List Toolbox and Midterms

Ne pas utiliser l'opérateur @ ni les fonctions du module `List`.

1 Boite à outils

1.1 Basics

Certaines des fonctions de cette partie sont nécessaires pour le prochain TP !

Fichier à rendre (**et à conserver !**) : `list_tools.ml`.

1.1.1 `length*`

Écrire une fonction qui calcule la longueur d'une liste.

```
val length : 'a list -> int = <fun>

# length [0; 1; 0; 1; 0; 0; 0; 0; 1; 1] ;;
- : int = 10
```

1.1.2 `product`

Écrire la fonction `product` qui calcule le produit de tous les éléments d'une liste d'entiers.

```
val product : int list -> int = <fun>

# product [1; 2; 3; 4; 5] ;;
- : int = 120
# product [];;
- : int = 1
```

1.1.3 `nth*`

Écrire une fonction qui donne la valeur du $i^{ème}$ élément d'une liste. La fonction devra déclencher une exception `Invalid_argument` si i est négatif, ou une exception `Failure` si la liste est trop courte.

```
val nth : int -> 'a list -> 'a = <fun>

# nth 5 [1; 2; 3; 4; 5] ;;
Exception: Failure "nth: list is too short".
# nth 0 ['a'; 'b'; 'c'] ;;
- : char = 'a'
# nth (-5) [] ;;
Exception: Invalid_argument "nth: index must be a natural".
```

1.1.4 `search_pos`

Écrire une fonction qui retourne la position d'un élément dans une liste.

```
val search_pos : 'a -> 'a list -> int = <fun>

# search_pos 0 [1; 5; -1; 0; 8; 0] ;;
- : int = 3
# search_pos 'z' ['r'; 'h'; 'j'; 'o'] ;;
Exception: Failure "search_pos: not found".
```

1.1.5 `sum_digits`

Écrire la fonction `sum_digits` qui calcule la somme des chiffres d'un entier naturel.

```
# sum_digits 235 ;;
- : int = 10
# sum_digits 123456 ;;
- : int = 21
```

1.1.6 common

Écrire la fonction `common` qui à partir de deux listes d'entiers strictement croissantes retourne la première valeur qu'elles ont en commun, la valeur 0 si aucune valeur n'est trouvée.

```
# commun ([1;2;4;8;16],[5;10;16]);;
- : int = 16
# commun ([1;2;4;8;23],[5;10;16;31;42]);;
- : int = 0
```

1.2 Construire - Modifier

1.2.1 init_list

Écrire la fonction `init_list n x` qui crée une liste de `n` valeurs `x`.

```
val init_list : int -> 'a -> 'a list = <fun>

# init_list 5 0 ;;
- : int list = [0; 0; 0; 0; 0]
# init_list 0 'a' ;;
- : char list = []
# init_list (-5) 1.5 ;;
Exception: Invalid_argument "init_list: n must be a natural".
```

1.2.2 put_list

Écrire une fonction `put_list` v i *list* qui remplace la valeur en position i dans *list* par v si possible.

```
val put_list : 'a -> int -> 'a list -> 'a list = <fun>

# put_list 'x' 3 ['-'; '-'; '-'; '-'; '-'; '-'] ;;
- : char list = ['-'; '-'; '-'; 'x'; '-'; '-']
# put_list 0 10 [1; 1; 1; 1] ;;
- : int list = [1; 1; 1; 1]
```

1.3 'a list list

Dans la suite nous appellerons *matrice* (`board`) une liste de listes avec toutes les sous-listes de longueurs identiques.

Pour toutes les prochaines fonctions, voir les exemples d'applications ci-dessous !

1.3.1 init_board

Écrire la fonction `init_board` $(l,\ c)$ *val* qui retourne une *matrice* de taille $l \times c$ remplie de *val*.

```
val init_board : int * int -> 'a -> 'a list list = <fun>
```

1.3.2 get_cell

Écrire la fonction `get_cell` $(x,\ y)$ *board* qui retourne la valeur en position $(x,\ y)$ dans la matrice *board*.

```
val get_cell : int * int -> 'a list list -> 'a = <fun>
```

1.3.3 put_cell

Écrire la fonction `put_cell` *val* $(x,\ y)$ *board* qui remplace la valeur en $(x,\ y)$ dans la matrice *board* par la valeur *val*.
Si la case $(x,\ y)$ n'existe pas, *board* est retournée inchangée (pas d'exception).

```
val put_cell : 'a -> int * int -> 'a list list -> 'a list list = <fun>
```

Exemples

```
# let board = init_board (5, 3) 0;;
val board : int list list =
  [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]; [0; 0; 0]; [0; 0; 0]]

# let board = put_cell 1 (0, 0) board;;
val board : int list list =
  [[1; 0; 0]; [0; 0; 0]; [0; 0; 0]; [0; 0; 0]; [0; 0; 0]]

# let board = put_cell 2 (2, 1) board;;
val board : int list list =
  [[1; 0; 0]; [0; 0; 0]; [0; 2; 0]; [0; 0; 0]; [0; 0; 0]]

# get_cell (2, 1) board;;
- : int = 2
```

2 Old tests...

Fichier à rendre : `midterms.ml`.

