C#

Tableaux et listes

SOMMAIRE

Objectifs

L'objectif de ce chapitre et de créer, initialiser et utiliser des tableaux.

Définition

Imaginons que dans un programme, nous ayons besoin simultanément de 12 valeurs (par exemple, des notes pour calculer une moyenne). Evidemment, la seule solution dont nous disposons à l'heure actuelle consiste à déclarer douze variables, appelées par exemple Notea, Noteb, Notec, etc. Bien sûr, on peut opter pour une notation un peu simplifiée, par exemple N1, N2, N3, etc. Mais cela ne change pas fondamentalement notre problème, car arrivé au calcul, et après une succession de douze instructions de lecture distinctes, cela donnera obligatoirement quelque chose comme :

Moy = (N1 + N2 + N3 + N4 + N5 + N6 + N7 + N8 + N9 + N10 + N11 + N12) / 12

Ce qui est laborieux. Et pour un peu que nous soyons dans un programme de gestion avec quelques centaines ou quelques milliers de valeurs à traiter, cela se révèle fortement problématique.
Si, de plus, on est dans une situation on l'on ne peut pas savoir d'avance combien il y aura de valeurs à traiter, on se retrouve là face à un mur.

C'est pourquoi la programmation nous permet **de rassembler toutes ces variables en une seule**, au sein de laquelle chaque valeur sera désignée par un numéro. En bon français, cela donnerait donc quelque chose du genre « la note numéro 1 », « la note numéro 2 », « la note numéro 8 ».

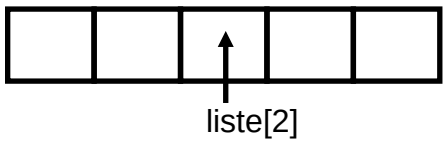
Un tableau est une **suite d'éléments de même type**. On peut accéder à un élément d'un tableau en utilisant sa position : l'index.
En C#, le premier élément se trouve à **l'index 0**.

Déclaration et création

Un tableau est déclaré en spécifiant son type, sa dimension (ou rang) et son nom.

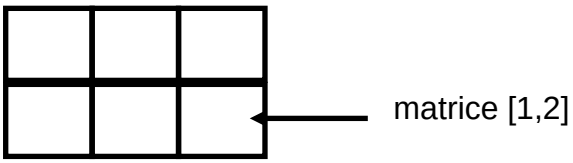
```
type [ ] nom ;

int [ ] liste;              // tableau à une dimension
```



liste[2] désignera le 3eme élément du tableau liste.
Un seul index est nécessaire pour repérer un élément de tableau.

```
string [ , ] matrice ;     // tableau à deux dimensions
```



matrice [1,2] désignera l élément situé sur la 2eme ligne, 3eme colonne du tableau matrice.
Deux index sont nécessaires pour repérer un élément de tableau, l'indice des lignes et l'indice des colonnes

La déclaration d'une variable tableau n'entraîne pas sa création: il faut obligatoirement utiliser **new** pour créer explicitement l'instance du tableau et spécifier la taille de toutes les dimensions.

```
int[] liste;            // déclaration
liste = new int[5];           création d'une instance
```

que l'on peut résumer en :

```
int[] liste = new int[5];// création d'une instance
string [,] matrice= new string [2,3] ;
```

Le compilateur C# initialise implicitement chaque élément du tableau à sa valeur par défaut (en fonction du type).

que l'on peut initialiser comme suit :

```
int[] liste = new int[5]{4,2,3,1,5};
int [,] matrice= new int[2,3] {
                        {1,2,3},
                        {10,20,30}};
```

Chaque élément doit obligatoirement être initialisé.

La taille d'un tableau n'est pas nécessairement une constante; elle peut également être spécifiée par une valeur entière au moment de la compilation.

```
int taille = int.Parse(Console.ReadLine());// lecture
int []liste= new int [taille];               // création
int [,]matrice= new int [2,taille];          // création
```

En C#, lors d'une tentative d'accès au tableau, l'index est automatiquement contrôlé de manière à garantir sa validité.
Un index hors limite – inférieur à 0 ou supérieur ou égal à sa taille- renvoie une exception **IndexOutOfRangeException.**

Utilisation

Soit 2 tableaux ainsi déclarés :

```
int[] liste = new int[5] {4,2,3,1,5};// création d'une instance
string [,] matrice= new string [2,3] {
                                        {1,2,3},
                                    {10,20,30}};
```

Propriétés

La propriété **Length** renvoie le nombre total de postes d'un tableau.

