Séance 4 - Fonctions - Tableaux

Le but de ce TP est de réaliser un solveur de sudokus faciles. Vous devez créer un seul projet Code : :Blocks pour tout le TP. Dans ce projet, comme pour le précédent TP, trois fichiers devront être créés :
– un fichier `main.c` qui contiendra la fonction main() seulement.
– un fichier `fonctions.h` qui contiendra tous les en-têtes (déclarations) de vos fonctions.
– un fichier `fonctions.c` qui contiendra tous les corps (définitions) de vos fonctions.
NE PAS OUBLIER D'INCLURE :
– dans le fichier `main.c` :
  – `<stdio.h>`
  – `<stdlib.h>`
  – `"fonctions.h"`
– dans le fichier `fonctions.h` :
  – `<stdio.h>`
  – `<stdlib.h>`
  – `<stddef.h>`
  – `<assert.h>`
Afin de réaliser ce solveur, nous allons créer trois structures. Ces structures seront créées dans le main, qui est disponible à l'adresse http ://www.lirmm.fr/∼watrigant/files/cplusplus/main.c
(deux instructions sont commentées car il faudra d'abord écrire toutes les autres fonctions et les tester avant d'appeler le programme principal).

Ces trois structures sont :
– `int grille[9][9];` : cette structure représentera la grille du sudoku, c'est donc un tableau statique d'entiers, de taille 9x9.
  Par convention, les cases non encore remplies comporteront la valeur 0.
  Dans le main donné, ce tableau est initialisée selon la grille suivante :

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

– `int* possibilites[9][9];` cette structure comportera, pour chaque case de la grille, un ensemble (sous forme d'un tableau dynamique) de valeurs qu'il est possible de lui affecter. Par exemple, dans la première case (en haut à gauche), on ne peut pas y mettre les valeurs 2, 3, 5, 7, 8, 9. Donc, `possibilites[0][0]` est un tableau qui contient les valeurs [1, 4, 6]. En résumé, c'est un tableau à trois dimensions, dont les deux premières sont fixées (9). De manière équivalente, c'est un tableau statique à deux dimensions stockant des tableaux (à une dimension) dynamiques. Il faut donc stocker quelque part la taille de ces 9x9=81 tableaux statiques, c'est le but de la structure suivante.
– `int taillesPossibilites[9][9];` ce tableau contient les tailles des tableaux de chaque case du tableau précédent. Par exemple, `taillesPossibilites[0][0]` contient la taille du tableau `possibilites[0][0]`, donc 3.
L'algorithme de résolution du sudoku consiste à choisir une case dont les possibilités sont les plus restreintes possibles, lui affecter une valeur, et mettre à jour les possibilités des autres cases (de la même ligne, colonne, et du même "carré").

Les fonctions à écrire sont décrites à la dernière page.

A FAIRE SUR PAPIER

Exercice 1 Ecrire la fonction `initPossibilites`. Cette fonction comporte deux parties :
- allocation des 9x9 tableaux dynamiques de `possibilite`, et remplissage de ceux-ci ainsi que du tableau `taillesPossibilites`. Au début les possibilités de chaque case est [1, 2, 3, 4, 5, 6, 7, 8, 9] si la case est vide, et [ ] (tableau vide) si la case est déjà remplie.
- ensuite, il faut mettre à jour les possibilités en fonction des chiffres qui sont déjà inscrits. Pour cela, on fera appel aux 3 fonction `MajLigne`, `MajColonne` et `MajCarre`.

Exercice 2 Ecrire les 3 fonctions `MajLigne`, `MajColonne` et `MajCarre`. Ces fonctions feront appel à la fonction `retraitSiBesoin`.

Exercice 3 Ecrire la fonction `retraitSiBesoin`. On fera appel aux fonctions `indice` et`supprime`.

Exercice 4 Ecrire la fonction `getNouvelleCase`.

Exercice 5 Ecrire la fonction `sudokuSolver`. On fera appel aux fonctions `getNouvelleCase`, `MajLigne`, `MajColonne`, `MajCarre` et `gagne`. Celle-ci doit aussi s'occuper de l'affichage, on utilisera donc la fonction `afficheGrille` pour afficher l'éventuel résultat ou bien des messages si la résolution n'a pas aboutie.

A FAIRE SUR MACHINE

Exercice 6 Ecrivez les fonctions restantes `indice`, `supprime`, `afficheGrille` et `gagne` et les tester dans le main afin de s'assurer qu'elles fonctionnent correctement. Il est également conseillé d'écrire une fonction `afficheTab` qui affiche un tableau à une dimension afin de pouvoir débugger facilement.

Exercice 7 Ecrivez enfin les fonctions vues sur papier, et résolvez votre sudoku !
Essayez un sudoku de niveau difficile que notre programme n'arrive pas à résoudre. Pourquoi la résolution n'aboutit-elle pas et que faudrait-il modifier à notre algorithme pour que ça fonctionne ?

• `void initPossibilites(int grille[9][9], int* possibilites[9][9], int taillesPossibilites[9][9])`
Cette fonction initialise les tableaux `possibilites` et `taillesPossibilites`.

• `void MajLigne(int i, int val, int grille[9][9], int* possibilites[9][9], int taillesPossibilites[9][9])`
Cette fonction va parcourir toutes les cases de la ligne `i`, et retirer la valeur `val` des possibilités de chaque case si elle y figure.

• `void MajColonne(int j, int val, int grille[9][9], int* possibilites[9][9], int taillesPossibilites[9][9])`
Idem que la précédente, mais pour la colonne `j`.

• `void MajCarre(int i, int j, int val, int grille[9][9], int* possibilites[9][9], int taillesPossibilites[9][9])`
Idem que les deux précédentes, mais elle parcourt cette fois-ci toutes les cases du carré contenant la case (i, j).

• `void retraitSiBesoin(int i, int j, int val, int grille[9][9], int* possibilites[9][9], int taillesPossibilites[9][9])`
Cette fonction retire la valeur `val` des possibilités de la case (i, j) si celle-ci y figure.

• `int indice(int val, int* tab, int taille)`
Cette fonction retourne l'indice où se trouve la valeur `val` dans le tableau `tab` si cette valeur est présente, et retourne -1 sinon.

• `int* supprime(int id, int* tab, int taille)`
Cette fonction retourne un tableau contenant les mêmes valeurs que `tab` sauf celle figurant à l'indice `id` (qui disparait). Indication : utilisez `realloc`.

• `int* getNouvelleCase(int grille[9][9], int* possibilites[9][9], int taillesPossibilites[9][9])`
Cette fonction retourne la case avec le moins de possibilités. Elle retourne en fait un tableau de 2 entiers, contenant la ligne et la colonne de cette case. Dans le cas où il n'y a plus de case vide, ou bien dans le cas où il y a une case avec aucune possibilité, cette fonction retourne un tableau avec comme valeurs -1 et -1.

• `void afficheGrille(int grille[9][9])`
Cette fonction affiche la grille.

• `int gagne(int grille[9][9])`
Cette fonction retourne 1 si la grille est remplie, et 0 sinon.

• `void sudokuSolver(int grille[9][9], int* possibilites[9][9], int taillesPossibilites[9][9])`
Cette fonction contient l'algorithme général.