2.1 Chargement de fichier

Pour charger les fonctions issues des fichiers précédents plus facilement, utiliser la directive *#use* dans le toplevel :

```
#use "file-name";;
Read, compile and execute source phrases from the given file. This is textual inclusion: phrases are pr
```

Par exemple,

```
#use "list_tools.ml";;
```

chargera toutes les définitions de la section 1.1 dans votre environnement CAML.

3 Notation positionnelle (`bases.ml`)

Extraits du cours d'architecture :

Un nombre n peut être représenté dans n'importe quelle base $b \geq 2$, utilisant b symboles. Chaque symbole possède une position numérotée. Dans cette *notation positionnelle*, la valeur d'un symbole dépend de sa position. Chaque *chiffre* de la représentation de l'entier n dans la base b, notée n_b, est associé à un poids, représentant b^p où p est la position du chiffre dans la représentation, de gauche à droite en partant de 0.

Habituellement les symboles utilisés sont les b premiers chiffres, suivis de lettres (dans le cas de bases $b > 10$).

Exemples :

— La base 2 utilise les symboles $0, 1$.
Décomposition en base 2 de 6 : 110 ($6 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$).

— La base 10 utilise les symboles $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.
Décomposition en base 10 de 2563 : 2563 ($2563 = 2 \times 10^3 + 5 \times 10^2 + 6 \times 10^1 + 3 \times 10^0$).

— La base 16 utilise les symboles $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$
Décomposition en base 16 de 2000 : $7D0$ ($2000 = 7 \times 16^2 + 13 \times 16^1 + 0 \times 16^0$).

— En base 4 (symboles $0, 1, 2, 3$), 297 sera représenté par 10321
($297 = 256 + 32 + 8 + 1 = 1 \times 4^4 + 0 \times 4^3 + 3 \times 4^2 + 2 \times 4^1 + 1 \times 4^0$)

Seule la position d'un symbole dans la base est importante pour connaître sa valeur.

Nous pouvons donc utiliser n'importe quel symbole pour écrire des nombres. Pour l'exercice nous représenterons une *base* par la liste de ses symboles (une liste de caractères, par exemple la base 16 classique sera représentée par `['0';'1';'2';'3';'4';'5';'6';'7';'8';'9';'A';'B';'C';'D';'E';'F']`). Donc la base sera égale au nombre d'éléments dans la liste. La représentation d'un entier sera donc également une liste de symboles.

Exemples :

— Base 2 : avec les symboles classiques la base sera représentée par la liste `['0' ; '1']` et l'entier 42_{10} dans cette base sera représenté par la liste `['1';'0';'1';'0';'1';'0']`
Si on choisit les symboles ['A' ; 'B'], l'entier 42_{10} sera représenté par `['B';'A';'B';'A';'B';'A']`.

— Dans la base 4 représentée par `['W';'X';'Y';'Z']`, l'entier 297_{10} sera représenté par `['X'; 'W'; 'Z'; 'Y'; 'X']`.

`recompose`

Écrire la fonction `recompose l b` qui retourne le nombre (entier naturel en base 10) recomposé à partir de sa représentation `l` en base `b` (liste de symboles).

```
val recompose : 'a list -> 'a list -> int = <fun>

# recompose ['1';'2';'0';'2';'7'] ['9';'8';'7';'6';'5';'4';'3';'2';'1';'0'];;
- : int = 87972
# recompose ['B';'A';'B';'A';'B';'A'] ['A' ; 'B'];;
- : int = 42
# recompose ['4';'2'] ['1';'0'];;
Exception: Failure "out of base".
```

`decompose`

Écrire la fonction `decompose x b` qui retourne le nombre `x` (entier naturel) décomposé en éléments de la base `b` (liste de symboles).

```
val decompose : int -> 'a list -> 'a list = <fun>

# decompose 42 ['0';'1';'2';'3';'4';'5';'6';'7';'8';'9'];;
- : char list = ['4'; '2']
# decompose 297 ['W'; 'X'; 'Y'; 'Z'];;
- : char list = ['X'; 'W'; 'Y'; 'Y'; 'X']
# decompose 42 ['A'; 'B'];;
- : char list = ['B'; 'A'; 'B'; 'A'; 'B'; 'A']
```

4 Représentation des polynômes par listes (`exo1.ml`)

◦ Un polynôme $a_n.x^n + ... + a_1.x + a_0$ peut être représenté par la liste $[a_0; a_1; ...; a_n]$ de ses coefficients [1].
La représentation proposée ci-dessus a l'inconvénient d'avoir une taille proportionnelle à l'ordre du polynôme et non pas au nombre de monômes non nuls.

