

Chapitre 4

Les bases du langage

1. Introduction

Comme tous les langages de programmation, C# impose certaines règles au développeur. Ces règles se manifestent au travers de la syntaxe du langage mais elles couvrent le large spectre fonctionnel proposé par C#. Avant d'explorer en profondeur les fonctionnalités du langage au chapitre suivant, nous étudierons donc les notions essentielles et fondamentales de C# : la création de données utilisables par une application et le traitement de ces données.

2. Les variables

Les données utilisables dans un programme C# sont représentées par des variables. Une variable est un espace mémoire réservé auquel on assigne arbitrairement un nom et dont le contenu est une valeur dont le type est fixé. On peut manipuler ce contenu dans le code en utilisant le nom de la variable.

2.1 Nommage des variables

La spécification du langage C# établit quelques règles à prendre en compte lorsque l'on nomme une variable :

- Le nom d'une variable ne peut comporter que des chiffres, des caractères de l'alphabet latin accentués ou non, le caractère ç ou les caractères spéciaux `_` et `μ`.
- Le nom d'une variable ne peut en aucun cas commencer par un chiffre. Il peut en revanche en comporter un ou plusieurs à toute autre position.
- Le langage est sensible à la casse, c'est-à-dire qu'il fait la distinction entre majuscules et minuscules : les variables `unevariable` et `uneVariable` sont donc différentes.
- La longueur d'un nom de variable est virtuellement illimitée : le compilateur ne remonte pas d'erreur lorsqu'il est composé de 30 000 caractères ! Il n'est évidemment pas recommandé d'avoir des noms de variables aussi longs, le maximum en pratique étant plus souvent de l'ordre de la trentaine de caractères.
- Le nom d'une variable ne peut pas être un mot-clé du langage. Il est toutefois possible de préfixer un mot-clé par un caractère autorisé ou par le caractère `@` pour utiliser un nom de variable similaire.

Les noms de variables suivants sont acceptés par le compilateur C# :

- `maVariable`
- `maVariableNumero1`
- `@void` (`void` est un mot-clé de C#)
- `µn3_µàRiãbl3`

De manière générale, il est préférable d'utiliser des noms de variables explicites, c'est-à-dire permettant de savoir à quoi correspond la valeur stockée dans la variable, comme `nomClient`, `montantAchat` ou `ageDuCapitaine`.

2.2 Type des variables

Une des caractéristiques de C# est la notion de typage statique : chaque variable correspond à un type de données et ne peut en changer. De plus, ce type doit être déterminable au moment de la compilation.

2.2.1 Types valeurs et types références

Les différents types utilisables en C# peuvent être décomposés en deux familles : les types valeurs et les types références. Cette notion peut être déconcertante au premier abord, puisqu'une variable représente justement une donnée et donc une valeur. Ce concept est en fait lié à la manière dont est stockée l'information en mémoire.

Lorsque l'on utilise une variable de type valeur, on accède directement à la zone mémoire stockant la donnée. Au moment de la création d'une variable de type valeur, une zone mémoire de la taille correspondant au type est allouée. Chaque octet de cette zone est automatiquement initialisé avec la valeur binaire 00000000. Notre variable aura donc une suite de 0 pour valeur binaire.

Dans le cas d'une variable de type référence, le comportement est différent. La zone mémoire allouée à notre variable contient une adresse mémoire à laquelle est stockée la donnée. On passe donc par un intermédiaire pour accéder à notre donnée. L'adresse mémoire est initialisée avec la valeur spéciale `null`, qui ne pointe sur rien, tandis que la zone mémoire contenant les données n'est pas initialisée. Elle sera initialisée lorsque la variable sera instanciée. Dans le même temps, l'adresse mémoire stockée dans notre variable sera mise à jour.

Une variable de type référence pourra donc ne contenir aucune donnée, tandis qu'une variable de type valeur aura forcément une valeur correspondant à une suite de 0 binaires.

Cette différence de fonctionnement a une conséquence importante : la copie de variable se fait par valeur ou par référence, ce qui signifie qu'une variable de type valeur sera effectivement copiée, tandis que pour un type référence, c'est l'adresse que contient la variable qui sera copiée, et il sera donc possible d'agir sur la donnée réelle indifféremment à partir de chacune des variables pointant sur ladite donnée.

2.2.2 Types intégrés

La plateforme .NET embarque plusieurs milliers de types différents utilisables par les développeurs. Parmi ces types, nous en avons une quinzaine que l'on peut considérer comme fondamentaux : ce sont les types intégrés (aussi nommés types primitifs). Ce sont les types de base à partir desquels sont construits les autres types de la BCL ainsi que ceux que le développeur crée dans son propre code. Ils permettent de définir des variables contenant des données très simples.

