1 Bronze 1 : Variables

1.1 Types simples

Les programmes informatiques consistent en une série d'instructions dont le but est de prendre des données en entrée et de 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. A

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.

1.2 Types numériques

1.2.1 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
sbyte	8 bits signés	−128	127
byte	8 bits non signés	0	255
short	16 bits signés	−32 768	32 767
ushort	16 bits non signés	0	65 535
int	32 bits signés	−2 147 483 648	2 147 483 647
uint	32 bits non signés	0	4 294 967 295
long	64 bits signés	−9 223 372 036 854 775 808	9 223 372 036 854 775 807
ulong	64 bits non signés	0	18 446 744 073 709 551 615

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

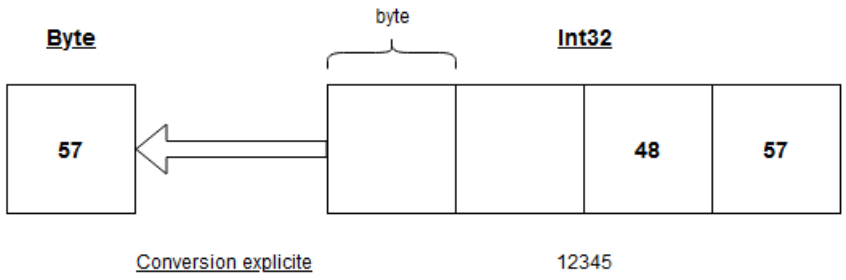
```
1  // Affectation d'une valeur au format binaire : 42.
2  sbyte petitEntierBinaire = 0B001010101;
3  // Affectation d'une valeur au format hexadecimal : 58.
4  sbyte petitEntierHex = 0X3A;
5  // Affectation d'une valeur au format decimal.
6  int entierClassique = 123456;
7  // Copie de la valeur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

1.2.2 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
float	4 octets	$\sim 6-9$ chiffres	$\pm 1.5 \times 10^{-45}$ à $\pm 3.4 \times 10^{38}$
double	8 octets	$\sim 15-17$ chiffres	$\pm 5.0 \times 10^{-324}$ à $\pm 1.7 \times 10^{308}$
decimal	16 octets	$\sim 28-29$ 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**). Cependant, le type **decimal** est le seul à pouvoir représenter exactement des valeurs comme 0.1 et éviter des erreurs d'arrondi préjudiciable.

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 litteral.
2  float valeurDecimaleMoinsPrecis = 15.4569F;
3  // Pas de suffixe, le litteral est par defaut 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 litteral.
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.

1.3 Autres types simples

1.3.1 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.
```

1.3.2 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 $\backslash u$, $\backslash x$ 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 intégraux 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**.