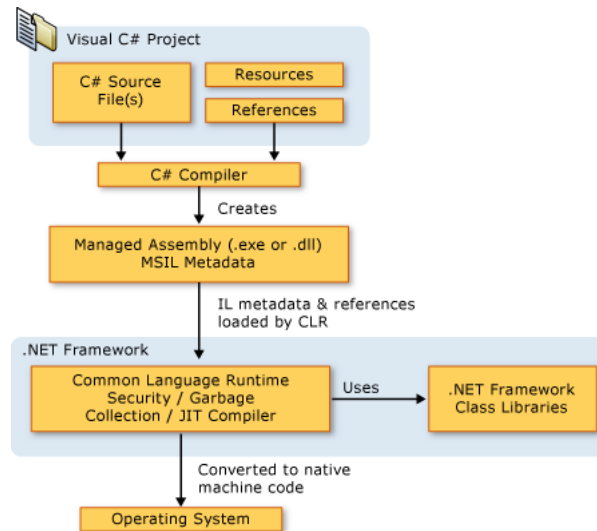


# 1 Module A.1 : Présentation

## 1.1 Préambule

Le C# est un langage de programmation créé, développé par Microsoft en 2002 et destiné à développer sur la plateforme Microsoft .NET. Fortement inspiré du langage Java, il est un langage orienté objet, fortement typé. Le C# permet de développer des applications Web, de bureau, des commandes et des bibliothèques de classes. Le langage dispose d'un nombre de bibliothèques considérables.



Le code C# est compilé en un langage intermédiaire MSIL (**Micro**Soft **I**ntermediate **L**anguage). Le code en langage intermédiaire est ensuite interprété en langage machine. Cette particularité permet de construire des applications composées de programmes réalisés dans des langages couverts comme le C#, le F#, VB .Net, Python ou encore le C++.

## 1.2 Premier programme en C#

Pour commencer, voici un premier programme écrit en C# dans Visual Studio de cette formation.

```
1 using System;                // Import de l'espace de noms.
2
3 namespace PremierProgramme // Déclaration de l'espace de noms utilisé
4 {
5     class Program            // Déclaration de la classe Program
6     {
7         static void Main()    // Déclaration de la méthode principale
8         {
9             int x = 12 * 30;    // Instruction 1
10            Console.WriteLine("La valeur de x est égale a : " + x);
11            int y = 10;
12            Console.WriteLine("La valeur de y est égale a : " + y);
13            int z = x + y;
14            Console.WriteLine("La somme de x et de y est égale a : " + z);
15        }                    // Fin de la méthode
16    }                        // Fin de la classe
17 }
```

Ce programme paraît pour l'instant fort compliqué, mais les lignes suivantes s'appliquent à le rendre intelligible. Le programme commence par la déclaration des espaces de noms utilisés dans le fichier .cs. La première ligne importe **System** qui contient des éléments de base pour la bonne exécution d'un programme C#. Ce n'est pas important pour le moment. A ce stade, il suffit de comprendre que lorsque l'on veut réaliser une certaine tâche, il faut importer la librairie permettant de la réaliser. La ligne 3 correspond à la déclaration de l'espace de nom du programme. La ligne 5 correspond à la déclaration de la classe principale du programme. C# est un langage-objet. La ligne 7 correspond à la déclaration de la méthode Main (principale). Elle est importante, car elle est le point d'entrée du programme.

Avant de décrire le corps de la méthode principale, il est important d'indiquer quelques éléments de base de la syntaxe d'un programme en C#. Un programme en C# est composé d'une série d'instructions qui s'exécutent de manière séquentielle. Ces dernières se terminent par un point-virgule.

Le programme précédent effectue les actions suivantes :

- Définit une variable entière avec un nom x et une valeur la multiplication de 12 et 30.
- Affiche « La valeur de x est : 360 ».
- Définit une variable entière avec un nom y et une valeur 10.
- Affiche « La valeur de y est : 10 ».
- Définit une variable entière avec un nom z et une valeur la somme de la valeur contenue dans x et celle contenue dans y.
- Affiche « La somme de x et de y est : 370 ».

Quand on exécute le programme précédent, on observe le résultat suivant :

```
La valeur de x est : 360
La valeur de y est : 10
La somme de x et de y est : 370
```

L’affichage se fait à partir d’une classe `Console` qui fournit des méthodes utilisables pour afficher des éléments dans la console de terminal. Cette classe est présente dans l’espace de noms **System**.

