C#

Les types de base et les entrées sorties

SOMMAIRE

OBJECTIFS

L'objectif de ce chapitre est l'approfondissement de ce que nous venons de découvrir au travers de ce premier exemple.

- La déclaration et l'utilisation des variables et des constantes.
- La lecture et l'écriture des données numériques sur les flux standard et l'utilisation des manipulateurs.
- Les opérateurs de calcul.

Le système de types communs (CTS) définit deux types de variables :

Les types valeur : variables qui contiennent directement leurs données : ce sont les types intégrés, les structures, les variables énumération.

Les types référence : variables qui stockent des références à des données : leurs données sont stockées dans une zone séparée de la mémoire (classe String, classes).

LES TYPES INTEGRES

Définition

Les types de données simples - de type scalaire (entiers) ou virgule flottante (réels) - sont identifiés par des mots clés réservés, ou en utilisant le type struct prédéfini.

Type C#	Type .NET	Taille	Val. Min.	Val. Max.	Préfixe
bool	**System.Boolean**	1	False	True	**b**
byte	**System.Byte**	1	0	255	
sbyte	**System.SByte**	1	-128	127	
char	**System.Char**	2	0	65 535	**c**
short	**System.Int16**	2	-32 768	32 767	**s**
ushort	**System.UInt16**	2	0	65 535	**us**
int	**System.Int32**	4	-2 147 483 648	2 147 483 647	**i**
uint	**System.UInt32**	4	0	4 294 967 295	**ui**
long	**System.Int64**	8	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	**l**
ulong	**System.UInt64**	8	0	18 446 744 073 709 551 615	**ul**
float	**System.Single**	4	-1.5×10^{-45}	3.4×10^{38}	**f**
double	**System.Double**	8	-5.0×10^{-324}	1.7×10^{308}	**lf**
decimal	**System.Decimal**	16	-1.0×10^{-28}	env. 7.9×10^{28}	**d**

La première colonne indique le type de données que l'on peut utiliser dans un programme écrit en C#.

La seconde colonne indique le type de donnée correspondant dans l'environnement .NET. A noter qu'il est également possible, bien entendu, de les utiliser directement en C# (en omettant **`System.`** Si la directive **`using System`** est présente dans le programme).

La troisième colonne donne la taille en octets (place prise en mémoire) d'une variable déclarée avec ce type.

La quatrième colonne indique la valeur minimum admise pour une donnée de ce type.

La cinquième colonne indique la valeur maximum admise pour une donnée de ce type.

La sixième colonne indique le préfixe qu'il est préférable de mettre devant l'identificateur d'une variable de ce type. Ce préfixe n'est pas obligatoire mais renseigne précieusement tout développeur et à chaque niveau d'un programme de quel type est une variable. Dans la littérature, certains ouvrages utiliseront les préfixes, d'autres non.

Dans le code, il est possible de déclarer la variable de 2 façons :

```
double rayon
```

ou

```
System.Double rayon
```

Qui définiront toutes deux, un réel signé sur 8 octets.

Les constantes numériques entières peuvent être écrites sous 3 formes :

- en décimal
- en hexadécimal (base 16)
- en caractère.

Ainsi, 65 (décimal), 0x41 (hexadécimal), 'A' désignent la même constante numérique entière (le code ASCII de A est 65).

Exemples :

```
long grandEntier ;          // entier signé sur 8 octets
float leReel ;              // réel en notation virgule flottante (mantisse, exposant)
double dblPrecision ;       // idem en double précision
const int N1=65 ;
const int N2=0x41 ;
const int N3='A';
const int N4 =N1 + N2 ;
const int N5 = N1 + 10 ;
```

ATTRIBUTION DE VALEURS AUX VARIABLES

Une valeur peut être attribuée à une variable lors de sa déclaration.

```
double pi = 3.14159 ;
```

ce qui est équivalent à

```
double pi;
pi = 3.14159 ;
```

On parle d'instruction d'**affectation** ou **d'assignation**, et on lit PI **prend pour valeur** 3.14159.
Pour attribuer une valeur à une variable de type caractère, on écrira :

```
char lettre;
lettre = 'A' ;
```

