Chapitre 2 : Les structures de données en C#

2.1 Les types de données prédéfinis

Type	Type .NET	Description	Suffixe des valeurs	Plage des valeurs	Taille
short	System.Int16	Entier		-32 768 à 32 767	2 octets
ushort	System.UInt16	Entier non signé		0 à 65 535	2 octets
int	System.Int32	Entier		- 2 147 483 648 à 2 147 483 647	4 octets
unit	System.Uint32	Entier non signé	U	0 à 4294967295	4 octets
long	System.Int64	Entier non signé	L	-9 223 372 036 854 775 808 à -9 223 372 036 854 775 807	8 octets
ulong	System.UInt64	Entier	UL	0 à 18446744073709551615	8 octets
float	System.Single	Réel	F	-3,4028235E+38 à -1,401298E-45 pour les valeurs négatives 1,401298E-45 à 3,4028235E+38 pour les valeurs positives	4 octets
double	System.Double	Réel	D	-1,79769313486231570E+308 à -4,94065645841246544E-324 pour les valeurs négatives -4,94065645841246544E-324 à -1,79769313486231570E+308 pour les valeurs positives	8 octets
decimal	System.Decimal	Réel	M	+/- 79 228 162 514 264 337 593 543 950 335 sans chiffre décimal +/- 7,922 816 251 426 433 759 354 395 033 5 avec 28 positions à droite de la virgule	16 octets
byte	System.Byte	Entier		0 à 255	1 octets
sbyte		Entier signé		-128 à 127	1 octets
bool	System.Boolean	Booléen		True / False	1 octet
String	System.String	Chaîne de caractères		de 0 à environ 230 caractères	
Char	System.Char	Caractère		1 caractère	2 octets
Object	System.Object	Référence d'objet			
DateTime	System.DateTime	Date		01/01/1 à 31/12/9999	8 octets

2.2 Déclaration de Données

2.2.1 Déclaration d'une constante

Une constante est déclarée de la manière suivante :

Const type Identificateur = valeur

Exemples :

```
Const float PI = 3.14 ;
Const byte Coef  = 3 ;
```

2.2.2 Déclaration de variable

Une variable est déclarée de la manière suivante :

Nom_Type Nomvariable ;

Exemples :

```
byte age;
int x;
```

Ou

var variable1=valeur1,variable2=valeur2,...;

Le mot clé var ne veut pas dire que les variables n'ont pas un type précis. La variable variablei a le type de la donnée valeuri qui lui est affectée. L'initialisation est ici obligatoire afin que le compilateur puisse en déduire le type de la variable.

Exemple :

```
1. using System;
2.
3. namespace Chap2 {
4. class C2 {
5. static void Main(string[] args) {
6. int i=2;
7. Console.WriteLine("Type de int i=2 : {0},{1}",i.GetType().Name,i.GetType().FullName);
8. var j = 3;
9. Console.WriteLine("Type de var j=3 : {0},{1}", j.GetType().Name, j.GetType().FullName);
10. var aujourdhui = DateTime.Now;
11. Console.WriteLine("Type de var aujourdhui : {0},{1}", aujourdhui.GetType().Name,
aujourdhui.GetType().FullName);
12. }
13. }
14. }
```

Description de certaines lignes

• ligne 6 : une donnée typée explicitement
• ligne 7 : (donnée).GetType().Name est le nom court de (donnée), (donnée).GetType().FullName est le nom complet de (donnée)
• ligne 8 : une donnée typée implicitement. Parce que 3 est de type int, j sera de type int.
• ligne 10 : une donnée typée implicitement. Parce que DateTime.Now est de type DateTime, aujourdhui sera de type DateTime.

A l'exécution, on obtient le résultat suivant :

1. Type de int i=2 : Int32,System.Int32
2. Type de var j=3 : Int32,System.Int32
3. Type de var aujourdhui : DateTime,System.DateTime

2.3 Les conversions entre nombres et chaines de caractères

nombre -> chaîne nombre.ToString() *ou "" + nombre*
chaine -> int int.Parse(chaine) ou System.Int32.Parse
chaîne -> long long.Parse(chaine) ou System.Int64.Parse
chaîne -> double double.Parse(chaîne) ou System.Double.Parse(chaîne)
chaîne -> float float.Parse(chaîne) ou System.Float.Parse(chaîne)

La conversion d'une chaîne vers un nombre peut échouer si la chaîne ne représente pas un nombre valide. Il y a alors génération d'une erreur fatale appelée exception. Cette erreur peut être gérée par la clause try/catch suivante :

```
try    {
            appel de la fonction susceptible de générer l'exception
       } catch (Exception e)        {
                                    traiter l'exception e
                              }
 instruction suivante
```

Si la fonction ne génère pas d'exception, on passe alors à instruction suivante, sinon on passe dans le corps de la clause catch puis à instruction suivante.