`liste.Length` rend la valeur 5
`matrice.Length` rend la valeur 6 (2 * 3)

La propriété **Rank** renvoie le nombre de dimensions du tableau

`liste.Rank` rend la valeur 1
`matrice.Rank` rend la valeur 2.

Méthodes

La méthode **Sort** permet de trier un tableau.

`System.Array.Sort(liste);` rend dans liste un tableau trié.

La méthode **Clear** permet de réinitialiser un tableau à sa valeur par défaut.

```
// initialise p éléments à partir de la position i
System.Array.Clear(liste, i, p);
```

La méthode **GetLenght** permet de renvoyer la longueur d'une dimension.

```
matrice.GetLength(0) ;   //rend la valeur 2.
matrice.GetLength(1) ;   //rend la valeur 3.
```

La méthode **Clone** crée une nouvelle instance de tableau dont les éléments sont des copies des éléments du tableau cloné.

```
int[] listeCopie = (int [])liste.Clone();
```

alors que l'instruction suivante ne copie pas l'instance du tableau, mais se contente de référencer la même instance de tableau.

```
int[] listeCopie = liste ;
```

Utilisations particulières

Un tableau peut être paramètre de retour de méthode, ou passé en paramètre dans une méthode.

```
namespace Tableaux
{
 class Test1
 {
      static void Main()
      {
       // création nouveau tableau
       int[] nouveauT = CreeTableau(5);
       Console.WriteLine("Nb de postes ={0}",nouveauT.Length);

       // ....
       // Affichage du tableau
       AfficheTableau(nouveauT);
       //
       Console.ReadLine();
      }
      static int[] CreeTableau(int taille)
      {
       return new int[taille];
      }
      static void AfficheTableau(int [] unTableau )
      {
      for (int i = 0; i < unTableau.Length; i++)
      {
       Console.WriteLine("Poste {0}: {1}", i, unTableau[i]);
      }
      }

 }
}
```

Tableau en retour de méthode

Passage d'un tableau en parametre

C'est ainsi que Main peut accepter un tableau de chaînes comme paramètre, réceptionnant les arguments de la ligne de commande.

```
      static void Main(string [] args)
      {
       // ....
      }
```

L'instruction foreach

L'instruction **foreach** simplifie le parcours des éléments d'un tableau.

```
    foreach (int entier in unTableau)
    {
     Console.WriteLine(entier);
    }
```

Les éléments d’index (initialisation, contrôle et incrémentation) sont inutiles.

Opération et tri sur les tableaux

Opérations diverses sur un tableau non trié

Sur une structure de données de type tableau, il est possible de faire plusieurs types de traitement, comme par exemple **la recherche du minimum ou du maximum**, **la somme** ou **le produit ou la moyenne des postes du tableau**.
On peut également vouloir **rechercher un élément donné** dans un tableau.

Par exemple, sur un tableau de 35 postes numériques, on désire calculer la somme des postes, et rechercher le plus petit élément.
Pour le calcul de la somme, on ajoutera le contenu des cases une à une, depuis la première jusqu'à la trente cinquième.
Pour la recherche du minimum :

- On va supposer que la première case contient le minimum relatif
- On va comparer le contenu de la deuxième case avec le minimum relatif : si celui-ci est inférieur, il deviendra le minimum relatif.
- On recommencera l'opération avec les postes restants
-

Tri d'un tableau

Un tableau est ordonné lorsqu'il existe une relation d'ordre entre les différentes cases : On parle de :

- tri croissant si le contenu de la case d'indice i est inférieur ou égal au contenu de la case d'indice i + 1
- tri décroissant si le contenu de la case d'indice i est supérieur ou égal au contenu de la case d'indice i + 1

Plusieurs méthodes de tri existent ; en voici deux exemples sur la base d'un tableau de 30 valeurs numériques VALNUM

Tri croissant par recherche successive des minima

Le principe est le suivant :

- Recherche du minimum dans le tableau de 30 valeurs, et échange du contenu des cases d'indice 1 et d'indice correspondant à la valeur du minimum.
- Application du même principe sur 29 valeurs (30 - première), puis sur 28, puis 27 .... jusqu'au tableau de deux cases.