Types d'affectation :

```
int resultat, Y = 3 ;       //seule la variable Y est initialisée à 3
resultat = 5 ;              // résultat prend la valeur 5
resultat = Y ;              // résultat prend la valeur 3
                            // Y reste à 3
resultat = Y + 2;           // résultat prend la valeur 5
resultat = resultat + 2 ;   // résultat prend la valeur 7
```

On pourra utiliser la syntaxe abrégée, quelque soit l'opérateur arithmétique :

```
Y = Y + 2 ;                 // Y prend la valeur 5
Y += 2 ;                    // syntaxe abrégée
```

Attention :

```
double d = 2.5 ;            //syntaxe OK
float f1 = 2.5 ;            //pas bon : 2.5 est au format double
float f2 = 2.5f ;
float f3 = (float)[1]2.5 ;  // syntaxe OK
float f4 = 2.5e10f;
```

[1] Voir le paragraphe relatif au casting

LES OPERATEURS

Arithmétiques

+	Addition	**a + b**
-	Soustraction	**a - b**
-	Changement de signe	**-a**
*****	Multiplication	**a * b**
/	Division	**a / b**
%	Reste de la division entière	**a % b**

Lorsqu'une expression contient plusieurs opérateurs, l'ordre dans lequel sont effectués les calculs dépend de l'ordre de priorité des opérateurs.

L'expression x + y * z est évaluée sous la forme x + (y * z) car l'opérateur de multiplication a une priorité supérieure à celle de l'opérateur d'addition.

Règle : Les opérateurs – et + ont une priorité plus basse que celle des opérateurs *, / et %. On peut contrôler la priorité à l'aide de parenthèses.

D'affectation

=	Affectation	**a = 5;**
+=	Incrémentation Les deux exemples sont équivalents	**a += b;** **a = a + b;**
-=	Décrémentation Les deux exemples sont équivalents	**a -= b;** **a = a - b;**

D'incrémentation

++	Pré-incrémentation (+1)	**++a**
++	Post-incrémentation (+1)	**a++**
--	Pré-décrémentation (-1)	**--a**
--	Post-décrémentation (-1)	**a--**

Exemples :

```
int   a, b;

a = 5;
b = a++;
// a = 6 et b = 5;

a = 5;
b = ++a;
```

```
// a = 6 et b = 6;
```

a++ est incrémenté après affectation, alors que **++a** est incrémenté avant affectation.

CONVERSION IMPLICITE ET CASTING

Quand deux opérandes de part et d'autre d'un opérateur binaire (qui a deux opérandes) sont de types différents, une conversion implicite est effectuée vers le type "le plus fort" en suivant la relation d'ordre suivante :

bool < byte < char < short < int < long < float < double

Exemple : La conversion d'un type de données **int** en un type de données **long** est implicite : cette conversion réussit toujours et n'entraîne jamais de perte d'informations.

```
int    a = 78;
long   b = a;
```

Par contre, le compilateur ne voudra pas exécuter l'inverse, le code suivant

```
long   a = 78;
int    b = a;
```

Il est impossible de convertir implicitement un type **long** en un type **int**, car le risque de perdre de l'information existe. ***Une conversion explicite(casting) doit être mise en place***.

```
long   a = 78;
int    b = (int) a;
```

Un ***casting*** se fait en mentionnant le type entre parenthèse avant l'expression à convertir.

Par ailleurs, un opérateur binaire dont les deux opérandes sont de même type opère dans ce type. Ce qui semble donner parfois des résultats curieux. Pour avoir le résultat attendu, il faut forcer la conversion par un ***casting*** (changement de type) afin de préciser dans quel référentiel on opère.

Exemples :

```
int n1 = 5;
int n2 = 2;
double x = n1 / n2;
```

Aussi bizarre que cela paraisse, la valeur de x est de 2.0. En effet, 5 et 2 sont des entiers. Le résultat de la division entière de ces deux nombres est 2 (et il reste 1) et non pas 2.5. Pour obtenir cette dernière valeur, il faut forcer la conversion de l'un des opérandes en un nombre virgule flottante :

```
double x = (double)n1 / n2;
```

Le fait de faire le *casting* **`(double)`** devant **`5`** force la conversion de 5 (type **int**) en 5.0 (type **double**). L'opérateur **/** va donc devoir opérer sur un **double** et un **int**. Il y a alors une conversion implicite de 2 (type **int**) en 2.0 (type **double**). L'opérateur peut maintenant opérer sur deux **double** et le résultat (2.5) est un **double**.

