Le jeu du sudoku: une grille de sudoku est un carré 9x9 qu'on peut décomposer en 9 carrés 3x3 appelés *régions*. Initialement la grille est partiellement remplie de chiffres de 1 à 9. Le but est de remplir chaque case vide par un chiffre de 1 à 9 de façon à ce qu'aucune ligne, colonne ou région contienne deux fois le même chiffre. Idéalement il y a une seule façon de compléter une grille, mais ce n'est pas forcément le cas de toutes les grilles qu'on peut trouver sur le web ou dans les magazines. Dans la grille ci-dessous, la case d'indice (ligne, colonne) = (9, 7) doit forcément contenir 1 pour que la région en bas à droite ait une solution et on en déduit progressivement les valeurs de toutes les cases de cette région :

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

Le but est d'automatiser partiellement la recherche d'une solution à une grille de sudoku en programmant quelques stratégies simples de remplissage basées sur des choix forcés pour certaines cases. En combinant ces stratégies et en itérant le procédé on peut plus ou moins compléter des grilles, selon leur niveau de difficulté. Aller au-delà peut nécessiter d'émettre des hypothèses sur la valeur à placer dans une case, hypothèse qui peut ensuite aboutir à une solution complète ou au contraire à une contradiction amenant à réfuter l'hypothèse initiale et le remplissage de toutes les cases qui en découlait. Pour les grilles les plus complexes il faut enchaîner plusieurs hypothèses avant d'aboutir à une solution ou à un blocage.

Description des types de données : On définit deux types à utiliser : `Grille` et `Indices`. Le second type permet de manipuler globalement un couple d'indices pour désigner une case de la grille, plutôt que via deux entiers séparés. Dans une grille, la valeur `0` représente une case actuellement vide.

```
typedef struct {
   string nom;                          // nom de la grille
   vector<vector<int>> grille;          // la grille elle-même
} Grille;


typedef struct {
  size_t lig;                           // indice de ligne
  size_t col;                           // indice de colonne
} Indices;
```

En interne les indices ont des valeurs adaptées aux opérations sur les vecteurs donc de 0 à 8. Pour l'utilisateur les indices naturels vont de 1 à 9. A chaque fois qu'on interagit avec l'utilisateur, on prendra soin de convertir les valeurs des indices.

La conception du programme est partiellement imposée et demande que vous programmiez un certain nombre de fonctions. Votre réalisation sera testée automatiquement en appelant ces fonctions, il est donc impératif de respecter strictement leur en-tête. Vous pouvez ajouter toute fonction qui vous

semble nécessaire ou programmer des stratégies additionnelles.

Un fichier est mis à disposition avec le squelette des fonctions demandées et quelques fonctions de test. Des fonctions supplémentaires seront mises à disposition selon l'avancée du projet (par exemple pour lire une grille à partir d'un fichier).

Fonctions demandées :

- `bool bienFormee(const Grille &g)` : renvoie `true` si et seulement si la grille dans `g` a la bonne forme et ne contient que des entiers entre 0 et 9. Cette fonction ne sert qu'à valider le format d'une grille sans vérifier les autres contraintes d'une grille de sudoku.
- `bool valide(const Grille &g)` : renvoie `true` si et seulement si toute valeur entre 1 et 9 apparaît au plus une fois dans chaque ligne, colonne ou région. Cette fonction sert à vérifier si la grille initiale est valide mais peut aussi servir à tester un placement de valeur : on place la valeur et on appelle `valide` sur la nouvelle grille : si on obtient `false`, il suffit d'annuler ce placement et essayer autre chose.

Les 3 fonctions suivantes correspondent toutes à une manière de faire progresser la résolution du sudoku si convenablement appliquées aux cases vides de la grille.

