TP C#4 : Les tableaux de la fin du monde ...

1 Introduction

Nous vous souhaitons tous la bienvenue pour ce dernier TP consacré à la fin du monde.
Dans ce TP ...
Bon, on va reprendre, ne stressez pas, le rendu sera ouvert avant que tout le monde ne disparaisse ! =)
Pour ce TP, vous allez découvrir un nouveau type de structure utilisable en C# : les tableaux. Vous allez aussi vous familiariser avec un nouveau type d'assignation de variables : le passage par référence. Sous ce nom barbare se cache une méthode très pratique pour 'renvoyer' à la fin d'une fonction plusieurs variables par exemple. Bien sûr, cela ne se passe pas comme ça en réalité mais c'est une bonne façon d'illustrer cette technique. Pour ce faire, vous allez devoir faire un petit exercice d'échauffement pour ensuite réaliser un Sudoku et pourquoi pas quelques bonus de votre cru. Avec ce jeu, vous aurez compris que vous allez manger plein de tableaux. :D

2 Comment te dire qu'il va falloir apprendre !

2.1 Tâblo : Kézako ?

En C# , il existe comme dans bien d'autres langages une façon de stocker plusieurs éléments à la fois dans une seule variable : le tableau. Aussi appelé *array* (in English please !),

> *« C'est une structure de données qui contient un certain nombre de variables du même type »*

Comme nous le décrit si gentiment le site de Microsoft : MSDN [1]. Vous allez voir que ces tableaux sont pratiques, ils permettent en effet, par exemple, de représenter une liste statique. Prenons un tableau d'entier, voici comment le déclarer :

```
int[] monTableau = new int[4];
```

Ici, on va déclarer un tableau d'entier en rajoutant au type 'int' des crochets pour spécifier la création du tableau. Après le '=', nous devons déclarer sa valeur, on reprend donc le type du tableau en rajoutant le mot-clé 'new' qui va créer une nouvelle occurrence de ce type. Pour finir, on définit directement sa taille ce qui est obligatoire, en effet ce n'est pas un type dynamique.
Si vous voulez déclarer un tableau en indiquant directement ses valeurs, il n'est pas nécessaire d'indiquer la taille :

```
int[] monTableau = {42, 21, 0, 101010};
```

Ici la taille vaudra ... 4 ! Well done !
On peut faire encore mieux. Si jamais, et c'est aussi l'objet de ce TP, vous aviez envie de créer une matrice pour des calculs mathématiques complexes (Bon je sais on peut toujours rêver ...) ? On va utiliser la même technique mais pour créer un tableau avec une seconde dimension. Cependant, deux choix s'offrent à nous.

```
int[,] monTableau2D = new int[4,2];
int[][] monDeuxiemeTableau2D = new int[4][2];
```

1. http://msdn.microsoft.com/fr-fr/library/9b9dty7d(v=vs.100).aspx

La différence ici réside, vous l'aurez compris, au niveau du typage. Sur la première ligne, nous définissons un tableau multidimensionnel alors que sur la deuxième nous définissons un 'Jagged Array' aussi surnommé tableau de tableaux. En effet, le premier peut se représenter comme un tableau classique sous Excel par exemple (même si en mémoire les cases allouées sont contiguës), alors que l'autre est en fait un tableau à une dimension dont les cases mémoires sont elles mêmes des tableaux. Tordu n'est-ce pas ?
Alors que faut-il vraiment utiliser, bonne question ...

