TD 7 : Tableaux à deux dimensions (tableau 2D)

Exercice 1 (Échauffement)**.**
On considère le tableau d'entiers à deux dimensions suivant :

```
vector<vector<int>> Tab2d = { {0,1,2,3}, {10,11,12,13}, {20,21,22,23} };
```

(1) Quelles sont les valeurs `Tab2d[0][0]`, `Tab2d[0][1]`, `Tab2d[2][3]` ?

(2) Donner le nombre de lignes et de colonnes du tableau `Tab2d`.

Exercice 2 (Déclaration, allocation et initialisation de tableau 2D)**.**
Ecrire un programme qui créé un tableau 2D de L lignes et C colonnes et qui l'initialise par un entier v.

Exercice 3 (Opérations sur tableaux à deux dimensions)**.**
Dans tout cet exercice, `t` est un tableau d'entiers à deux dimensions. Pour chaque opération ci-dessous, spécifier et implanter une fonction qui prend `t` en paramètre et la réalise :

(1) renvoyer le nombre de lignes de `t`.

(2) renvoyer le nombre de colonnes de `t` (supposé rectangulaire).

(3) afficher tous les éléments de la ligne numéro l de `t`.

(4) afficher tous les éléments de la colonne numéro c de `t`.

(5) afficher tous les éléments de la diagonale de `t` (supposé carré).

(6) afficher tous les éléments de `t` ; pour l'exemple de l'exercice 1, l'affichage sera :

```
 0  1  2  3
10 11 12 13
20 21 22 23
```

(7) tester si un tableau à deux dimensions contient un élément `x`.

Exercice 4 (Matrices)**.**
Pour être plus spécifique et éviter d'avoir à écrire `vector<vector<int>>` à tout bout de champ, on peut définir un raccourci. Dans les questions suivantes, qui traitent de matrices, on utilisera par exemple le raccourci suivant :

```
typedef vector<vector<int>> Matrice;
```

Pour un tableau 3x3, les deux constructions suivantes sont alors totalement équivalentes :

```
vector<vector<int>> tab = { {1,2,3}, {4,5,6}, {7,8,9} };
```

```
Matrice tab = { {1,2,3}, {4,5,6}, {7,8,9} };
```

Spécifier et implanter une fonction pour chacune des questions suivantes :

(1) teste si un tableau $n \times n$ est symétrique, *i.e.* si $T_{i,j} = T_{j,i}$ pour tous i et j.

(2) calcule la somme de deux matrices (supposées de même type (n, p)). On vous rappelle que la somme de deux matrices T et T' est une matrice C, où $C_{i,j} = T_{i,j} + T'_{i,j}$ pour tous i et j.

(3) ♣ calcule le produit de deux matrices. On vous rappelle que le produit de deux matrices T et T' est une matrice C, où $C_{i,j} = T_{i,1}T'_{1,j} + T_{i,2}T'_{2,j} + \cdots$.

Exercice 5 (Réservation de salle)**.**
Une salle de réunion peut être utilisée par différents employés d'une entreprise. La réservation se fait par plage d'une heure, de 8H00 du matin à 19H00. Chaque plage d'une heure commence à l'heure pile (par exemple, il y a une plage 9H00-10H00 mais il n'y a pas de plage 9H15-10H15. Un tableau de booléens à deux dimensions est utilisé pour représenter si la salle est occupée (valeur `true`) ou disponible (valeur `false`) pendant une semaine. Une dimension est utilisée pour coder les jours ouvrables de 0 (lundi) à 4 (vendredi). L'autre dimension est utilisée pour les plages horaires de 0 (8H00-9H00) à 10 (18H00-19H00). Chaque case correspond à la réservation de la salle pour une plage d'un jour donné.

(1) Écrire le code pour déclarer le tableau repésentant l'état d'occupation d'une salle sur une semaine.

(2) Écrire une fonction qui prend en paramètre le tableau d'état d'une salle et qui l'affiche de façon intelligible (par exemple : salle occupée le mardi de 9H00 à 10H00). Pour cela, on suppose avoir le tableau suivant :

```
vector<string> jours = {"lundi", "mardi", "mercredi", "jeudi", "vendredi"};
```

(3) Écrire une fonction qui calcule le taux d'occupation d'une salle, c'est à dire le nombre de plages réservées divisé par le nombre total de plages.

Exercice ♣ 6 (Le jeu du démineur)**.**
L'objectif de cet exercice est de réaliser une version simple du jeu du « démineur ». Le but est de localiser des mines cachées dans un champ virtuel avec pour seule indication le nombre de mines dans les zones adjacentes.

Plus précisément, le champ consiste en une grille rectangulaire dont chaque case contient ou non une mine. Au départ, le contenu de chaque case est masqué. À chaque étape, l'utilisateur peut :

— Démasquer le contenu d'une case ; s'il y a une mine, "BOUM !", il a perdu. Sinon, le nombre de cases adjacentes (y compris en diagonale) contenant une mine est affiché.
— Marquer une case, s'il pense qu'elle contient une mine.

L'utilisateur a gagné lorsqu'il a démasqué toutes les cases ne contenant pas de mine.

Pour représenter en mémoire l'état interne de la grille, on utilisera un tableau à deux dimensions de caractères (type `vector<vector<char>>`). On utilisera les conventions suivantes pour représenter l'état d'une case :

— 'm' : présence d'une mine, 'M' : présence d'une mine, case marquée ;
— 'o' : absence de mine, 'O' : absence de mine, case marquée ;
— ' ' : absence de mine, case démasquée.

Afin d'éviter d'avoir à écrire `vector<vector<char>>` à tout bout de champ on utilise un raccourci.

```
typedef vector<vector<char>> GrilleDemineur ; // tableau 2D de caractères
```

(1) Implanter une fonction permettant de compter le nombre total de mines (marquées ou pas) dans une grille.

(2) Implanter une fonction permettant de tirer au hasard une grille initiale. On supposera fournie une fonction `bool boolAleatoire()` renvoyant un booléen tiré au hasard.

(3) Implanter une fonction permettant de tester si une grille est gagnante.

(4) Implanter une fonction permettant de compter le nombre de mines dans les cases adjacentes à une case donnée d'une grille.

(5) Implanter une fonction permettant de renvoyer une chaîne de caractères représentant la grille telle que doit la voir le joueur.