Visualisation du traitement sur 4 valeurs :

Tableau initial		8	1	7	5
Après le premier passage	1	8	7	5	
Après le deuxième passage		1	5	7	8

Après le troisième passage	1	5	7	8	

Lors du premier passage, on a inversé les cases d'indices 1 et 2.
Lors du deuxième passage, le minimum de (5, 7,8) étant 5, on a inversé les cases d'indices 2 et 4.
Lors du troisième passage, rien ne s'est passé.

Tri à bulle

Le principe est le suivant :

Le tableau est parcouru en comparant les éléments consécutifs.
S'ils sont mal ordonnés, ces deux éléments sont permutés. On recommence jusqu'à ce qu'il n'y ait plus d'échange.

Visualisation du traitement sur 5 valeurs :

Tableau initial	5	18	14	4	26
Premier passage	5	14	18	4	26
	5	14	4	18	26
Deuxième passage	5	4	14	18	26
Troisième passage	4	5	14	18	26
Quatrième passage	4	5	14	18	26

Comme aucune permutation n'a été réalisée, l'algorithme s'arrête.

Méthode :
Les éléments sont comparés deux à deux, et on affecte une variable booléenne à vraie si un échange est réalisé.
La condition d'arrêt du traitement est que la variable booléenne soit restée à faux.

Recherche d'un élément sur un tableau trié

Une première manière de vérifier si un mot se trouve dans le dictionnaire consiste à examiner successivement tous les mots du dictionnaire, du premier au dernier, et à les comparer avec le mot à vérifier.

Ca marche, mais cela risque d'être long : si le mot ne se trouve pas dans le dictionnaire, le programme ne le saura qu'après 40 000 tours de boucle ! Et même si le mot figure dans le dictionnaire, la réponse exigera tout de même en moyenne 20000 tours de boucle. C'est beaucoup, même pour un ordinateur.

Or, il y a une autre manière de chercher, bien plus intelligente pourrait-on dire, et qui met à profit le fait que dans un dictionnaire, les mots sont triés par ordre alphabétique. D'ailleurs, un être humain qui cherche un mot dans le dictionnaire ne lit jamais tous les mots, du premier au dernier : il utilise lui aussi le fait que les mots soient triés.

Pour une machine, quelle est la manière la plus rationnelle de chercher dans un dictionnaire ? C'est de comparer le mot à vérifier avec le mot qui se trouve pile poil au milieu du dictionnaire. Si le mot à vérifier est antérieur dans l'ordre alphabétique, on sait qu'on devra le chercher dorénavant dans le première moitié du dico. Sinon, on sait maintenant qu'on devra le chercher dans la deuxième moitié.
A partir de là, on prend la moitié de dictionnaire qui nous reste, et on recommence : on compare le mot à chercher avec celui qui se trouve au milieu du morceau de dictionnaire restant. On écarte la mauvaise moitié, et on recommence, et ainsi de suite.
A force de couper notre dictionnaire en deux, puis encore en deux, etc. on va finir par se retrouver avec des morceaux qui ne contiennent plus qu'un seul mot. Et si on n'est pas tombé sur le bon mot à un moment ou à un autre, c'est que le mot à vérifier ne fait pas partie du dictionnaire.

Regardons ce que cela donne en terme de nombre d'opérations à effectuer, en choisissant le pire cas : celui où le mot est absent du dictionnaire :

Au départ, on cherche le mot parmi 40 000.
Après le test n°1, on ne le cherche plus que parmi 20 000.
Après le test n°2, on ne le cherche plus que parmi 10 000.
Après le test n°3, on ne le cherche plus que parmi 5 000.
etc.
Après le test n°15, on ne le cherche plus que parmi 2.
Après le test n°16, on ne le cherche plus que parmi 1.

Et là, on sait que le mot n'existe pas. Moralité : on a obtenu notre réponse en 16 opérations contre 40 000 précédemment ! Il n'y a pas photo sur l'écart de performances entre la technique barbare et la technique futée. Attention, toutefois, même si c'est évident**, la recherche dichotomique ne peut s'effectuer que sur des éléments préalablement triés**.

