$`]` et `l2 = [`$y_1$`;...; `$y_p$`]` que retourne `cartesian_product l1 l2` ?
4. Écrire une version récursive terminale de la fonction `cartesian_product`.

**Exercice 3 — 9pts.** Pour représenter des chemins dans le plan partant de l'origine et faisant des pas d'une unité vers le Nord (`N`), le Sud (`S`), l'Est (`E`) ou l'Ouest `O`, on définit les types suivants :

```
type direction = N | S | E | O
type chemin    = direction list
type point     = int * int
```

Un chemin commence au point de coordonnées $(0\,,0)$, et passe donc par des points de coordonnées entières. Par exemple `[E;E;N;E;E;S;O]` passe par (0,0), (1,0), (2,0), (2,1), (3,1), (4,1), (4,0), (3,0).




1. Écrire une fonction `arrivee` de type `chemin -> point` qui retourne le point d'arrivée du chemin passé en paramètre. Par exemple, `arrivee []` doit retourner `(0,0)` et `arrivee [E;E;N;E;E;S;O]` doit retourner `(3,0)`.
   **Note.** On peut si besoin récupérer les coordonnées individuelles d'un point `p` par **`let`** `(x,y) = p` **`in`**`...`
2. Écrire une fonction `coordonnees` de type `chemin -> point list` qui renvoie la liste des points visités le long du chemin. Notez que cette liste a un élément de plus que le nombre de pas dans le chemin. En particulier `coordonnees []` renvoie la liste `[(0,0)]`. De même, `coordonnees [E;E;N;E;E;S;O]` renvoie la liste `[(0,0); (1,0); (2,0); (2,1); (3,1); (4,1); (4,0); (3,0)]`.
3. En utilisant la fonction `coordonnees` (même si vous ne l'avez pas écrite), écrire une fonction `se_recoupe` de type `chemin -> bool` qui teste si un chemin rencontre deux fois le même point. Par exemple, `se_recoupe [N;N;E;S]` renvoie `false`, alors que `se_recoupe [E;N;E;S;O;S]` renvoie `true` (on passe deux fois par le point (1,0)) et `se_recoupe [N;S]` renvoie aussi `true` (on passe deux fois par (0,0)).
4. Écrire une fonction `symetrie` de type `chemin -> chemin` qui renvoie le chemin symétrique de son paramètre par rapport à l'axe des abscisses : les pas `N` sont changés en `S`, les pas `S` sont changés en `N`, et les deux autres pas sont inchangés.

Une *transformation* du plan associe à un point $(x\,,y)$ un autre point $(x'\,,y')$. Une translation de vecteur $(v_x\,,v_y)$ qui à un point $(x\,,y)$ associe le point $(x+v_x\,,y+v_y)$ est un exemple de transformation. On utilise des fonctions de type `point -> point` pour représenter les transformations du plan.

5. Implémenter la fonction `translation` de type `int -> int -> point -> point` telle que `translation vx vy` retourne la translation de vecteur `(vx, vy)`.
   Exemples :

```
translation 2 1;;
- : point -> point = <fun>
(translation 2 1) (3, 1);;
- : point = (5, 2)
(translation 2 1) (3, 6);;
- : point = (5, 7)
```

6. Écrire une fonction `points` de type `chemin -> int -> int -> point list` telle que `points c vx vy` retourne la liste de points obtenus à partir de `l = coordonnees c` en translatant chaque point de cette liste par le vecteur `(vx,vy)`.
7. **(Question bonus)** Écrire une fonction testant qu'un chemin décrit exactement le contour d'un carré dont chaque segment est parcouru une unique fois. Par exemple, la fonction doit retourner `true` sur l'argument `[N;N;E;E;S;S;O;O]` ainsi que sur sur `[E;N;N;O;O;S;S;E]`, et `false` sur `[E;S;O]` (le chemin ne forme pas un carré) et sur `[O;E;N;O;S]` (un segment est parcouru deux fois).