Cours : Programmation orientée objet en C#
Cote du cours : 26978
Session : Hiver 2022

Chapitre 02

Les bases du langage : variables, types et opérateurs

Table des matières

Introduction

Comme tous les langages de programmation, C# impose des règles (Syntaxe) aux développeurs. Dans ce chapitre nous allons étudier les notions essentielles et fondamentales de C#.

I. Les variables

Une variable est une donnée utilisable dans un programme. Elle représente un espace mémoire réservé référencé par un nom et dont le contenu est une valeur appartenant à un type fixé. Pour accéder, modifier ou manipuler le contenu d'une variable, on utilise son nom.

Comme tous les langages de programmation, le nom d'une variable obéi à un ensemble de règles :

- Il est composé de chiffres, des caractères de l'alphabet latin accentués ou non, le caractère **ç** ou les caractères spéciaux **_** et **µ**.
- Il ne peut pas **commencer** par un chiffre.
- C# est sensible à la casse.
- La longueur d'un nom de variable est théoriquement illimitée.
- Le nom d'une variable ne peut pas être un mot-clé du langage. Si vous avez besoin d'utiliser un mot-clé en tant que nom de variable, il faut préfixer le nom de l'identifiant par le caractère **@**.

On recommande l'utilisation de nom de variable significatif dans vos programmes.

Exemple :

Variables correctes	Variables incorrectes
compteBancaire	$x
CompteBancaire	1x
age	2_z
_x1	Compte-bancaire
µy	adresse$
@void	Compte@bancaire
int	age.

II. Les types de données :

Les types de données en C# peuvent être décomposés en deux familles :

- Les types valeurs : la variable stocke la valeur et y accède directement.
- Les types références : la variable stocke une référence vers la valeur et y accède à travers cette référence.

La plateforme .NET contient un bon nombre de type de données utilisable par les développeurs. Parmi ces types, il y a quinzaine qui représente les types primitifs (appelé aussi type intégré). Ils

sont appelés ainsi car ils représentent les types de base à partir desquels sont construits les autres types. Ces types primitifs ont des alias intégrés à C#.

1. Les types numériques (primitives)

Ces types permettent de définir des variables numériques entières ou décimales. Le tableau ci-dessous présente ces types avec leurs alias ainsi que leurs plages de valeurs.

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
System.Double	**double**	±5,0e-324 à ±1,7e308	8 octets
System.Decimal	decimal	±1,0e-28 à ±7,9e28	16 octets

Vous avez la possibilité d'ajouter des séparateurs dans les valeurs numériques afin d'améliorer la lisibilité. À partir de la version 7 de C#, le caractère utilisé pour cela est _ :

Exemple :

Entier en binaire **0b**1010_0110_0011_1100_0101	Représente l'entier : 680901
Entier en hexadécimale **0x**12_AB_F8	Représente l'entier : 1223672
Entier en base 10 (par défaut) 123_452	Représente l'entier : 123452

2. Les types textuels (primitives)

Les types textuels sont de deux types :

- **System.Char** ou **char** : représente les caractères Unicode et c'est un type par valeur stocker sur deux octets. Pour représenter un caractère, on utilise les ' '. le caractère \ est un caractère d'échappement utilisé pour définir des caractères spécifiques ou des séquences spécifiques.

\'	Simple quote	**\t**	Tabulation Horizentale
\"	Double quote	**\v**	Tabulation verticale
\n	Saut de ligne	**\f**	Saut de page
\a	Alerte sonore	**\uXXXX**	Caractère Unicode dont le code hexadécimal est XXXX
\\	Backslash (\)		

➢ **System.String** ou **string** : représente les chaines de caractères Unicode (Tableau de type caractère) et c'est un type par référence. Pour définir les chaines de caractères on utilise les " ". les chaines de caractères sont immutables (invariables) c'est-à-dire qu'elles ne peuvent pas être modifiées et que toutes modification entraine la création d'une nouvelle chaine.

3. Type booléen

Une valeur booléenne (**bool**) représente un état vrai ou faux et accepte uniquement deux valeurs : **true** et **false**. C'est un type très essentiel pour l'élaboration des conditions plus tard.

III. Déclaration des variables

Une variable doit être déclarée avant d'être utilisée. La syntaxe générale de déclaration d'une variable est la suivante :

```
TypeVariable NomVariable [ = valeur initiale] [, secondeVariable];
```

L'opérateur = permet d'affecter une valeur à la variable. Si la valeur initiale de la variable n'est pas définie à la déclaration, la variable aura pour valeur la valeur par défaut de son type.