LE TYPE CHAINE

Définition

En C#, une chaîne de caractères est un objet de la classe **String**, de l'espace de nom **System**.
Ainsi une chaîne peut être déclarée :

```
string strChaine ;
```

ou

```
String strChaine;
```

La variable `strChaine` ne *contient* pas la chaîne de caractères : elle est la référence d'une chaîne de caractères. Telle qu'elle est déclarée ci-dessus, la valeur de `strChaine` est **`null`**.

On pourra l'initialiser :

```
strChaine = "Bonjour" ;  // si elle est préalablement déclarée
```

ou

```
string strChaine = "Bonjour" ;
```

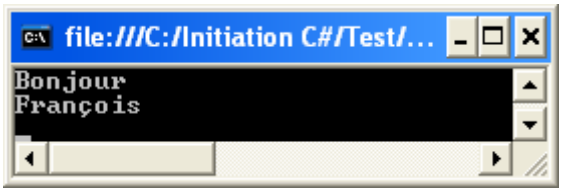
Entre les doubles-quottes, le caractère \ est utilisé comme modificateur et interprète le caractère immédiatement après pour coder des caractères non-affichables comme le saut de ligne ou une tabulation :

\n	Saut de ligne
\r	Retour de chariot
\t	Tabulation
\b	Retour arrière
\v	Tabulation verticale
\f	Saut de page
****	Le caractère \ lui-même
\'	Le caractère ' (quote) lui-même
\"	Le caractère " (double quote) lui-même
\xHHHH	Caractère de code HHHH en hexadécimal

Exemple : Le code suivant

```
string strNom = "François";
Console.WriteLine("Bonjour \n" + strNom);
```

produira l'affichage suivant :



La **concaténation** de chaînes de caractères s'obtient grâce à l'opérateur **+**. Le résultat de cette opération est une nouvelle chaîne de caractères qui peut être affectée à une variable de type **`String`** :

```
string strMsg = "Bonjour " + strPrenom;
```

On accède à un caractère de rang fixé d'une chaîne par l'opérateur [ ] (la chaîne est lue comme un tableau de char, le premier caractère est indicé par 0) :

```
char lettre = strNom [3] ; // lettre contient le caractère n
```

par contre

```
strNom [3] = "B";          // provoque une erreur de compilation
```

Le texte d'une chaîne ne peut pas être modifié après sa création (immuable).

MANIPULATION DE CHAINE DE CARACTERES

La classe **String** possède intrinsèquement les méthodes (fonctions membres) nécessaires à la manipulation des chaînes de caractères. Comme beaucoup de méthodes définies dans une classe d'objet, elles s'utilisent conjointement à une variable de type **String** avec le caractère **.** (point).

Toutes les méthodes de la classe **String**, ne seront pas traitées ici. Pour plus de précisions la documentation en ligne.

Longueur d'une chaîne de caractères

La longueur d'une chaîne de caractères est calculée par la propriété **`Length`** d'un objet de la classe **`String`**.

```
string strTitre = "Support de cours C#";
int iLen = strTitre.Length;   // iLen vaudra donc 19
```

Position d'une chaîne dans une autre

La méthode d'instance **`IndexOf`** permet de calculer la position d'une séquence de caractères dans une chaîne. La valeur **`0`** de cette position correspond au premier caractère de la chaîne. La valeur **`-1`** est retournée si la séquence de caractères n'a pas été trouvée dans la chaîne. Il existe plusieurs versions de cette fonction dont les plus utilisées sont les suivantes :

```
int pos1 = str1.IndexOf(str2);
int pos2 = str1.IndexOf(str2, iPos);
```

Dans le premier cas **`pos1`** contiendra la position de **`str2`** dans **`str1`**. Dans le second cas, la position de **`str2`** sera cherchée à partir du **`iPos`**^{ième} caractère de **`str1`**. Ceci permet de repérer plusieurs occurrences de **`str2`** dans **`str1`** en utilisant une boucle de parcours.
Exemples:

```
string str1 = "Pascal disait: je pense, donc je suis";
int pos1 = str1.IndexOf( "je" );           // pos1 = 15
int pos2 = str1.IndexOf( "je", pos1 + 1 ); // pos2 = 30
```