2.2 Paramètres et référencement

Les paramètres sont, dans une fonction, une moyen de communiquer avec celle-ci. Ils vont permettre de personnaliser la façon dont la fonction va s'exécuter. Par exemple, si votre fonction doit récupérer un nombre pour effectuer un calcul complexe dessus comme son carré, il faut le passer en paramètre. N'utilisez SURTOUT PAS de variables globales pour ce genre de choses ! Voir même, n'en n'utilisez pas du tout. Le passage par référence est une autre facette de l'utilisation des paramètres. Cette technique va vous permettre de modifier le contenu passé en paramètre. En effet, lorsque vous passez un paramètre à une fonction, le compilateur va créer une copie locale de ce paramètre pour la zone mémoire de la fonction. Donc, même si vous modifiez un paramètre à l'intérieur d'une fonction, comme la copie locale n'existe qu'à l'intérieur de celle-ci, lorsque vous sortez la modification a disparue. D'où le passage par référence ! Voici tout de suite un petit exemple :

```
void Incrementation(ref int monEntier, int n)
{
        monEntier += n;
}
```

Remarquez le mot-clé *ref* devant l'entier, il indique que le paramètre peut être modifié à l'intérieur de la fonction tout en ayant un effet à l'extérieur.
Je suis d'accord que cette fonction n'a absolument aucune utilité, elle a un but purement pédagogique. Le fait est que les fonctions utiles seront demandées dans la première partie du TP. ;D

Il faut aussi savoir que le mot 'ref' est aussi nécessaire devant les paramètres à référencer lors de votre appel à ce type de fonction :

```
int nombreDeFou = 39;

Incrementation(ref nombreDeFou, 3);
```

Passons dès à présent à la partie intéressante !

3 Architecture de rendu

Comme d'habitude, vous devez impérativement suivre l'architecture de rendu qui va suivre. Pour le fichier AUTHORS, rappelez-vous qu'il n'a pas d'extension et qu'il est fait comme ceci :

```
* login_x

```

Remarquez le saut de ligne.
Bien entendu, login_x est à remplacer par votre véritable login (à moins que vous ne vous appeliez Xavier Login et non pas Loginard sinon ça fait logina_x ;DDD).

```
rendu-tp-login_x.zip
    | login_x/
        | AUTHORS
        | MEB/
            | MEB.sln
            | MEB/
        | Sudoku/
            | Sudoku.sln
            | Sudoku/
```

Dans votre archive de rendu, on ne le répétera jamais assez, ne mettez aucun fichier binaire. Donc il faut pour cela supprimer les dossiers *bin* et *obj* à l'intérieur de chaque projets.
Pour la partie Sudoku de ce TP, une archive vous est fourni contenant des FIXME à combler pour que votre programme fonctionne.

4 Consignes

Pour ce qui est des consignes de rendu, vous devez absolument les respecter sous peine de malus sur votre note :

- il faut respecter l'architecture de rendu ci-dessus
- le format du fichier AUTHORS doit être correct
- aucun fichier binaire ne doit venir polluer votre tarball
- nous vous conseillons de faire les exercices dans l'ordre du sujet, ils vont du plus simple au plus compliqué
- ne passez pas trop de temps en TP avec vos assistants sur le premier exercice, vous y reviendrez chez vous
- tout transfert de code entre élèves de la même promo est considéré comme de la triche, votre note sera donc revue en conséquence
- votre code doit être correctement indenté
- essayez de respecter le plus possible les conventions C# vues en TP (demandez à vos assistants lesquelles sont-elles)
- Visual Studio et MSDN sont vos amis, utilisez les correctement

- veillez à respecter scrupuleusement les prototypes qui vous sont demandés (nom de fonction, nom de variable, type, ...)

Bon TP !

5 Partie I : MEB : Mise En Bouche

Dans cette partie, nous allons apprendre à maîtriser les principes de bases énoncés précédemment. J'espère que vous avez compris les notions du début car comme le dit le titre, vous allez en bouffer des tableaux !

5.1 J'aime les nombres !

5.1.1 Swap

Pour notre première fonction, il va falloir implémenter la fonction swap suivante :

```
public static void Swap(ref int a, ref int b);
```

Cette fonction va échanger le contenu des deux entiers passés en paramètres.

5.1.2 EvenSquare

Maintenant on va coder la fonction SquareEven (ou CarréPair pour les anglophobes mais c'est très moche) :

```
public static bool SquareEven(ref int nb);
```

Cette fonction va mettre au carré le nombre en paramètre uniquement si celui-ci est pair. Attention tout de même, cette fonction retourne la parité du nombre.