Type	Valeur par défaut
Type par référence	null
Les entiers et les réelles	0
Les booléens	False
Les caractères	'\0'

Exemple :

```
System.String s = "Bonjour tout le monde";
string s2 = "Merci d'être présent aujourd'hui";
int x = 3, y = 7, z = x + y;
long a = 12L;
double b = 5.5D, c = 3.3d, d = 8.9;     //D ou d pour désigner le type double
float f1 = 4.2F, f2 = 5.3f;             //F ou f pour désigner le type float car par défaut c'est le
                                        //type double
```

IV. L'inférence de type

L'inférence de type est la capacité à déterminer le type d'une variable en fonction de la valeur qui lui est assignée. Le mot-clé var permet de déclarer une variable en inférant son type.

```
var NomVariable = valeurInitiale ;
```

Exemple :

```
var s = "Mohamed";   // s de type string
var x = 3;           // x de type int
var y = 3.3;         // y de type double
```

V. Les constantes

Une constante est une variable dont la valeur ne change pas. Dans le langage C#, le mot clé **const** permet de définir une constante.

Exemple

```
const double A = 5.632;
```

Si on essaie de modifier la valeur de A alors une erreur de compilation sera générée.

VI. Les opérateurs

Les opérateurs sont des mots-clés du langage permettant l'exécution de traitements sur des variables (les opérandes). La combinaison d'un **opérateur** et d'un ou plusieurs **opérandes** constitue une **expression**.

1. Opérateur d'affectation

Un des premiers opérateurs est l'opérateur d'affectation = qui permet d'assigner une valeur à une variable. Celle-ci peut être une variable, une valeur littérale, le résultat d'une fonction ou le résultat d'une expression.

Exemple

```
int x = 3;          //Affectation de la valeur 3 à la variable x.
long a = 12L;       //Affectation de la valeur 12 sous la forme d'un long à la variable a.
double d = 5.5D;    //Affectation de la valeur 5.5 sous la forme d'un double à la variable d.
```

2. Les opérateurs arithmétiques

Les opérateurs arithmétiques permettent d'effectuer des calculs sur leurs opérandes.

Opérateur	Nom de l'opération
+	Addition
*	Multiplication
-	Soustraction
/	Division
%	Modulo (reste de la division entière)

Exemple

```
int x = 11, y = 3, a, b, c, d, e;    //Définition des variables.
a = x + y;                           //a = 14
b = x * y;                           //b = 33
c = x – y;                           //c = 8
```

```
d = x / y;                    //d = 3
e = x % y;                    //e = 2
```

l'opérateur d'addition + appliquer avec les chaines de caractères permet de faire la concaténation (joindre) de deux chaines de caractères.

Exemple

```
string s1 ="Bonjour", s2 = " tout le monde.", s;    //Définition de deux chaines.
s = s1 + s2;                                        //s = Bonjour tout le monde.
```

3. Les opérateurs de comparaison

Les opérateurs de comparaison permettent de définir des expressions dont le résultat est une valeur booléenne. Ils sont principalement utilisés pour l'évaluation de conditions dans les structures de contrôle.

Opérateur	Nom de l'opération
==	Égalité
!=	Inégalité
>	Supérieur
>=	Supérieur ou égale
<	Inférieur
<=	Inférieur ou égale
is	Comparaison de type

Exemple

```
int x = 3, y = 5;
var a1 = x == y;                  //a1 = false
var a2 = x != y;                  //a2 = true
var a3 = x > y;                   //a3 = false
var a4 = x >= y;                  //a4 = false
var a5 = x < y;                   //a5 = true
var a6 = x <= y;                  //a6 = true
var a7 = "Mohamed" is string;     //a7 = true
```

4. L'opérateur ternaire

L'opérateur ternaire renvoie une valeur en fonction d'une expression booléenne. La syntaxe est comme suit :

```
var x =  (condition) ? valeur_si_vrai : valeur_si_faux;
```

Exemple

```
int x = 3;
var res = x > 2 ? x*x : x*2;    // res = 9
```

5. Opérateur de fusion de valeur nulle

Cet opérateur accepte deux opérandes et renvoie le premier des deux dont la valeur n'est pas null.

Exemple

```
string s1 = "Mohamed", s2 = null;
var res1 = s1 ?? "Non définie";          //res1 = Mohamed
var res2 = s2 ?? "Non définie";          //res2 = Non définie
```

6. Les opérateurs logiques

À l'exception de l'opérateur "!", les opérations logiques permettent de combiner plusieurs expressions renvoyant un booléen.