Extraction d'une sous-chaîne

La méthode d'instance **`Substring`** permet d'extraire une sous-chaîne d'une chaîne de caractères :

```
string strPrenom = strNom.Substring(0, 6);
// Extrait les 6 premiers caractères de la chaine strNom
```

Conversion Majuscules / Minuscules

Les méthodes d'instance **ToUpper** et **ToLower** permettent de retourner une chaîne dont tous les caractères sont convertis respectivement en majuscules et en minuscules.

```
string strSalut = "Bonjour Tout Le Monde";
Console.WriteLine(strSalut.ToUpper());// BONJOUR TOUT LE MONDE
Console.WriteLine(strSalut.ToLower());     // bonjour tout le monde
```

Suppression d'espaces

La méthode d'instance **Trim** supprime tous les espaces de début et fin de chaîne.

```
string strSalut = "   Bonjour   ";
Console.WriteLine(strSalut.Trim());  // Bonjour
```

Comparaison de chaînes de caractères

La méthode **Equals** permet de comparer l'égalité de 2 chaînes : elle renvoie un booléen positionné à **true / false** si les 2 chaînes sont identiques/différentes ; elle peut être employée de 2 façons :

Exemple :

```
string str1 = "abc";
string str2 = "abc";
```

- En tant que méthode d'instance

```
bool bIden = str1.Equals(str2);          // bIden = true
```

- En tant que méthode de classe

```
bool bIden = String.Equals(str1,str2);  // bIden = true
```

Notez la différence entre une méthode de classe et une méthode d'instance : nous reviendrons sur le sujet dans un prochain chapitre.

La méthode de classe **Compare** permet de comparer les chaînes en fonction de leur ordre de tri : elle renvoie **0** si les chaînes sont identiques, **-1** si la 1ere chaîne précède la 2eme, **1** si la deuxième précède la 1ere.

```
string str1 = "abc";
string str2 = "def";
int iIden = String.Compare(str1,str2);  // iIden = -1
```

Une autre version de la méthode permet de positionner un booléen en 3eme paramètre qui spécifiera si la casse doit être ignorée ou non.

```
string str1 = "abc";
string str2 = "Abc";
```

```
int iIden = String.Compare(str1,str2, true); // iIden = 0
```

La casse est ignorée.

Remplacement de sous chaînes dans une chaîne

La méthode **Replace** permet de remplacer toutes les occurrences d'une sous chaîne de la chaîne de base par une autre.

```
string str1 = "Bonjour";
string str2 = str1.Replace("on","ON ") ;
// str2 contient "BON jour", str1 est inchangé
```

Eclatement d'une chaîne en tableau de sous chaînes

La méthode **Split** permet de retourner un tableau de mots constituant la chaîne de base en spécifiant le(ou les) séparateur(s).

```
string strSalut = "Bonjour Tout Le Monde";
string[ ] tsc = strSalut.Split( ' ') ; // un seul séparateur
// tsc[1] contient "Bonjour", tsc[2] contient "Tout",
// tsc[3] contient "Le", tsc[4] contient "Monde",
```

Les opérations d'entrée sortie

L'affichage des données se fait à travers le flux de sortie **Console**. à l'aide des méthodes **Write()** ou **WriteLine()**.

Exemples :

```
Console.WriteLine("Bon")
```

L'argument est une chaîne de caractères affichée avec un retour et un saut de ligne.

```
int i=5 ;
Console.WriteLine(i);
```

L'argument est un entier : une des formes de **WriteLine** ou **Write** accepte un entier en argument ; l'entier est converti en une chaîne de caractères pour l'affichage ; d'autres formes acceptent les autres types (bool, char, double …)

```
int i=5 , j=4 ;
Console.WriteLine(" i = " + i + ", j= " + j);     (équivalent à )
Console.WriteLine(" i = {0}, j= {1} ", i, j);
```

Les valeurs de i et j sont respectivement positionnées à la place des paramètres 0 et 1

Mise en format de chaînes de caractères

Chaque spécificateur a la forme **{ N [[, M] : *chaîne de format*] }** où :