5.1.3 TimeAfterTime

Plus dur, c'est au tour de la fonction TimeAfterTime :

```
public static string TimeAfterTime(ref int days, ref int hours,
                                   ref int mins, ref sec nb);
```

Cette fonction va corriger le temps séparé en paramètres pour qu'il corresponde aux valeurs classiques. Par exemple si vous passez 96 secondes en paramètre, vous obtiendrez 1 minute et 36 secondes. Facile non ?
Seul bémol, à quoi sert la valeur de retour me direz-vous, et bien vous allez devoir gérer certaines conditions. En effet, nous voulons que cette fonction soit utilisée par un programme ne supportant pas le nombre 42 (oui je sais, ce programme est stupide), il faudra donc que les paramètres modifiés à la fin de la fonction ne soit pas égaux au nombre 42. Cette règle n'est donc valable que pour les secondes, les minutes et les jours. Dernière chose, notre programme ne peut pas prendre un montant total de secondes excédant 42661337 secondes.
Pour ce qui est de la valeur de retour, nous souhaitons que vous retourniez une chaîne de caractères contenant :

- "SUCCESS" : si tout c'est bien passé
- "ERROR : invalid amount" si le montant total excède celui ci-dessus
- "ERROR : [#42#] %s" si un des résultats finaux est égal à 42, avec %s le nom du paramètre en question. Si plusieurs paramètres sont dans cette situation, le programme renverra "Segmentation fault"

5.2 World of Tank...

5.2.1 CountMeMaybe

On va ici juste s'attaquer aux caractères, pour ceux qui n'auraient pas compris, avec la fonction CountMeMaybe :

```
public static int CountMeMaybe(ref string str, string sample);
```

Cette fonction va supprimer les caractères communs à str et à sample. Il faudra aussi les compter et le renvoyer.

6 Partie II : Sudoku

Pour cette seconde partie, il est temps de passer aux choses sérieuses. Votre but sera de concevoir un *"Sudoku Solver"*. Pour ceux qui ne connaîtrait pas ce jeu très utilisé en tant que passe temps dans les transports, je vous invite à regarder le Wikipédia[2] correspondant.

2. C'est un tort : http ://fr.wikipedia.org/wiki/Sudoku

6.1 Instructions - Archive fournie

Pour cette partie du TP, une archive correspondant à la structure de ce TP vous est donnée. Comme nous voulons vous faciliter la vie un maximum, nous vous avons codé un fichier contenant des fonctions d'entrée/sortie et placé des fichiers Sudoku.cs et Solver.cs dans lequel vous devrez coder les fonctions demandées dans les parties respectivement I et II. Aucune fonction *Main* ne doit être présente lors du rendu, vous pouvez en utiliser une pour tester mais un *Main* de test qui nous est propre sera utilisé pour la correction.
Grâce au fichier IO.cs fourni, vous allez pouvoir récupérer un fichier texte contenant une grille de sudoku, la passer à votre programme, et écrire le résultat dans un nouveau fichier texte (ou le même).

6.2 Mise en place des prérequis

Commençons donc la réalisation de notre 'Sudoku Solver'.
Nous avons essayé de découper au maximum les fonctions nécessaires pour que vous puissiez vous organiser au niveau de votre code. Suivez bien les instructions car cette partie est longue.
Sachez que dans cette partie nous utiliserons uniquement des tableaux multidimensionnels à deux dimensions et des listes.

6.2.1 InitTab

On commence en douceur, vous allez devoir programmer cette fonction :

```
public static void InitTab(ref int[,] tab);
```

Cette fonction initialise tous les éléments du tableau à 0. Attention, elle ne renvoie rien du tout !

