TP 3 : matrices

Ce TP est à déposer à la fin de cette séance sur moodle (https://foad.univ-rennes1.fr/course/view.php?id=1006723). Vous pouvez également déposer une version améliorée de votre TP jusqu'à la fin de cette semaine.

Fichiers à récupérer depuis moodle : `tp3.mlw`

Exercice 1 : matrices d'entiers

1. Lire la bibliothèque Why3 des matrices donnée ci-dessous (et consultable ici : http://why3.lri.fr/stdlib-1.1.0/matrix.html).

```
module Matrix
use int.Int
use map.Map as M
type matrix 'a = private { ghost mutable elts: int -> int -> 'a;
                           rows: int; columns: int }
   invariant { 0 <= rows /\ 0 <= columns }

predicate valid_index (a: matrix 'a) (r c: int) =
  0 <= r < a.rows /\ 0 <= c < a.columns

function get (m: matrix 'a) (r c: int) : 'a = m.elts r c

val get (m: matrix 'a) (r c: int) : 'a
requires { valid_index m r c }
ensures { result = m.elts r c }

val ghost function update (m: matrix 'a) (r c: int) (v: 'a) : matrix 'a
ensures { result.rows = m.rows }
ensures { result.columns = m.columns }
ensures { result.elts = m.elts[r <- (m.elts r)[c <- v]] }

val set (m: matrix 'a) (r c: int) (v: 'a) : unit
requires { valid_index m r c }
writes { m }
ensures { m.elts = (old m.elts)[r <- (old m.elts r)[c <- v]] }



val make (r c: int) (v: 'a) : matrix 'a
requires { r >= 0 /\ c >= 0 }
ensures  { result.rows = r }
ensures  { result.columns = c }
ensures  { forall i j. 0 <= i < r /\ 0 <= j < c -> get result i j = v }

end
```

---

2. Récupérer le fichier `tp3.mlw`. Compléter le module `TP31` afin de définir le type `mint` représentant des matrices d'entiers.
3. Dans le module de test `Test31`, compléter le sous-programme `testLib`, qui teste au moyen d'assertions les champs `rows` et `columns`, ainsi que les sous-programmes `set` et `make` définis dans la bibliothèque des matrices. Le sous-programme `testLib` :

- crée en utilisant la fonction `make` une matrice 3x4 remplie de zéros,
- modifie la case d'indice (0,0) de la matrice en lui affectant la valeur 2.
- Le sous-programme `testLib` comprendra des assertions testant en particulier la taille de la matrice, ainsi que les valeurs de différentes cases (y compris les différentes valeurs de la case modifiée).

4. Écrire une fonction `nbelts` renvoyant le nombre d'éléments d'une matrice.
5. Utiliser le prédicat `valid_index` de la bibliothèque des matrices pour définir le prédicat `ttes_cases_sauf_une (m : mint)` qui est vrai si et seulement si une matrice d'entiers est composée de cases dont la valeur est 2, sauf pour une seule case de la matrice qui vaut 1.
6. Tester ce prédicat sur les matrices 3x4 suivantes :

- une matrice ne contenant que des cases dont la valeur est 1,
- une matrice ne contenant qu'une seule case dont la valeur est 1, et telle que toutes ses autres cases valent 2,
- une matrice ne contenant ni la valeur 1, ni la valeur 2.

Exercice 2 : matrice de couleurs

1. Compléter le module `TP32` afin de définir un type énuméré `couleur`, dont les seules valeurs possibles sont rouge, vert et bleu.
2. Définir le type `mcouleur` représentant des matrices de couleurs.
3. Définir le prédicat `ttes_bleues` qui est vrai si et seulement si toutes les cases de la matrice sont bleues.
4. Dans le module de test `Test32` :

- Écrire un sous-programme de test similaire à celui du module Test31, mais adapté aux matrices de couleurs.
- Tester le prédicat `ttes_bleues` sur trois matrices 4x5 de votre choix.

5. Définir le prédicat `au_moins_une_verte` qui est vrai si et seulement si au moins une case de la matrice est verte.
6. Dans le module `Test32`, écrire un sous-programme de test du prédicat `au_moins_une_verte`.