- `vector<int> possibles(const Grille &g, Indices ind)`: renvoie un vecteur contenant les valeurs autorisées pour la case d'indices donnés, compte-tenu du contenu actuel de la grille `g`.
- `bool force(Grille &g, Indices ij, vector<int> possibles)` : reçoit en paramètre les indices pour une case ainsi que l'ensemble des valeurs autorisées pour cette case. Vérifie si pour l'une de ces valeurs, cette case est la seule de sa région dans `g` pour laquelle cette valeur convient. Dans ce cas modifie `g` et renvoie `true`. Si pour chaque valeur il existe plusieurs possibilités de la placer dans la région, laisse `g` inchangée et renvoie `false`. Pour cette fonction il peut être utile de calculer l'ensemble des indices des cases vides de la région considérée. Cette stratégie est plus coûteuse mais si elle permet de remplir au moins une case, on peut relancer des stratégies plus simples.
- `bool userSuggest(Grille &g)`: demande à l'utilisateur de fournir une valeur pour l'une des cases vides de `g` dont il précise les indices. La fonction vérifie que la case est vide et que la valeur fournie est possible. Dans ce cas, modifie `g` et renvoie `true`. Si l'utilisateur ne souhaite pas suggérer une valeur, laisse `g` inchangée et renvoie `false`. Normalement `userSuggest` est appelée uniquement si les autres stratégies ne permettent pas d'avancer.
- `bool joue(Grille & g)` : recherche une solution pour `g` en itérant l'application des stratégies définies jusqu'à obtention d'une solution ou blocage. La fonction principale de votre application. Retourne `true` ou `false` selon que la grille a été résolue ou non.

Bonus : les fonctions suivantes permettent d'étendre les possibilités de votre système :

- (*complexité modérée*) Implémentez un système pour sauvegarder et rejouer une partie. En début de partie on choisit de sauvegarder ou non les choix (valeur et indices de la case) de l'utilisateur lors de la résolution (c'est la seule chose à sauvegarder, en plus du nom de la grille associée, puisque le reste se déroule automatiquement). En fin de partie, ces choix sont enregistrés dans un fichier. Avant de démarrer une résolution, on demande si on veut restaurer une partie auquel cas à chaque fois que le système a besoin d'une valeur, on commence par utiliser dans l'ordre les valeurs sauvegardées. Lorsque toutes les valeurs sauvegardées ont été utilisées, les valeurs suivantes sont demandées à l'utilisateur. En supprimant manuellement plus ou moins de valeurs du fichier, on peut « facilement » rejouer partiellement une partie avant de reprendre le contrôle.
- (*plus compliqué)* : Implémentez un système de gestion automatique des hypothèses de manière à obtenir un système entièrement automatique. A chaque fois qu'une hypothèse est nécessaire, une valeur parmi celles possibles est sélectionnée et ce choix est mémorisé avec la case

associée. En cas de blocage ultérieur, on revient au dernier choix effectué pour voir s'il existait une autre possibilité pour la même case. En ce cas on l'essaye à son tour, sinon on propage le blocage jusqu'au précédent point de choix. Ceci suppose de mémoriser toutes les hypothèses déjà faites pour une case dans un certain contexte, ainsi que les points de choix dans une pile pour pouvoir facilement remonter en arrière dans la chaîne des hypothèses.

Modalités d'évaluation : L'évaluation tiendra compte de la qualité de votre réalisation et d'une brève démonstration dont les modalités seront précisées ultérieurement. **Les documents à fournir sont** :

- une **version électronique** de vos fichiers sources, à envoyer par mail à votre chargé(e) de TD qui en accusera explicitement réception. En l'absence d'accusé de réception, il sera considéré que vous n'avez pas rendu le projet;
- un **synopsis** d'une page décrivant votre démonstration, **à fournir au début de la démonstration** (ce qui suppose que vous ayez réfléchi à l'avance à ce que vous souhaitez présenter et que vous ayez **organisé et minuté son déroulement)**.
- Une page décrira **la contribution de chaque membre du binôme**.

Le projet est à faire en binôme mais la note pourra être individualisée au vu de la participation de chacun et de la démonstration (chacun devra être présent et devra pouvoir répondre aux questions). ***L'équipe pédagogique pourra utiliser des logiciels de calcul de similarités entre programmes pour mettre en évidence d'éventuels problèmes de plagiat.***