Les tableaux

Tableaux à une dimension

Supposons qu'on veuille enregistrer le résultat d'un match de tennis, c'est-à-dire le score de chaque joueur à chaque set.

Si le match est en 5 sets maxi, il nous faudrait en tout 10 variables pour stocker toutes les infos.

Mais il existe un moyen plus simple de représenter une série de valeurs du même type : les tableaux.

On peut stocker les scores des 2 joueurs pour chaque set dans 2 tableaux de nombres entiers, déclarés comme ceci :

```
byte[] scoresJ1 = new byte[5];
byte[] scoresJ2 = new byte[5];
```

Les crochets après le type indiquent qu'il s'agit d'un tableau. La deuxième partie de l'instruction après le signe égal permet de réserver la place en mémoire pour les éléments du tableau. Ici nous réservons 5 places pour des bytes.

J'utilise le type byte, car le score d'un joueur ne dépassera jamais 255 (valeur maxi d'un byte).

Par défaut, les éléments d'un tableau de nombres sont initialisés à 0.

Pour pouvoir repérer facilement les sets non encore joués, on peut utiliser des valeurs nullables :

```
byte?[] scoresJ1 = new byte?[5];
byte?[] scoresJ2 = new byte?[5];
```

Pour enregistrer les scores du joueur 1, on utilise ensuite la syntaxe suivante :

```
scoresJ1[0] = 6;
scoresJ1[1] = 3;
scoresJ1[2] = 5;
scoresJ1[3] = 6;
scoresJ1[4] = 7;
```

On accède à chaque case du tableau par son indice. Celui-ci commence à 0 et non à 1.

On pourrait aussi enregistrer tous les scores d'un coup dès la création du tableau, avec la syntaxe suivante :

```
byte?[] scoresJ2 = { 2, 6, 7, 3, 5 };
```

Les valeurs sont placées entre accolades et séparées par des virgules.

Le compilateur définit la taille du tableau (c'est-à-dire son nombre de cases), à partir du nombre de valeurs fournies. C'est pourquoi il n'est pas nécessaire de la spécifier explicitement par `new byte?[5]` après le signe égal.

Cette syntaxe fonctionne pour tous les types. Par exemple pour des chaînes :

```
string[] courses = { "chou", "riz", "cacao" };
```

La taille d'un tableau doit toujours être définie (implicitement ou explicitement) au moment de son initialisation, et ne peut plus être modifiée ensuite !

C'est pourquoi les tableaux sont adaptés pour stocker un nombre d'éléments connus à l'avance. Si on ne connaît pas le nombre d'éléments au départ, on utilise plutôt des collections de types listes ou dictionnaires, qu'on verra plus loin dans cette formation…

Les éléments d'un tableau peuvent être modifiés comme bon nous semble, comme des variables classiques. Par exemple, on peut corriger le score du second joueur au premier set comme ceci :

```
ScoresJ2[0] = 3;
```

On peut récupérer la taille d'un tableau grâce à sa propriété `Length`.

```
Console.WriteLine(scoresJ1.Length); // affiche 5
```

ça nous servira dans la prochaine leçon pour parcourir les éléments du tableau…

Tableaux à plusieurs dimensions

Les tableaux que nous avons créés précédemment sont des tableaux à une dimension. C# permet de créer des tableaux à 2, 3… ou autant de dimensions que l'on souhaite.

NB/ On peut se représenter facilement un tableau à 2 dimensions sous forme de grille et un tableau à 3 dimensions sous forme de cube. Les tableaux à 4 dimensions ou plus sont des objets plus abstraits.

On pourrait par exemple stocker les résultats des 2 joueurs dans un unique tableau à 2 dimensions de la façon suivante :

```
byte?[,] scores = new byte?[2,5];
scores[0, 0] = 6;
scores[1, 0] = 2;
scores[0, 1] = 3;
scores[1, 1] = 6;
scores[0, 2] = 5;
scores[1, 2] = 7;
...
```

La première dimension représente les joueurs et la seconde leurs scores.

On peut également initialiser les valeurs du tableau en même temps que sa déclaration, comme ceci :

```
byte?[,] scores = { { 6, 2 }, { 3, 6 }, { 5, 7 }, { 6, 3 },{ 7, 5 } };
```

La bibliothèque de classes du framework .Net utilise des tableaux à une dimension, mais je n'ai jamais vu de méthodes prenant en paramètre ou renvoyant un tableau à plusieurs dimensions. On utilise plutôt des collections d'objets, qui sont plus faciles à manier.

Chaînes de caractères

Une chaîne de caractères renferme en fait un tableau de caractères. C'est pourquoi on peut accéder à un caractère particulier de la chaîne via son indice, comme le montre l'exemple ci-dessous.

```
string phrase = "Le C# est un langage moderne et puissant !";

char car = phrase[4];
Console.WriteLine($"Le 5ème caractère est : {car}");
```

C'est une technique qui peut être utile lorsqu'on veut parcourir une chaîne caractère par caractère pour compter ou récupérer certaines infos.

En revanche une chaîne est immuable, c'est-à-dire qu'on ne peut pas modifier ses caractères après coup.

```
phrase[2] = 'b'; // Interdit
```