6.2.2 PrintTab

Maintenant, pour que vous puissiez dès à présent visualisez correctement les tableaux que vous manipulez, vous devrez coder celle-ci :

```
public static void PrintTab(int[,] tab);
```

Cette fonction imprime donc dans la console le contenu du tableau passé en paramètre. Une seule condition à respecter, le format choisi :

```
+-----------------------+
| 4 1 2 | 9 3 5 | 8 7 6 |
| 9 3 8 | 6 7 2 | 5 4 1 |
| 7 5 6 | 8 4 1 | 3 9 2 |
|-------+-------+-------|
| 8 4 9 | 5 6 3 | 1 2 7 |
| 3 2 5 | 4 1 7 | 6 8 9 |
| 6 7 1 | 2 9 8 | 4 3 5 |
|-------+-------+-------|
| 2 6 3 | 7 5 4 | 9 1 8 |
| 1 9 7 | 3 8 6 | 2 5 4 |
| 5 8 4 | 1 2 9 | 7 6 3 |
+-----------------------+
```

6.2.3 RandomlyFillTab

Pour cette fonction, on va vous aidez un petit peu, voici son prototype :

```
public static int RandomlyFillTab(ref int[,] tab, int nb);
```

Vous allez devoir la coder pour que son tableau passé en paramètre soit rempli de *nb* cases aléatoires supplémentaires. Les cases ont des valeurs aléatoires mais respectent aussi les règles du Sudoku. Notre programme sera donc une boucle exécutée *nb* fois avec à chaque fois pour la case aléatoirement sélectionnée :

- on vérifie si elle n'est pas déjà remplie
- on vérifie la colonne de celle-ci pour qu'il n'y ait pas le même chiffre
- on vérifie la ligne de celle-ci pour qu'il n'y ait pas le même chiffre
- on vérifie la région (carré) de celle-ci pour qu'il n'y ait pas le même chiffre
- si toutes ces conditions sont remplies on peut la remplir avec le chiffre aléatoire sélectionné et on décrémente *nb* sinon on refait la boucle sans décrémenter *nb*

Hint

N'utilisez pas cet algorithme avec de grosses valeurs, restez sur des $nb < 60$
Pour avoir un nombre aléatoire, il faut déclarer un objet Random au début de votre fichier et ensuite l'utiliser comme suit :

```
Random rnd = new Random();
int monEntier = rnd.Next(1, 10);
```

Ici *monEntier* vaudra un nombre aléatoire en 1 (inclus) et 10 (exclus).

6.3 Programmation du Solver

6.3.1 Petit cours sur la classe List<T>

Pour cette partie, il est vivement conseillé d'utiliser la classe List<T> disponible de base en C# . Il faut pour cela ajouter une référence vers cette classe grâce aux directives *using* :

```
using System.Collection.Generic;
```

Vous ne serez autorisés à utiliser que cette classe et uniquement celle-ci. Comme son nom l'indique, c'est une structure de liste déjà implémentée en C# . Pour définir une nouvelle liste, vous devrez faire comme pour une variable normale à l'exception d'une chose : il faut remplacer le T par le type des éléments souhaités dans la liste.

```
List<int> maListe = new List<int>();
// On ajoute le nombre 42 à notre liste nouvellement créée.
maListe.Add(42);
// On recherche le nombre 42 dans la liste.
maListe.Find(42);
// On supprime le nombre 42 dans la liste
maListe.Remove(42);
// Il est aussi pratique de connaître ce moyen pour initialiser une liste.
maListe = new List<int>(new int[] = {0, 42, 21, 101010});
```

Dans le dernier example, on utilise un tableau d'entier pour déclarer les éléments de notre liste. Comme vous pouvez le voir, ce tableau peut être directement déclaré à l'intérieur de notre autre déclaration. Tordu mais pratique non ?
À vous ensuite d'utiliser les fonctions propres à cette classe comme celles d'ajout, de retrait et de comptage d'éléments. Pour plus de renseignements, je vous invite à utiliser MSDN.

6.3.2 GetLinePossibleNumbers

Pour commencer, voici le prototype :

```
static void GetLinePossibleNumbers(ref List<int> numList, int line, int[,] tab);
```

Vous devez écrire une fonction qui va trouver tous les chiffres possibles pour les cases vides d'une ligne *line* de *tab* parmi les chiffres contenus dans *numList*. On doit donc procéder à la suppression des chiffres présents dans la ligne de la liste en paramètre.