## 1.3 Types simples

Les programmes informatiques consistent en une série d’instructions dont le but est de prendre des données en entrée et fournir un résultat en sortie. La représentation des données est une préoccupation de tous les langages informatiques et toute une partie de cette formation consiste en une présentation de celle-ci.

En C#, on représente les données avec des variables. Une variable est une association entre un nom et une valeur stockée en mémoire. Comme le C# est un langage typé statiquement, chaque variable possède un type. Ce type ne peut plus évoluer après sa déclaration. Lors de la déclaration d’une variable, on lui attribue un type, un nom et éventuellement une valeur.

Par exemple, la déclaration et l’affectation d’une variable longueur de type `int` et de valeur entière 12.

```
1 int longueur;
2 longueur = 12;
```

Le compilateur va allouer une zone mémoire correspondant aux caractéristiques du type déclaré. Le nom permet au compilateur de se souvenir de la zone de mémoire utilisée. Il existe une série de types prédéfinis que l’on peut utiliser directement. Les sections suivantes en présentent quelques uns.

## 1.4 Types numériques

### 1.4.1 Entiers

#### Caractéristiques :

Les premiers types présentés sont les types numériques permettant de représenter des entiers. Ci-dessous, un tableau récapitulatif des différents types existants.

Mot clé C#	Taille	Valeur minimale	Valeur maximale
<b>sbyte</b>	8 bits signés	-128	127
<b>byte</b>	8 bits non signés	0	255
<b>short</b>	16 bits signés	-32 768	32 767
<b>ushort</b>	16 bits non signés	0	65 535
<b>int</b>	32 bits signés	-2 147 483 648	2 147 483 647
<b>uint</b>	32 bits non signés	0	4 294 967 295
<b>long</b>	64 bits signés	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
<b>ulong</b>	64 bits non signés	0	4 294 967 295

L'affectation d'une valeur à un type intégral à l'aide d'un littéral est réalisable de trois manières différentes. Un littéral est par défaut en base décimale. Si le littéral est en base hexadécimale ou en base binaire, le préfixe *0x* (*0X*) ou *0b* (*0B*) respectivement doit être déclaré. Le suffixe permet de choisir le type du littéral. S'il n'y pas de suffixe, le type du littéral est le premier type intégral signé en partant de **int** permettant de représenter ce nombre sans perte d'information. Le suffixe *u* (*U*) permet de suivre la même logique avec les types intégraux non signés. Le suffixe *L* permet de choisir explicitement le type **long** ou **ulong** pouvant le représenter.

Exemple : Déclaration et initialisation d'entiers

```
1 // Affectation d'une valeur au format binaire : 42.
2 sbyte petitEntierBinaire = 0B001010101;
3 // Affectation d'une valeur au format hexadécimal : 58.
4 sbyte petitEntierHex = 0X3A;
5 // Affectation d'une valeur au format decimal.
6 int entierClassique = 123456;
7 // Copie de la valeur de entierClassique : 123456.
8 int entier = entierClassique;
9 // Suffixe pour indiquer le type long du littéral.
10 long grandEntier = 123456789012L;
```

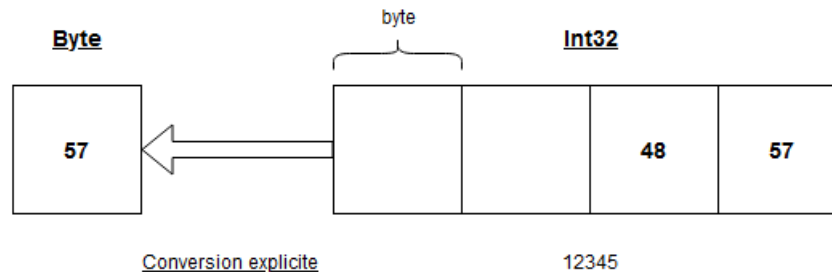
La valeur par défaut de chaque type intégral est 0. Les constantes `MinValue` et `MaxValue` fournissent les valeurs minimales et maximales pouvant être stockées dans le type en question.

### Conversion des types intégraux :

Tous les types intégraux sont convertibles les uns dans les autres. Lors de la conversion d'un type intégral vers un autre, si le type destinataire à une plage de valeurs qui englobe celle du type expéditeur la conversion est implicite. Si ce n'est pas le cas, la conversion peut entraîner une perte d'ordre de grandeur. Elle doit être explicite à l'aide d'un opérateur **cast** en déclarant le type destinataire entre parenthèses avant la valeur à convertir.