Exercices

Exercice 1 : **Manipulations de base**
Ecrivez un programme permettant de créer un tableau, dont la taille est saisie au clavier.
Ensuite l'utilisateur doit rentrer les différentes valeurs du tableau.
Puis votre programme doit afficher le contenu du tableau.

Exercice 2
Créer le programme C# qui fournira un menu permettant d'obtenir les fonctionnalités suivantes à partir d'un tableau à une dimension:

- Affichage du contenu de tous les postes du tableau,
- Affichage du contenu d'un poste dont l'index est saisi au clavier,
- Affichage du maximum et de la moyenne des postes du tableau

Ce programme sera structuré de la manière suivante :

- une fonction **GetInteger** pour lire un entier au clavier,
- une fonction **InitTab** pour créer et initialiser l'instance de tableau de type entier : le nombre de postes souhaité sera entré au clavier,
- une fonction **SaisieTab** pour permettre la saisie des différents postes du tableau,
- une fonction **AfficheTab** pour afficher tous les postes du tableau,
- une fonction **RechercheTab** pour afficher le contenu d'un poste de tableau dont le rang est saisi au clavier
- une fonction **InfoTab** qui affichera le maximum et la moyenne des postes.

Les fonctions seront appelées successivement dans le Main.

Exercice 3 :
On recherche dans un tableau contenant 20 prénoms, un prénom saisi au clavier.
Lorsque cet élément est trouvé, on l'élimine du tableau en décalant les cases qui le suivent, et en mettant à blanc la dernière case

Exercice 4 : **Tri d'un tableau**
Ecrire le programme C# qui réalise le tri à bulles.

AUTRES TYPES DE DONNEES

Objectifs

- Créer et utiliser des types de données **enum** et **struct** définis par l'utilisateur
- Utiliser les classes de manipulation de date et heure

Le type enum

Les énumérateurs sont utiles lorsqu'une variable ne peut prendre qu'un ensemble spécifique de valeurs. Leur but est de rendre le code source plus compréhensible.

Un type énuméré est un entier défini par le programmeur à l'aide du mot enum ; il s'agit d'une liste de constantes symboliques qui possèdent un identificateur et formant un ensemble de valeurs appartenant à un nouveau type.

Définition d'un type enum :

```
enum EtatCivil {Celibataire, Marie, Divorcé, Veuf} ;
```

Utilisation d'un type enum :

On pourra déclarer une variable *ec* de type `EtatCivil`, en utilisant la syntaxe suivante :

```
EtatCivil ec ;
```

On pourra l'affecter en codant :

```
ec = EtatCivil.Marie ;
```

Ou

```
ec =(EtatCivil)1; // cast d'un type int en EtatCivil
```

On pourra l'afficher en codant :

```
Console.WriteLine("Statut :{0}",ec);
```

Par défaut, le compilateur associe la valeur 0 à célibataire, la valeur 1 à marié, la valeur 2 à divorcé et la valeur 3 à Veuf.

La déclaration peut être placée dans la classe, mais en dehors d'une fonction, ou en dehors de la classe ; le point virgule n'est pas obligatoire.

On pourrait définir des valeurs spécifiques pour chaque valeur de l'énumération :

```
enum EtatCivil
{
 Celibataire=10,
 Marie =20,
 Divorcé= Marie + 1,
 Veuf= 40
}
```

Il est cependant prudent de prévoir une valeur zéro pour la plus commune des valeurs pour des problèmes d'initialisation.

On peut également convertir une chaîne de caractères en l'une des valeurs possibles de l'énumération :

```
string strStatut = "Veuf";
ec = (EtatCivil)Enum.Parse(typeof(EtatCivil),strStatut,false) ;
```

le 3eme argument signale que la casse est ignorée.

Lorsque le 2eme argument ne correspond à aucune des valeurs possibles, une erreur de type **ArgumentException** est générée.

Le type struct

Dans sa forme la plus simple, une structure comprend plusieurs informations regroupées dans une même entité.

Les objets créés à partir de structures se comportent comme des types prédéfinis.

