Cours 6 : Révision Tableaux + tableaux à deux dimensions

Objectifs :.Apprentissage des techniques de manipulation des tableaux à 2 dimensions et consolidations des acquis sur les tableaux à une dimension.

Difficultés : .Aucune difficulté majeure, juste de la pratique. ☺☺

C# comme tout langage de programmation, permet de stocker des variables dans des tableaux. Un tableau est composé d'un nombre déterminé de variables de **même type** (primitif ou objet).

Rappels :

Un tableau est déclaré à partir d'un type de données, par exemple :

```
String [] tableaudechaines // tableaudechaines est donc un tableau contenant des chaînes de caractères.
Int [] tableaudentier // tableaudentier est donc un tableau d'entier.
```

Les crochets permettent de spécifier qu'il s'agit d'un tableau. Toutefois à ce niveau, les tableau sont déclarés, mais non créés. En fait ils n'existent pas concrètement en mémoire, vous venez de définir une entité virtuelle. Pour pouvoir les manipuler il faut donc les construire :

```
tableaudechaines= new String [100] ;
tableaudentier= new int[6252];
```

tableaudechaine est maintenant défini et manipulable, comme un tableau de 100 cellules pouvant contenir une chaîne de caractère.

La taille du tableau est définie au moment de sa création, et ne peut plus être changée par la suite. Si on manque de place dans un tableau, il faut donc obligatoirement en créer un nouveau plus grand.

Une fois crée, le tableau est vide, les cellules sont initialisée à 0 pour les tableaux de valeurs numériques, à false pour les booléens, et à null pour les tableaux d'objets.

Pour résumer :
Type [] nomtableau ; // Définition du tableau.
nomtableau= new Type [taille] ; // Création du tableau.
nomtableau[numerocellule]=valeurdetype // Affectation d'une valeur.

L'accès aux cellules se fait en spécifiant une nombre entier (de type byte, char, short, ou int) **indexé à partir de 0**.

Exercices complémentaires de révisions

Tp61.cs

```
using System;
class nom {

// Ceci crée deux tableaux unidimensionnels pour des prénoms et des noms de
famille
string[] preNom = { "Pierre", "Paul", "Pascal", "Pepito" };
string[] nomDeFamille = { "Badet", "Baptiste", "Auguy", "Dellacaza" };
// Ceci affiche les paires de noms (prénom + nom de famille) dans l'ordre
void afficheNoms() {

for (int i = 0; i < preNom.Length; ++i)
{
      Console.WriteLine(preNom[i] + " " + nomDeFamille[i]);
}
}
// Ceci crée une instanciation et appelle la méthode d'affichage
static void Main(string[] args) {
nom a = new nom();
a.afficheNoms();
}
}
```

Les tableaux unidimensionnels: déclaration explicite

Pour illustrer la *déclaration explicite* d'un tableau, nous souhaitons créer un tableau de cinq cellules que nous remplirons de chiffres aléatoires.

RandomNumber.cs

```
using System;
class RandomNumber {
// Ceci crée un tableau de 5 cellules
int[] nombre = new int[5];

// Ceci stocke des nombres aléatoires
void stockeNombres() {
Random rnd = new Random();

for (int i = 0; i < nombre.Length; ++i) {
      nombre[i] =rnd.Next(10,50)    ;
}
}

// Ceci affiche les nombres stockés
void afficheNombres() {
for (int i = 0; i < nombre.Length; ++i) {
Console.WriteLine(nombre[i] + " ");
}
}

// Ceci crée une instanciation et appelle les 2 méthodes
static void Main(string[] args)  {
RandomNumber a = new RandomNumber();
a.stockeNombres();
a.afficheNombres();
}
}
```

Les tableaux ou matrices multidimensionnelles

Les tableaux ou matrices multidimensionnelles ressemblent fortement aux tableaux unidimensionnels. Les dimensions supplémentaires sont indiquées par des paires de crochets supplémentaires.
Ainsi float note[][] = new float[3][4];
déclare une matrice bidimensionnelle contenant 3 éléments sur la première dimension et 4 éléments sur la deuxième. La tradition informatique veut que la dimension variant la plus fortement soit spécifiée à la fin. Ceci permettrait d'écrire une boucle double-enchâssée comme suit:

```
for (int i = 0; i < note.length; ++i) {
        for (j=0; j < note[0].length; ++j) {
                note[i][j] = x;
        }
}
```

Dans cette double boucle, l'élément de la deuxième dimension varie plus rapidement. L'exemple suivant illustrera comment ceci peut être utilisé dans un contexte de traitement répétitif.

