Une grille de Sudoku est formée par 9 carrés de 3x3 cases dont chacune est soit vide, soit contient un chiffre de 1 à 9. Une grille est **valide** si dans chaque sous-carré, ligne globale et colonne globale, un chiffre apparaît au plus une fois. Une grille **résolue** est une grille valide sans case vide.

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

On représente une grille de Sudoku par le type C++ ci-dessous qui ne fait pas apparaître le découpage en carrés 3x3. On utilise le type *int* comme valeur de case et on met la valeur 0 dans une case pour représenter qu'elle est vide.

typedef vector<vector<int>> Grille;

Une grille de Sudoku est alors une matrice 9x9 représentée par un vecteur de 9 « lignes », chaque ligne devant être un vecteur de 9 valeurs de type *int*. Si g est de type *Grille*, on rappelle que l'accès à la case *g(i,j)* s'écrit *g.at(i).at(j)* où *i* est l'indice de ligne et *j* l'indice de colonne.

On définit un second type pour manipuler globalement un couple d'indices qui désigne une case :

typedef struct { size_t lig; size_t col; } Indices;

Attention : pour l'utilisateur, les indices des lignes et des colonnes vont de 1 à 9, tandis que dans les vecteurs C++ les indices démarrent à 0. Par exemple, pour l'utilisateur la case au milieu de la grille correspond au couple d'indices (5, 5) mais en interne elle est associée aux indices 4 et 4 des vecteurs qui représentent la grille. On convertira en entrée et en sortie pour l'utilisateur les indices qu'on manipule en interne. Dans les fonctions ci-dessous, les indices considérés sont des **indices de vecteurs**.

La fonction *bool bienFormee(const Grille &g)* renvoie *true* si et seulement si *g* représente une grille de Sudoku : les vecteurs de g ont la bonne taille et les cases contiennent des valeurs entre 0 et 9.

Les vérifications étant identiques pour les lignes, les colonnes et les carrés 3x3, on va définir une unique fonction qui effectue les vérifications sur un vecteur de taille 9 dans lequel on

aura stocké les valeurs de l'élément de grille considéré (ligne, colonne, sous-carré). On prévoit quelques fonctions utiles pour nos futures étapes de résolution automatique de grilles :

La fonction *Indices supGauche(Indices ij)* prend en argument le couple d'indices d'une case et renvoie le couple d'indices du coin supérieur gauche du sous-carré contenant cette case. Par exemple, le couple d'indices *{ 4, 4 }* de la case (vide) au centre de la grille, la fonction doit renvoyer *{ 3, 3 }*.

Les fonctions *void Ligne2V(const Grille &g, size_t i, vector &v)* et *void Colonne2V(const Grille &g, size_t j, vector &v)* stockent dans *v* les valeurs de *g* (grille supposée bien formée) pour la ligne d'indice *i* (resp. la colonne d'indice *j*).

Void Carre2V(const Grille &g, Indices ij, vector &v) fait de même pour un sous-carré en prenant en paramètre les indices de son coin supérieur gauche.

La fonction *bool vOK(const vector &v)* prend en paramètre un vecteur v supposé de taille 9 et qui renvoie *true* si et seulement si chaque valeur de 1 à 9 apparaît au plus une fois dans *v*.

Bool valide(const Grille &g) vérifie si une grille bien formée est valide.

La fonction *vector<int> possibles(const Grille &g, Indices ij)* prend en paramètre le couple d'indices d'une case libre et renvoie un vecteur contenant les valeurs qu'on peut y placer, compte-tenu du contenu actuel de la grille. Une telle valeur n'apparaît dans aucun des trois vecteurs qu'on peut construire à partir de la ligne, la colonne et du carré 3x3 qui contient cette case. Dans la case du milieu de la grille en exemple, on ne peut placer que la valeur 5. Si aucune valeur ne convient (choix de l'utilisateur aboutissant à une impasse), on renvoie un vecteur de taille 0.

Vector<Indices> casesVides (const Grille &g, Indices ij) est une fonction qui prend les indices d'une case et renvoie un vecteur contenant les couples d'indices de toutes les cases vides du sous-carré qui contient cette case.

Dans la suite, on construit un système de résolution de la grille qui va enchaîner des étapes simples de résolution jusqu'au succès ou un blocage. Dans le cadre de ce projet, on se limite à deux étapes avant de devoir recourir à l'utilisateur (si on veut résoudre des grilles de difficulté autres que « facile » ou « moyenne »).

La première stratégie effectue un passage sur toutes les cases vides et remplit automatiquement celles pour lesquelles il ne reste qu'une valeur possible compte-tenu des cases déjà remplies.

La fonction *int uneValeur(Grille &g)* prend en entrée une grille *g* et remplit toutes les cases pour lesquelles il n'existe qu'une valeur possible. La fonction renvoie le nombre de cases ainsi remplies.

Une deuxième stratégie effectue un passage sur toutes les cases vides et, pour chacune d'entre elles, vérifie si l'une des valeurs possibles pour cette case ne peut être placée dans aucune autre case du sous-tableau. On la place alors forcément dans la case considérée. La fonction renvoie le nombre de cases ainsi remplies.

La fonction *int unePosition(Grille &g)* implémente cette stratégie.

En désespoir de cause, on demande à l'utilisateur la valeur d'une des cases vides : il fournit la valeur et les indices de la case. Le système doit vérifier que la case est bien vide et la suggestion correcte (on ne remplit ainsi qu'une seule case). Une suggestion peut être insuffisante et aboutir ultérieurement à une autre demande de suggestion. Une suggestion peut aussi être inadaptée et aboutir à une impasse (la grille est incomplète mais il n'est plus possible de remplir une case en gardant la grille valide) : lors de la demande de suggestion, l'utilisateur pourra arrêter la résolution.

Bool *userSuggest(Grille &g)* demande une valeur à l'utilisateur (s'il répond 0 on considère qu'il abandonne et on renvoie *false*), puis les indices de la case à remplir. Si la suggestion est correcte, la fonction affecte la valeur à la case et renvoie *true*, sinon elle ne modifie pas la case et recommence la demande.

La fonctionne *void automatique(Grille &g)* enchaîne les étapes précédentes jusqu'à la résolution complète de la grille ou un blocage.