<table>
<tr><th>Opérateur</th><th>Nom de l'opérateur</th><th>Table de vérité</th></tr>
<tr><td>!</td><td>La négation : permet d'inverser une valeur booléenne.</td><td>
<table>
<tr><th>X</th><th>!X</th></tr>
<tr><td>true</td><td>false</td></tr>
<tr><td>false</td><td>true</td></tr>
</table>
</td></tr>
<tr><td>&&</td><td>Le ET conditionnel : permet de combiner deux expressions afin de déterminer si elles sont toutes deux vraies. La seconde expression n'est évaluée que si la première est évaluée comme vrai.</td><td>
<table>
<tr><th>A</th><th>B</th><th>A && B</th></tr>
<tr><td>true</td><td>true</td><td>true</td></tr>
<tr><td>true</td><td>false</td><td>false</td></tr>
<tr><td>false</td><td>true</td><td>false</td></tr>
<tr><td>false</td><td>false</td><td>false</td></tr>
</table>
</td></tr>
<tr><td>||</td><td>Le OU conditionnel : permet de combiner deux expressions afin de déterminer si au moins une des deux est vraie. La seconde expression n'est évaluée que si la première est évaluée comme faux.</td><td>
<table>
<tr><th>A</th><th>B</th><th>A || B</th></tr>
<tr><td>true</td><td>true</td><td>true</td></tr>
<tr><td>true</td><td>false</td><td>true</td></tr>
<tr><td>false</td><td>true</td><td>true</td></tr>
<tr><td>false</td><td>false</td><td>false</td></tr>
</table>
</td></tr>
</table>

Exemple

```
bool x = true, y = false;
Console.WriteLine(!x);          //false
Console.WriteLine(!y);          //true
Console.WriteLine(x && y);      //false
Console.WriteLine(x || y);      //true
```

7. Les opérateurs binaires

Les opérateurs binaires ne peuvent être appliqués que sur des types numériques entiers. Ils effectuent sur leurs opérandes des opérations logiques au niveau des **bits**. Nous retrouvons :

- ***ET binaire*** **&** : effectue un ET logique sur chacun des bits des opérandes.
- ***OU binaire*** | : effectue un OU logique sur chacun des bits des opérandes.
- ***OU EXCLUSIF binaire*** ^ : effectue un OU exclusif sur chacun des bits des opérandes.
- ***NEGATION*** ~ : inverse la valeur de chacun des bits de son opérande.
- ***Décalage vers*** la ***droite*** >> : décale vers la droite les bits composant le premier opérande du nombre de positions spécifiées par le second opérande.

❖ ***Décalage vers la gauche*** << : décale vers la gauche les bits composant le premier opérande du nombre de positions spécifiées par le second opérande.

8. Opérateur d'incrémentation et de décrémentation

L'opérateur d'incrémentation et de décrémentation permettent respectivement d'ajouter et de diminuer 1 de la variable associée.

Opérateur	Opération
++	Incrémentation
--	Décrémentation

Exemple

```
int x = 3;
x++;
Console.WriteLine(x);   // x= 4
++x;
Console.WriteLine(x);   // x = 5
x--;
Console.WriteLine(x);   // x = 4
--x;
Console.WriteLine(x);   // x = 3
```

On parle de post-incrémentation quand les ++ sont placer après la variable et de pré-incrémentation quand le ++ sont placés avant la variable. De même pour la décrémentation.

Exemple

```
int x = 3;
var y = ++x;
Console.WriteLine(x);   // x = 4
Console.WriteLine(y);   // y = 4

x = 3;
y = x++;
Console.WriteLine(x);   // x = 4
Console.WriteLine(y);   // y = 3

x = 3;
y = --x;
Console.WriteLine(x);   // x = 2
Console.WriteLine(y);   // y = 2

x = 3;
y = x--;
Console.WriteLine(x);   // x = 2
Console.WriteLine(y);   // y = 3
```

9. Opérateur d'affectation composé

La syntaxe de l'opérateur est la suivante :

Valeur1 **op=** valeur2 ;

Opérateur	Signification
x += y	x = x + y
x -= y	x = x - y
x /= y	x = x / y
x *= y	x = x * y
x %= y	x = x % y

