TD 7 - fonctions, boucles et tableaux à deux dimensions

1. Échauffement

Exercice 1 (Questions de cours)**.**

(1) Quelles sont les deux zones du modèle de mémoire utilisé pour l'exécution d'un programme ? Que contiennent-elles ?

(2) Rappeler ce qu'est un tableau.

(3) Implanter la fonction suivante :

```
/** Teste si un tableau est de taille 9 et ne contient
    que des entiers entre 1 et 9
 * @param t un tableaux d'entiers
 * @return un booleen
 **/
bool verifie(vector<int> t) { //
```

(4) Implanter la fonction suivante :

```
/** Construit un tableau 2D n x n dont les valeurs sont initialisées à v
 * @param n un entier
 * @param v un entier
 * @return le tableau d'entiers
 **/
vector<vector<int>> tableau2DInitialise(int n, int v) { //
```

2. Sudokus

L'objectif de cette séance est de concevoir un programme pour résoudre automatiquement les grilles de sudokus. Ce problème étant assez difficile en général, nous nous concentrerons sur les déductions immédiates que l'on peut effectuer sur une grille.

Commençons par introduire le problème. Une grille de sudoku M est un tableau à deux dimensions 9×9, où chaque case $M_{i,j}$ est comprise entre 0 et 9 (avec 0 dénotant une case vide). Voici une grille de sudoku incomplète et la même grille complétée :

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Exercice 2.

(1) Comment représenter une grille de sudoku en mémoire ?

(2) Spécifier et implanter une fonction `grilleVide` qui construit une grille de sudoku vide et la renvoie.

Pour éviter d'avoir à écrire `vector<vector<int>>` à tout bout de champ, on définit un raccourci `Grille`. En C++, cela se fait avec :

```
typedef vector<vector<int>> Grille;
```

Il devient alors totalement équivalent d'écrire, par exemple,

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

ou

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

Exercice 3.
Spécifier et implanter une fonction `verifie` qui prend en argument un tableau d'entiers à deux dimensions et renvoie vrai si et seulement si le tableau a des tailles et des valeurs compatibles avec une grille de sudoku.

Exercice 4.
Afin de visualiser ce que l'on manipule, implanter (et spécifier) une fonction `affiche` qui affiche à l'écran une grille de sudoku.

♣ : rajouter des espacements pour séparer les neufs carrés 3×3 lors de l'affichage.

L'objectif du jeu de sudoku est de compléter la grille de telle sorte que chaque ligne, chaque colonne et chacun des neufs sous-carrés 3×3 disjoints contienne chaque chiffre de 1 à 9 exactement une fois.

Exercice 5.
Écrire une fonction `estPresentDansColonne` qui prend en argument une grille, un indice de colonne ainsi qu'une valeur et renvoie `true` si cette colonne contient la valeur, et `false` sinon. Sur le même modèle, implanter et spécifier des fonctions `estPresentDansLigne` et `estPresentDansCarre`. Cette dernière fonction prend en argument les coordonnées de la *case en haut à gauche* du carré, c'est à dire que le carré correspondra à $\{\text{ligne}, ..., \text{ligne} + 2\} \times \{\text{colonne}, ..., \text{colonne} + 2\}$.

Exercice 6.

(1) Spécifier et implanter une fonction `estRemplie` qui renvoie vrai si la grille est remplie, c'est-à-dire ne contient pas de 0.

(2) Spécifier et implanter une fonction `estSolution` qui renvoie `true` si la grille est une solution, c'est-à-dire si chaque ligne, chaque colonne et chacun des neufs sous-carrés contienne chaque chiffre 1 à 9 exactement une fois.

Quelqu'un propose l'idée suivante : *Étant donnée une grille partielle, regardons s'il existe une case vide n'ayant qu'une seule valeur directement possible, et dans ce cas, remplissons la avec la valeur correspondante.*

Exercice 7.
Appliquer à la main cette idée sur les deux grilles suivantes :

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

		1				8		
	7		3	1			9	
3				4	5			7
	9		7			5		
	4	2		5		1	3	
		3			9		4	
2			5	7				4
	3			9	1		6	
		4				3		

Que concluez vous ?

Exercice 8.
Écrire une fonction `valeursPossibles` qui prend en argument une grille et une case et qui renvoie les valeurs directement possibles stockées dans un tableau. Rappel : en `C++` : si `t` est un `vector<int>` et `u` un `int`, alors `t.push_back(u)` ajoute `u` à la fin du tableau `t`.

Exercice 9.
Écrire une fonction `complete` qui prend en entrée une grille et qui retourne en sortie une grille ne possédant plus aucune case vide n'ayant qu'une seule valeur directement possible.

Exercice ♣ 10 (Pour aller plus loin)**.**
Comme vous avez pu le constater cette fonction ne suffit pas. Pour obtenir un programme complet pour résoudre les sudokus, il faut faire des hypothèses et revenir dessus si elles ne conduisent pas à la solution (sans que l'on puisse prévoir quand). Notez que cette idée seule pourrait résoudre le problème en théorie ; cependant elle serait trop lente en pratique. C'est pourquoi les fonctions des exercices 8 et 9 sont très utiles pour accélérer l'exploration dans la plupart des cas !

Du code effectuant cette exploration sera fourni en TP, mais vous pouvez réfléchir à sa structure, notamment aux deux options possibles : avec une seule grille ou avec des copies des grilles courantes.

Exercice ♣ 11.
Écrire un programme qui prend en argument un fichier où chaque ligne est constituée de 81 entiers consécutifs

```
138570009209400176000910050003807004620394817840000 0...
```

puis résoudre le sudoku associé.

On pourra trouver de telles suite à l'adresse : http ://www.printable-sudoku-puzzles.com/wfiles/
(♣ ♣ ♣) Vous avez sans doute remarqué que certaines grilles, comme celle ci-dessous, sont "difficiles" à résoudre dans le sens où le temps de calcul est très long.

```
000000012
008030000
000000040
120500000
000004700
060000000
507000300
000620000
000100000
```

Proposer des optimisations.
(♣ ♣ ♣) Une grille peut avoir plusieurs solutions, comment les énumérer ?
(♣ ♣ ♣) Comment générer une grille de sudoku ?