Exemple : Simulation d'un point en géométrie

```
struct Point
{
 public int x;
 public int y;
}
```

La structure contient 2 membres de type `int` spécifiant la position horizontale et verticale d'un point.

La structure peut être définie à l'intérieur ou à l'extérieur d'une classe, mais en dehors d'une fonction; le point virgule n'est pas obligatoire.

Utilisation d'un type struct :

On pourra déclarer une variable *p* de type *`Point`*, en utilisant la syntaxe suivante :

```
Point point ;
```

Les champs de p seront accessibles par *`point.x`* *et* *`point.y`* .et ne sont pas initialisés.

Exemple:

```
static void Main()
{
 Point point;
 point.x = 0; Point.y = 0;
 Console.WriteLine("Coordonnées : {0},{1}",point.x, point.y);
}
```

Dates et heures

Le type valeur **DateTime** représente des dates et des heures dont la valeur est comprise entre 12:00:00 (minuit), le 1er janvier de l'année 0001 de l'ère commune et 11:59:59 (onze heures du soir), le 31 décembre de l'année 9999 après J.C. (ère commune).

Il permet d'effectuer diverses opérations sur les dates : déterminer le jour de la semaine, déterminer si une date est inférieure à une autre, jouter un nombre de jours à une date.

Déclaration d'une date :

```
// crée un objet DateTime
DateTime d1= new DateTime();
//crée un objet DateTime pour le 01/01/2008
DateTime d2 = new DateTime(2008, 01, 01);
// crée un objet DateTime pour le 01/01/2008 12h 0mn 0s 0 ms
DateTime d3 = new DateTime(2008, 01, 01, 12, 0, 0, 0);
```

Utilisation d'un type DateTime :
La structure **DateTime** possède quelques propriétés intéressantes :

- **Date** permet de rendre un nouveau DateTime avec la partie heure initialisée à 0

```
Console.WriteLine(d3.Date);// rend 2008/01/01 à 0h
```

- **Day, Month, Year** permettent de rendre respectivement jour, mois année.
  Hour, Minute, Second, Millisecond permettent de rendre respectivement heures, minutes, secondes et millisecondes.

```
Console.WriteLine("{0},{1},{2}", d2.Day, d2.Month, d2.Year);
// rend 1,1,2008
```

- **DayOfWeek** permet de rendre le jour de la semaine

```
Console.WriteLine("{0}", d2.DayOfWeek); // rend Tuesday
```

- **DayOfYear** permet de rendre le quantième correspondant à la date.

```
Console.WriteLine("{0}", d2.DayOfYear); // rend 1
```

- **Now** donne la date et l'heure du jour

```
// crée un objet DateTime initialisé avec date et heure du jour
DateTime d1 = DateTime.Now;
```

- **Today** donne la date du jour : l'heure est initialisée à 0

```
// crée un objet DateTime initialisé avec date du jour
DateTime d1 = DateTime.Today;
```

Et possède également un certain nombre de méthodes :

- **AddDays, AddMonths, AddYears** permettent d'ajouter respectivement jour, mois, et année.
  AddHours, AddMinutes, AddSeconds, AddMilliseconds permettent d'ajouter respectivement heures, minutes, secondes et millisecondes.

- **Compare, CompareTo** permettent de comparer deux dates

```
int res1 = DateTime.Compare(d2, d3);
int res2 = d2.CompareTo(d3);
```

  et renvoient 0 si d2=d3, 1 si d2> d3, -1 sinon.
  Les opérateurs de comparaison peuvent également être employés.

- **ToString** permet de mettre en forme une date
  En utilisant des formats généraux

```
// Format date longue
```

```
Console.WriteLine(d2.ToString("D")); // mardi 1 janvier 2008
// Format date courte
Console.WriteLine(d2.ToString("d")); // 01/01/2008
// Jour + Mois en clair
Console.WriteLine(d2.ToString("M")); // 1 janvier
```

En utilisant des formats personnalisés, par exemple :

d, M	Jour (0 à 31), mois (01 à 12)
dd ,MM	Jour (01 et 31), mois (01 à 12)
ddd, MMM	Abréviation du jour/ mois
dddd, MMMM	Nom complet du jour / mois
y	Année (de 1 à 99)
yy	Année (de 01 à 99)
yyyy	Année (de 1 à 9999)

```
Console.WriteLine(d2.ToString("d-MMMM-yy")); // 1-janvier-08
```

(Voir l'aide en ligne pour les autres méthodes, et tous les formats).