Exemple : Perte d'informations

```
1 int entier = 12345;
2 byte petitEntier = (byte) entier; // 57, conversion explicite.
```



Par contre, tous les types intégraux sont convertibles implicitement vers les types numériques à virgule flottante. L'inverse n'est pas vrai.

## 1.4.2 Nombres à virgule flottante

Alors que les types intégraux sont représentés de manière exacte, les nombres décimaux sont représentés de manière approximative. Ils ont une certaine précision. Le tableau suivant présente les trois types permettant de les représenter.

Mot clé C#	Taille	Précision	Plage approximative
<b>float</b>	4 octets	~ 6 – 9 chiffres	$\pm 1.5 \times 10^{-45}$ à $\pm 3.4 \times 10^{38}$
<b>double</b>	8 octets	~ 15 – 17 chiffres	$\pm 5.0 \times 10^{-324}$ à $\pm 1.7 \times 10^{308}$
<b>decimal</b>	16 octets	~ 28 – 29 chiffres	$\pm 1.0 \times 10^{-28}$ à $\pm 7.9228 \times 10^{28}$

Les variables dont le type est **decimal** ont une meilleure précision et sont donc adaptées dans le cadre des calculs financiers et monétaires, mais peuvent représenter une plage de valeurs plus courte. Attention au choix du type, une variable de type **decimal** est deux fois plus volumineuse qu'une variable de type **double** (quatre fois plus qu'une instance de type **int**).

L'affectation d'une valeur à un type numérique à virgule flottante peut être réalisée à l'aide d'un littéral. Un suffixe *f* (*F*) ou *m* (*M*) est nécessaire pour signaler le type de ce littéral dans le cas d'un type **float** ou **decimal** respectivement. Si le littéral est un **double**, aucun suffixe n'est nécessaire. Seule la conversion implicite **float** vers **double** est définie. Le littéral peut être en notation décimale ou en notation scientifique à l'aide du caractère *e* (*E*).

Exemple : Déclaration et initialisation de nombre décimaux

```
1 // Suffixe pour indiquer le type float du littéral.
2 float valeurDecimaleMoinsPrecis = 15.4569F;
3 // Pas de suffixe, le littéral est par défaut de type double.
4 double valeurDecimale = 15.364578;
5 // Utilisation de la notation scientifique.
6 double valeurNotationScientifique = 1.5658E-2;
7 // Suffixe pour indiquer le type decimal du littéral.
8 decimal valeurDecimalePrecise = 1.4589742564478415698m;
```

La valeur par défaut des types numériques à virgule flottante est 0. Ces types fournissent à l'instar des types intégraux les valeurs `MinValue` et `MaxValue`. Les types **double** et **float** fournissent également des valeurs pour représenter les signes  $-\infty$ ,  $+\infty$  à savoir `NegativeInfinity` et `PositiveInfinity`.

## 1.5 Autres types simples

### 1.5.1 Représentation d'une valeur booléenne

Le type **bool** représente un booléen avec deux valeurs possibles, *true* ou *false*. Toutes les expressions logiques sont de type booléen. La valeur par défaut du type **bool** est *false*.

Exemple : Déclaration et initialisation d'un booléen

```
1 bool estVraie = true; // Utilisation d'un littéral.
2 bool estLogique = 12 < 100 && 3 > 10; // false, opération logique.
```

### 1.5.2 Représentation de caractères

Le type **char** est une représentation d'un caractère Unicode de 16 bits encodé. La valeur par défaut d'un **char** est U+ 0000 soit 0. Il existe plusieurs représentations acceptées. La représentation usuelle ne demande pas de préfixe, la représentation sous forme de code Unicode et hexadécimale demande un préfixe `\u`, `\x` respectivement. Un littéral de type **char** est à mettre entre crochets `'char'`.

Exemple : Déclaration et initialisation de caractères

```
1 char caractereLiteral = 'a';
2 char caractereUnicode = '\u009A' // 154, préfixe Unicode.
3 char caractereUnicode = '\x009A' // 139, préfixe Hexadécimal.
```

Une instance de type **char** peut être convertie implicitement vers tous les types entiers sauf **byte**, **sbyte** et **short** et les types numériques à virgule flottante. Par contre, aucune conversion implicite n'existe depuis ces types vers le type **char**.