Ces types ont comme particularité d'avoir chacun un alias intégré à C#.

Types numériques

Ces types permettent de définir des variables numériques entières ou décimales. Ils couvrent des plages de valeurs différentes et ont chacun une précision spécifique. Certains types seront donc plus adaptés pour les calculs entiers, d'autres pour les calculs dans lesquels la précision décimale est très importante, comme les calculs financiers.

Les différents types numériques sont énumérés ci-dessous avec leurs alias ainsi que les plages de valeurs qu'ils couvrent.

Type	Alias C#	Plage de valeurs couverte	Taille en mémoire
System.Byte	byte	0 à 255	1 octet
System.SByte	sbyte	-128 à 127	1 octet
System.Int16	short	-32768 à 32767	2 octets
System.UInt16	ushort	0 à 65535	2 octets
System.Int32	int	-2147483648 à 2147483647	4 octets
System.UInt32	uint	0 à 4294967295	4 octets
System.Int64	long	-9223372036854775808 à 9223372036854775807	8 octets
System.UInt64	ulong	0 à 18446744073709551615	8 octets
System.Single	float	±1,5e-45 à ±3,4e38	4 octets

Type	Alias C#	Plage de valeurs couverte	Taille en mémoire
System.Double	double	$\pm 5,0e-324$ à $\pm 1,7e308$	8 octets
System.Decimal	decimal	$\pm 1,0e-28$ à $\pm 7,9e28$	16 octets

Les types numériques primitifs sont tous des types valeurs. Une variable de type numérique et non initialisée par le développeur aura pour valeur par défaut 0.

■ Remarque

.NET 4 a apporté le type `System.Numerics.BigInteger` afin de manipuler des entiers d'une taille arbitraire. Ce type est aussi un type valeur, mais il ne fait pas partie des types intégrés.

Les valeurs numériques utilisées au sein du code peuvent être définies à l'aide de trois bases numériques :

- **Décimale** (base 10) : c'est la base numérique utilisée par défaut.
Exemple : 280514
- **Héxadécimale** (base 16) : elle est très utilisée dans le monde du Web, pour la définition de couleurs, et plus généralement pour encoder des valeurs numériques dans un format plus condensé. **Une valeur héxadécimale est écrite en la préfixant par 0x.**
Exemple : 0x0447C2
- **Binaire** (base 2) : cette base est celle qui permet de s'approcher au plus près de la manière dont la machine interprète le code compilé. L'utilisation du binaire en C# peut par conséquent se révéler précieuse dans le cadre d'optimisations (décalages de bits au lieu de multiplications par 2, stockage de multiples données au sein d'une même variable, etc.). **Les valeurs binaires sont préfixées par 0b.**
Exemple : 0b01000100011111000010

La lecture de valeurs numériques peut s'avérer difficile dans certains cas, notamment lorsqu'elles sont représentées sous forme binaire. L'équipe en charge des évolutions de C# a intégré, dans la septième version du langage, la possibilité d'ajouter des séparateurs dans les valeurs numériques afin d'améliorer la lisibilité globale. Le caractère utilisé pour cela est `_` :

- 280_514
- 0x04_47_C2
- 0b0100_0100_0111_1100_0010

Types textuels

Il existe dans la BCL deux types permettant de manipuler des caractères Unicode et des chaînes de caractères Unicode : `System.Char` et `System.String`. Ces types ont respectivement pour alias `char` et `string`.

Le type `char` est un type valeur encapsulant les mécanismes nécessaires au traitement d'un caractère Unicode. Par conséquent, une variable de type `char` est stockée en mémoire sur deux octets.

Les valeurs de type `char` doivent être encadrées par les caractères `'` : `'a'`.

Certains caractères ayant une signification particulière pour le langage doivent être utilisés avec le caractère d'échappement `\` afin d'être correctement interprétés. D'autres caractères n'ayant pas de représentation graphique doivent être aussi déclarés avec des séquences spécifiques. Le tableau suivant résume les séquences qui peuvent être utilisées.

Séquence d'échappement	Caractère associé
<code>\'</code>	Simple quote <code>'</code>
<code>\"</code>	Double quote <code>"</code>
<code>\\</code>	Backslash <code>\</code>
<code>\a</code>	Alerte sonore
<code>\b</code>	Retour arrière
<code>\f</code>	Saut de page
<code>\n</code>	Saut de ligne