DateTime et TimeSpan :

Les types valeur **DateTime** et **TimeSpan** se distinguent par le fait que **DateTime** représente un instant, tandis que **TimeSpan** représente un intervalle de temps. Cela signifie, par exemple, que vous pouvez soustraire une instance **DateTime** d'une autre pour obtenir l'intervalle de temps les séparant. De la même façon, vous pouvez ajouter un **TimeSpan** positif au **DateTime** en cours pour calculer une date ultérieure. Vous pouvez ajouter ou soustraire un intervalle de temps d'un objet **DateTime**. Les intervalles de temps peuvent être négatifs ou positifs ; ils peuvent être exprimés en unités telles que les graduations ou les secondes ou comme un objet **TimeSpan**.

Déclaration d'un TimeSpan :

```
// crée un nouveau TimeSpan
TimeSpan ts1 = new TimeSpan();
//crée un nouveau TimeSpan en heure, mn, secondes
TimeSpan ts2 = new TimeSpan(10,20,30);
```

La structure **TimeSpan** possède des propriétés:

- **Days, Minutes, Seconds, Milliseconds** permettent de rendre respectivement le nombre de jour, minutes, secondes et millisecondes de l'intervalle de temps.
-

Et aussi des méthodes :

- **Add, Substract** permettent d'ajouter ou de soustraire un intervalle de temps passé en argument.

Exemple : Calcul du nombre de jours écoulés depuis le début du 21eme siècle

```
DateTime dj = DateTime.Today;// aujourd'hui
DateTime ds = new DateTime(2001, 01, 01); // début du 21eme
siècle
```

```
TimeSpan ts = dj - d;
Console.WriteLine("Nb de jours : {0}", ts.Days);
```

Exercices

Exercice : **Majorité**
A partir de la saisie de sa date de naissance, affichez si le candidat est majeur ou non.
Plusieurs solutions sont possibles …

Les statistiques

On saisit des entiers et on les range dans un tableau (maximum 50)

Ecrire un programme qui affiche le maximum, le minimum et la valeur moyenne de ces nombres.

Conversion de bases

Ecrivez un programme qui convertit un nombre en base 2 vers son équivalent en base 10

Ecrivez le programme qui effectue l'opération inverse.

Comment pouvez-vous modifier votre programme pour effectuer une conversion vers la base 16.

Pour vous entrainer …

Le tri par insertion

Voici l'algorithme du tri par insertion, traduisez le en langage C#

```
array est un tableau <- [666, 1, 7, 69, 33, 13]
j <- 1
n <- longueur de array
TantQue j <> n
    i <- j - 1
    tmp = array[j]
    TantQue i > -1 et array[i] > tmp
        array[i+1] <- array[i]
        i <- i - 1
    FinTantQue
    array[i+1] <- tmp
    j <- j + 1
FinTantQue
```

Le calcul des heures

Le programme réalise l'addition de deux données exprimées en HH:MM:SS et affiche le résultat sous la même forme.

Le crible d'Erathostène

Cet algorithme permet d'afficher progressivement la liste des nombres premiers inférieurs à une valeur donnée : MAX.

Pour ce faire on construit un tableau de MAX éléments, vide au départ, que l'on parcourt.
Chaque fois que la case est vide cela signifie que l'indice du tableau est un nombre premier, on l'affiche puis on remplit avec une valeur quelconque toutes les cases du tableau indicées par un multiple de l'indice courant.

exemple pour MAX = 10

tableau = 0 0 0 0 0 0 0 0 0 0
indice :
1 1 est un nombre premier (je ne marque rien !)
2 2 est un nombre premier ==> je marque
tableau = 0 1 0 1 0 1 0 1 0 1
3 3 est un nombre premier ==> je marque
tableau = 0 1 1 1 0 1 0 1 1 1
4 4 n'est pas un nombre premier
5 5 est un nombre premier ==> je marque etc....

Ecrire un programme qui demande un nombre et affiche tous les nombres premiers inférieurs à un nombre donné par l'utilisateur.