Notes.C#

```
using System;
class Notes {
// Ceci crée un tableau bidimensionnel
// pour les notes de 3 groupes de 4 étudiants.
// Si ce tableau était explicite, il serait déclaré note[][] = new float[3][4];
static double [,] note = {
{ 4.75, 5.00, 5.75, 5.25 },
{ 3.00, 5.25, 4.75, 6.00 },
{ 4.25, 5.00, 4.75, 5.52 } };

// Ceci crée un tableau pour les moyennes
double []moyenne = new double[3];
static int groupes, etParGroupe;

// Ceci calcule les moyennes
void calculeMoyennes() {
double sum;

for (int i = 0; i < groupes; ++i) {
sum = 0.0;
for(int j=0;j<etParGroupe;++j){
sum += note[i,j];
}
moyenne[i] = sum/etParGroupe;
}
}

// Ceci affiche les notes stockées et leurs moyennes
void afficheNotesMoy() {
// Affiche titres
for (int i = 0; i < etParGroupe; ++i) {
Console.WriteLine(". " + i);

}
Console.WriteLine("");

// Affiche notes et moyennes
for (int i = 0; i < groupes; ++i) {
Console.WriteLine("Grp " + i + '');
for (int j = 0; j < etParGroupe; ++j) {
Console.WriteLine(note[i,j] + "");
}
Console.WriteLine(moyenne[i]);
}
```

```
}

// Ceci crée une instanciation et appelle les 2 méthodes
static void Main(string[] args) {
Notes f = new Notes();
groupes = note.GetLength(0);
etParGroupe = note.GetLength(1);
f.calculeMoyennes();
f.afficheNotesMoy();
}
}
```

Multmat.C#

```
using System;
class Multmat{
 static void Main(string[] args){

    int lignes=0, colonnes=0;

    /* Saisie de la 1ere matrice */
   Console.WriteLine("Saisie de la 1ere matrice :");

    //On vérifie que le nombre de lignes est plus grand que 0
    do {
      Console.WriteLine("  Nombre de lignes : ");
      lignes=System.Convert.ToInt32(System.Console.ReadLine());
    } while (lignes < 1);

    //On vérifie que le nombre de colonnes est plus grand que 0

    do {
      Console.WriteLine("  Nombre de colonnes : ");
      colonnes=System.Convert.ToInt32(System.Console.ReadLine());
    } while (colonnes < 1);

    /* Déclaration construction de la 1ere matrice */
    double [,] M1= new double[lignes,colonnes];
    for (int i=0; i < lignes; i++)
      for (int j=0; j < colonnes; j++) {
      Console.WriteLine("  M[" + (i+1)  +"," + (j+1) + "]=");
      M1[i,j]=System.Convert.ToInt32(System.Console.ReadLine());;
      }

    /* .. et on refait la même chose pour la 2eme matrice */
    Console.WriteLine("Saisie de la 2eme matrice :");
    do {
      Console.WriteLine("  Nombre de lignes : ");
      lignes=System.Convert.ToInt32(System.Console.ReadLine());
    } while (lignes < 1);

    do {
      Console.WriteLine("  Nombre de colonnes : ");
      colonnes=System.Convert.ToInt32(System.Console.ReadLine());
    } while (colonnes < 1);

    double [,] M2= new double[lignes,colonnes];
    for (int i=0; i < lignes; i++)
      for (int j=0; j < colonnes; j++) {
      Console.WriteLine("  M[" + (i+1)  +"," + (j+1) + "]=");
      M2[i,j]=System.Convert.ToInt32(System.Console.ReadLine());
    }
```

```
  /* Ici on multiplie les matrices */
  if (M1.GetLength(0) != M2.Length)
    Console.WriteLine("Multiplication de matrices impossible !");
  else {
     /* Déclaration-construction de la matrice résultat */
     double [,]prod = new double[M1.Length,M2.GetLength(0)];
    for (int i = 0; i < M1.Length; i++)
      for (int j = 0; j < M2.GetLength(0); j++) {
      prod[i,j] = 0.0;
      for (int k = 0; k < M2.Length; k++)
      prod[i,j] += M1[i,k] * M2[k,j];
      }

    /* Affichage du résultat */
    Console.WriteLine("Résultat :");

    for (int i=0; i < prod.Length; i++) {
      for (int j=0; j < prod.GetLength(i); j++)
      Console.WriteLine(prod[i,j] +  " ");
      Console.WriteLine();
    }
  }
  }
}
```

Exercices applicatifs

Pour les 2 exercices faites en sorte de faire remplir le tableau d'entier par une classe utilisant la fonction math.random(). Utilisez autant de classes que nécessaires (exemple une classe pour le remplissage, une pour le tri, une pour l'affichage…etc…). Ayez de la méthode, faite l'algo et bon courage !