Par exemple, le polynôme $4x^5 + x^2 - 3x + 1$ serait représenté par la liste :

```
[1; -3; 1; 0; 0; 4]
```

◦ Une autre solution (utilisée ici) consiste donc à représenter un polynôme par la liste des monômes. Un monôme $c.X^d$ est représenté par le couple d'entiers (c, d). Les monômes sont triés par degrés décroissants et chaque monôme a un coefficient non nul ($c \neq 0$).

1. Il peut sembler plus logique d'inverser l'ordre des coefficients, mais la manipulation sera beaucoup trop compliquée!

La liste vide représentera le polynôme nul.

Par exemple, le polynôme $4x^5 + x^2 - 3x + 1$ sera représenté par la liste :

```
# let poly = [(4,5); (1,2); (-3,1); (1,0)] ;;
 val poly : (int * int) list =
   [(4, 5); (1, 2); (-3, 1); (1, 0)]
```

1. Écrire la fonction `assoc` recherchant le coefficient associé à un degré n donné.
   Par exemple, sur le polynôme défini ci-dessus :

   ```
   # assoc 1 poly ;;
    - : int = -3

   # assoc 3 poly ;;
    - : int = 0
   ```

2. Écrire la fonction `add_poly` calculant la somme de deux polynômes. Le polynôme résultat devra être bien formé, c'est à dire :
   — il n'y a qu'un seul monôme par degré ;
   — il n'y a aucun coefficient nul dans la liste ;
   — les monômes sont triés par degrés décroissants.

   Exemple : $(4x^5 + x^2 - 3x + 1) + (2x^5 + 2x^4 - x^2 + x) = 6x^5 + 2x^4 - 2x + 1$

   ```
   # let poly1 = [(4,5) ; (1,2) ; (-3,1) ; (1,0)] ;;
    val poly1 : (int * int) list = [(4, 5); (1, 2); (-3, 1); (1, 0)]

   # let poly2 = [(2,5) ; (2,4) ; (-1,2) ; (1,1)] ;;
    val poly2 : (int * int) list = [(2, 5); (2, 4); (-1, 2); (1, 1)]

   # add_poly (poly1, poly2) ;;
    - : (int * int) list = [(6, 5); (2, 4); (-2, 1); (1, 0)]
   ```

5 Rivières numériques (`exo2.ml`)

Une rivière numérique est une suite de nombres où le nombre qui suit n est n augmenté de la somme des chiffres qui le constitue. Par exemple, 2341 est suivi par 2351 dans la mesure où 2+3+4+1=10. Si k est le premier nombre d'une rivière numérique, on l'appelle *la rivière k*.

Quelques exemples :
— La rivière 238 est la séquence de nombre commençant par {238, 251, 259, 275, ...}
— La rivière 480 est la séquence de nombre commençant par {480, 492, 507, 519, ...}
— La rivière 483 est la séquence de nombre commençant par {483, 498, 519, 534, ...}

Les vrais fleuves et les vraies rivières peuvent se rencontrer et il en va de même pour les rivières numériques. Cela arrive lorsque deux rivières numériques ont en commun au moins une valeur.
Par exemple : La rivière 480 rencontre la rivière 483 en 519 et la rivière 507 en 507. Par contre elle ne rencontre jamais la rivière 481.
Une rivière numérique finira toujours par rencontrer la rivière 1, la rivière 3 ou la rivière 9.

1. Utiliser `sum_digits` pour écrire la fonction `riviere` qui à partir de deux entiers n et *rang* tous deux positifs strictement (`Invalid_argument` devra être déclenchée dans le cas contraire) construit la rivière n jusqu'au rang *rang* : une liste de longueur *rang*.

   Exemples d'application :

   ```
   # riviere 238 5;;
    - : int list = [238; 251; 259; 275; 289]
   # riviere 42 10;;
    - : int list = [42; 48; 60; 66; 78; 93; 105; 111; 114; 120]
   ```

2. Écrire la fonction `rencontre` qui à partir de deux entiers n et *rang* indique si la rivière n a rencontré

une des trois rivières (parmi 1, 3 et 9) avant le rang *rang* : utiliser les fonctions `riviere` et `commun`. La fonction retournera :

— si la rencontre a lieu avant le rang donné : le couple (i, nb), où i est l'une des trois valeurs (1, 3 ou 9) et nb la valeur de rencontre
— sinon : le couple $(0, 0)$

Exemples d'application :

```
# rencontre 45 10;;
 - : int * int = (9, 45)
# rencontre 3 10;;
 - : int * int = (3, 3)
# rencontre 42 10;;
 - : int * int = (0, 0)
# rencontre 42 100;;
 - : int * int = (3, 111)
```