Exemple

```
int x = 3, y = 2;
var y += x;
Console.WriteLine(y);   // y = y + x = 5

y = 2;
y -= x;
Console.WriteLine(y);   // y = y - x = 0

y = 2;
y *= x;
Console.WriteLine(y);   // y = y * x = 6

y = 2;
y /= x;
Console.WriteLine(y);   // y = y / x = 0

y = 2;
y %= x;
Console.WriteLine(y);   // y = y % x = 2
```

VII. La conversion de type

On entend par conversion de type, le passage d'un type de donnée vers un autre. Par exemple le passage d'un int vers un double ou vice-versa. Nous allons nous intéresser uniquement à la conversion des variables de type primitifs.
Le tableau ci-dessous montre les conversions implicites possibles :

Du type	Vers le type
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
int	long, float, double, decimal

long	float, double, decimal
float	double
sbyte	short, int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
uint	long, ulong, float, double, decimal
ulong	float, double, decimal
char	int, uint, long, ulong, float, double, decimal

Examinons le code ci-dessous :

```
byte v1 = 2;
short v2 = 3;
int v3 = 4;
long v4 = 5;
//Conversion entre entiers sans perte d'information
v2 = v1;    //Affectation de byte en short
v3 = v2;    //Affectation de short en int
v3 = v1;    //Affectation de byte en int
v4 = v1;    //Affectation de byte en long
v4 = v2;    //Affectation de short en long
v4 = v3;    //Affectation de int en long
//Déclaration de deux réels
float v5 = 1.2F;
double v6 = 2.2;
//Conversion entre réels sans perte d'information
v6 = v5;
```

Dans le cas où l'opération de conversion implique une perte d'information, le compilateur exige que vous l'indiquiez explicitement (opération de **cast** ou **conversion explicite**) selon la syntaxe suivante :

typeVariable1 variable1 = **(typeVariable1)** variable2;

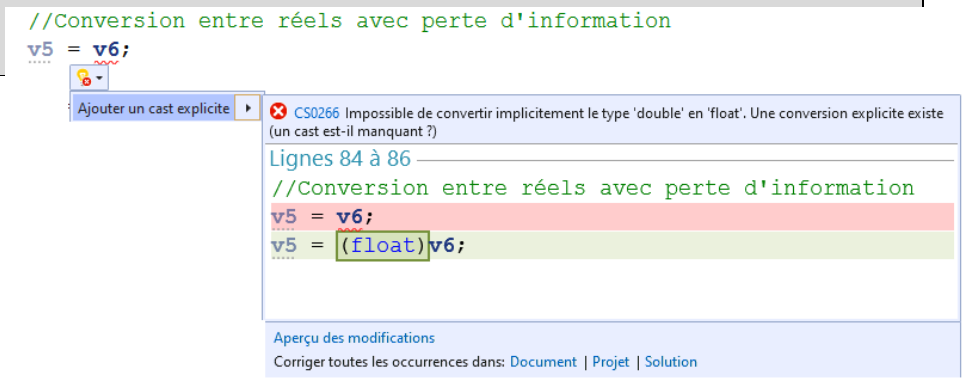
Exemple

```
//Déclaration de 4 entiers
byte v1 = 2;
short v2 = 3;
int v3 = 4;
long v4 = 5;
//Conversion entre entiers avec perte d'information
```

```
v1 = (byte)v3;      //Affectation de int dans un byte
                     //Même chose si vous convertissez un long vers un short...

//Déclaration de deux réels
float v5 = 1.2F;
double v6 = 2.2;
//Conversion entre réels avec perte d'information
v5 = v6;

v5 = (float)v6;
```



VIII. Les mots clé en C#

Les mots-clés sont des noms réservés par le langage C#. Ils sont interprétés par le compilateur et ne peuvent donc pas être utilisés en tant qu'identifiants (colorés en bleu par défaut sous visual studio).

Ci-dessous une liste des mots-clés du langage C# :

abstract	add	as	ascending	async
await	base	bool	break	by
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	descending	do	double	dynamic
else	equals	enum	event	explicit
extern	false	finally	fixed	float
for	foreach	from	get	global
goto	group	if	implicit	in
int	interface	internal	into	is
join	let	lock	long	nameof
namespace	new	null	object	on
operator	orderby	out	override	params
partial	private	protected	public	readonly
ref	remove	return	sbyte	sealed
select	set	short	sizeof	stackalloc
static	string	struct	switch	this
throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using
value	var	virtual	Volatile	void
where	While	yield		

IX. Liens utiles :

Voir la documentation officielle du langage C# sur le lien suivant : https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/