Exercice 1 : Le tri à bulles :

L'algorithme du tri bulle - ou bubble sort - consiste à regarder les différentes valeurs adjacentes d'un tableau, et à les permuter si le premier des deux éléments est supérieur au second. L'algorithme se déroule ainsi : les deux premiers éléments du tableau sont comparés, si le premier élément est supérieur au second, une permutation est effectuée. Ensuite, sont comparés et éventuellement permutés les valeurs 2 et 3, 3et 4 jusque (n-1) et n. Une fois cette étape achevée, il est certain que le dernier élément du tableau est le plus grand. L'algorithme reprend donc pour classer les (n-1) éléments qui précédent. L'algorithme se termine quand il n'y a plus de permutations possibles. Pour classer les n valeurs du tableau, il faut, au pire, effectuer l'algorithme n fois.

Cet algorithme porte le nom de tri bulle car, petit à petit, les plus grands éléments du tableau remontent, par le jeu des permutations, en fin de tableau. Dans un aquarium il en va de même : les plus grosses bulles remontent plus rapidement à la surface que les petites qui restent collés au fond.

Evolution du tableau au fil de l'algorithme (en bleu, les éléments qui sont comparés, et éventuellement permutés, pour passer à la ligne suivante).

5	3	1	2	6	4
3	5	1	2	6	4
3	1	5	2	6	4
3	1	2	5	6	4
3	1	2	5	6	4
3	1	2	5	4	6
1	3	2	5	4	6
1	2	3	5	4	6
1	2	3	5	4	6
1	2	3	4	5	6

L'algorithme se termine car il n'y a plus de permutations possibles. Ce fait sera constaté grâce à un dernier parcours du tableau ou aucune permutation n'a lieu.

Aide : Voici une courte séquence vous montrant l'essentiel.

Bout d'algo

```
tri_bulle(tableau T)
debut
entier longueur, i
booleen inversion

longueur<-taille(T)

faire
        inversion=faux
        pour i=0 à (longueur-1)
                si T(i)>T(i+1)
                        echanger(T,i,i+1)
                        inversion<-vrai
                fin si
        fin pour
tantque inversion=vrai
fin
```

Exercice 2 : le tri par sélection

Le tri par sélection est l'un des tris les plus instinctifs. Le principe est que pour classer n valeurs, il faut rechercher la plus grande valeur et la placer en fin de liste, puis la plus grande valeur dans les valeurs restante et la placer en avant dernière position et ainsi de suite...

Considérons un tableau à n éléments. Pour effectuer le tri par sélection, il faut rechercher dans ce tableau la position du plus grand élément. Le plus grand élément est alors échangé avec le dernier élément du tableau. Ensuite, on réitère l'algorithme sur le tableau constitué par les (n-p) premiers éléments où p est le nombre de fois où l'algorithme a été itéré. L'algorithme se termine quand p=(n-1), c'est à dire quand il n'y a plus qu'une valeur à sélectionner ; celle ci est alors la plus petite valeur du tableau.

Exemple
Grandes étapes de l'évolution du tableau au fil de l'algorithme. En bleu, les valeurs déjà traitées.

5	3	1	2	6	4
5	3	1	2	4	6
4	3	1	2	5	6
2	3	1	4	5	6
2	1	3	4	5	6
1	2	3	4	5	6

Aide : Séquence algorithmique :

Bout d'algo

```
tri_selection(tableau T)
debut
entier longueur, maxi, i
longueur<-taille(T)

tantque(longueur>0) faire
        //recherche de la position du plus grand élément dans le tableau non encore trié
        maxi<-0;

        pour i=1 à (longueur-1) faire
                si T(i)>T(maxi) alors
                        maxi<-i
                fin si
        fin pour

        //echange du plus grand élément avec le dernier
        echanger(T,maxi,longueur-1)

        //traitement du reste du tableau
        longueur<-longueur-1
fin tantque
fin
```