- N désigne le numéro de l'argument (0,1)
- M, facultatif désigne le nombre de positions d'affichage, valeur positive pour un alignement à droite, ou négative pour un alignement à gauche
- La chaîne de format, facultative peut représenter un format standard

```
double unDouble = 123456789;
Console.WriteLine(" le double = {0:e} ", unDouble);
// rend le double = 1.2345678e+008
Console.WriteLine(" le double = {0:f} ", unDouble);
// rend le double = 123456789.00
Console.WriteLine(" le double = {0:n}", unDouble);
// rend le double = 123 456 789.00
```

ou personnalisé, où les caractères suivants ont une signification particulière :

  - 0 représente un chiffre
  - # représente un chiffre ou rien
  - % le nombre sera multiplié par 100
  - , représente le séparateur de milliers
  - E indique une représentation scientifique

```
double unDouble = 34.5;
Console.WriteLine(" le double = {0:##.#0} ", unDouble);
// le double = 34.50
```

```
double unDouble = 34567.89;
Console.WriteLine(" le double = {0:##,###.##} ", unDouble);
// le double = 34 567,89
int unEntier = 34;
Console.WriteLine(" l'entier = {0,5:000} ", unEntier);
// l'entier =   034  La valeur 34 est précédée de 2 espaces.
int unEntier = 34;
Console.WriteLine(" l'entier = {0:000.00} ", unEntier);
// l'entier = 034,00
```

(Consultez l'aide en ligne pour plus de détails en cherchant String.Format(méthode) puis Vue d'ensemble des formats et Chaînes de format numériques standard / Chaînes de format numériques personnalisées)

La lecture des données se fait à travers le flux d'entrée **Console**. à l'aide de la méthode **`ReadLine`**. Cette méthode pose un problème principal :
Elle ne permet de ne lire qu'une chaîne de caractères **`string`** Il faut donc convertir explicitement cette chaîne de caractères pour obtenir l'objet numérique souhaité (**`Double`**, **`Float`**, **`Long`**, **`Int32`**, etc…):

Exemples :

```
// Pour lire l'entier k déclaré de type int
string strLine;
strLine = Console.ReadLine();
int k = Convert.ToInt32( strLine );
//ou encore
k = Int32.Parse( strLine );
```

La classe **System.Convert** fournit un jeu complet de méthodes pour les conversions prises en charge. On peut, par exemple convertir des types **string** en types numériques, des types **DateTime** en types **string** et des types **string** en types **boolean** .

D'autre part, tous les types numériques disposent d'une méthode **Parse** statique pouvant être utilisée pour convertir une représentation sous forme de chaîne d'un type numérique en un type numérique réel.
Les deux méthodes sont équivalentes.

Attention : Le programme se plantera si l'utilisateur introduit autre chose qu'un entier correct. Le problème sera résolu dans un chapitre ultérieur.

```
// Pour lire le nombre dSalaire déclaré de type double
string strLine;
strLine = Console.ReadLine();
double dSalaire = Convert.ToDouble( strLine );
// ou encore
dSalaire = Double.Parse( strLine );
```

Comme dans l'exemple précédent, le programme *se plantera* si l'utilisateur introduit autre chose qu'un réel. Le séparateur de décimales saisi doit être conforme aux caractéristiques régionales.

Exercices

CALCULATRICE

Ecrivez un programme qui demande deux nombres à l'utilisateur et qui affiche la somme de ces deux nombres.

CALCULATRICE SUITE

Modifier le programme précédent pour effectuer une Division.

CONVERSION DE TEMPERATURES

En utilisant la formule C = (5/9)(F-32) écrire un programme qui lit une température exprimée en degrés Fahrenheit et affiche sa valeur en degrés Celsius.

CALCUL DE LA MOYENNE PONDEREE DES NOTES D'UN ETUDIANT DANS UNE MATIERE

Il s'agit de calculer une moyenne sur la base de trois notes sachant que :

- une note de devoir surveillé a un coefficient de 3
- une note d'interrogation écrite a un coefficient de 2
- une note de travaux pratique a un coefficient de 1

CONVERSION D'UNE DUREE EXPRIMEE EN SECONDES EN HEURES, MINUTES ET SECONDES

Il s'agit pour un nombre de secondes entré au clavier d'en déduire, son expression en nombre d'heures de minutes et de secondes.

CONSOLE :

Ecrire un programme qui saisit un code Unicode en décimal et affiche le caractère correspondant.
Exemple la saisie de l'entier 65 donne le caractère « A ».

Que se passe t-il lorsque vous tapez 7 ?