Hint

> La liste en paramètre est au départ remplie de 0 à 9.
> Une petite boucle ?...

6.3.3 GetColumnPossibleNumbers

Cela reste le même principe que pour la fonction précédente mais cette fois-ci pour les colonnes :

```
static void GetColumnPossibleNumbers(ref List<int> numList, int line, int[,] tab);
```

6.3.4 GetRegionPossibleNumbers

Encore une fois le principe reste le même sauf que, avant de chercher parmi les cases de la région, il faut que vous définissiez la région dans laquelle on se trouve :

```
static void GetRegionPossibleNumbers(ref List<int> numList, int line,
                                     int column, int[,] tab);
```

6.3.5 GetMinList

Bon, on va arrêter de faire la même chose, voici le prototype :

```
static void GetMinList(ref List<int> minList, ref int xIndex,
                       ref int yIndex, int[,] tab);
```

Cette fonction va trouver la case vide ayant le moins de chiffres possibles pour la remplir. Elle retourne ensuite par référence les index de x et y de la case de la grille, puis la liste de ses possibilités.

Hint

> Voici le petit algo :
> – Création de la liste et d'un entier minimum avec initialisation adaptée.
> – Double boucle ?...
> – On recherche à chaque fois la plus petite liste.
>
> Pensez à vous servir des fonctions créées précédemment.

6.3.6 IsFinished

Une fonction un peu plus facile ?... :

```
static bool IsFinished(int[,] tab);
```

Cette fonction va parcourir toute la grille et vérifier si le tableau est correctement rempli en respectant les règles du Sudoku. Elle renvoi bien sur ce résultat.

Hint

> Souvenez vous qu'il faut utiliser les fonctions **d'avant** (GetMinList...).
> Pensez aux conditions pour lesquelles la grille est correctement établie.

6.3.7 Solve

Et voici le cœur du TP,... le Solveur !

```
public static bool Solve(ref int[,] tab);
```

Bon vous aurez sans doute compris que cette fonction va résoudre notre grille si elle est correcte.
Comme nous sommes sympathique, voici micro-algorithme :

- Retourner vrai si la grille est finie.
- Sauvegarder l'état actuel de la grille.
- Tant que des cases n'ont qu'une seule possibilité, les remplir avec celle-ci.
- Pour tous les chiffres possibles de la case qui en a le moins : remplir cette case avec l'un d'entre eux, utiliser la récursivité, retourner le résultat si vrai, sinon continuer avec le chiffre suivant et ainsi de suite.
- Tester à nouveau si la grille est finie au cas où on ne passe pas par l'étape précédente. On retourne si c'est vrai sinon on rétabli la grille sauvegardée et on retourne faux.

Essayer d'utiliser cette fonction sur la grille vide pour générer une grille générique. Autre truc à tester, la fonction sur une grille remplie par la fonction RandomlyFillTab (avec comme paramètre quelque chose entre 0 ou 20) : vous aurez le droit à votre belle grille aléatoire ! Sachez que la grille que génère RandomlyFillTab est correcte mais n'a pas forcément de solution.
Si vous avez besoin d'aide sur les fonctions de cet exercice, n'hésitez pas à demander à vos ACDC.

6.4 BONUS

Hop, Hop, Hop ! On ne peut pas vous laissez partir comme ça, vous voulez du challenge, essayez donc d'implémenter une boucle de jeu de Sudoku voir même un Windows Form.
Sinon voici d'autres idées :

- Algorithme itératif pour Solve (ne remplace pas celui demandé !).
- Windows Form pour remplir une grille de Sudoku puis l'enregistrer.
- Générateur de grilles aléatoires et valides (mettre des trous là où il faut en fonction de la difficulté).
- ...

Sachez seulement que la seule limite est votre imagination (et la deadline mais bon ...)

7 Conclusion

Voilà, nous espérons que votre Sudoku est fonctionnel et n'oubliez pas :

In MAYAS You Trust !... uh
In ACDC You Trust !