Remarques :

Le cast **:** Le cast permet de faire une conversion explicite d'un type numérique vers un autre. Il s'agit de mettre avant l'expression le type destination entre deux parenthèses. Par exemple :

```
int a, b, c ;
float  moy ;
…
moy=(float)(a+b+c)/3
```

Boxing et Unboxing **:** Le boxing consiste à convertir un type de base vers une référence à un objet. Le Unboxing est une conversion inverse.
Exemple :

```
int i = 123;       // a value type
object o = i;      // boxing
int j = (int)o;  // unboxing
```

2.4 Les tableaux

Un tableau C# est un objet permettant de rassembler sous un même identificateur des données de même type. Sa déclaration est la suivante :

Type[] tableau=new Type[n]

n est le nombre de données que peut contenir le tableau.

La syntaxe *Tableau[i]* désigne la donnée n° *i* où i appartient à l'intervalle *[0,n-1]*.
Toute référence à la donnée *Tableau[i]* où *i* n'appartient pas à l'intervalle *[0,n-1]* provoquera une exception.

Un tableau peut être initialisé en même temps que déclaré :

```
int[] entiers=new int[] {0,10,20,30};
```

ou plus simplement :

```
int[] entiers={0,10,20,30};
```

Les tableaux ont une propriété **Length** qui est le nombre d'éléments du tableau.

Un tableau à deux dimensions pourra être déclaré comme suit :

Type[,] tableau=new Type[n,m];

Où *n* est le nombre de lignes, *m* le nombre de colonnes. La syntaxe *Tableau[i,j]* désigne l'élément j de la ligne i de t*ableau*.

Le tableau à deux dimensions peut lui aussi être initialisé en même temps qu'il est déclaré :

```
double[,] réels=new double[,] { {0.5, 1.7}, {8.4, -6}};
```

ou plus simplement :

```
double[,] réels={ {0.5, 1.7}, {8.4, -6}};
```

Le nombre d'éléments dans chacune des dimensions peut être obtenue par la méthode **GetLength(i)** où i=0 représente la dimension correspondant au 1er indice, i=1 la dimension correspondant au 2ième indice, …

Le nombre total de dimensions est obtenu avec la propriété **Rank**, le nombre total d'éléments avec la propriété **Length**.

Un tableau de tableaux est déclaré comme suit :

Type[][] tableau=new Type[n][];

La déclaration ci-dessus crée un tableau de *n* lignes. Chaque élément *tableau[i]* est une référence de tableau à une dimension. Ces références *tableau[i]* ne sont pas initialisées lors de la déclaration ci-dessus. Elles ont pour valeur la référence *null*.
L'exemple ci-dessous illustre la création d'un tableau de tableaux :

```
1. // un tableau de tableaux
2. string[][] noms = new string[3][];
3. for (int i = 0; i < noms.Length; i++) {
4.    noms[i] = new string[i + 1];
5. }//for
6. // initialisation
7. for (int i = 0; i < noms.Length; i++) {
8.   for (int j = 0; j < noms[i].Length; j++) {
9.       noms[i][j] = "nom" + i + j;
10.     }//for j
11.     }//for i
```

• ligne 2 : un tableau *noms* de 3 éléments de type *string[][]*. Chaque élément est un pointeur de tableau (une référence d'objet) dont les éléments sont de type *string[]*.

• lignes 3-5 : les 3 éléments du tableau *noms* sont initialisés. Chacun "pointe" désormais sur un tableau d'éléments de type *string[]*. *noms[i][j]* est l'élément *j* du tableau de type *string []* référencé par *noms[i]*.

• ligne 9 : initialisation de l'éélment *noms[i][j]* à l'intérieur d'une double boucle. Ici *noms[i]* est un tableau de i+1 éléments. Comme *noms[i]* est un tableau, *noms[i].Length* est son nombre d'éléments.

Voici un exemple regroupant les trois types de tableaux que nous venons de présenter :

```
12. using System;
13.
14. namespace Chap2 {
15. // tableaux
16.
17. using System;
18.
19. // classe de test
20. public class P02 {
21. public static void Main() {
22. // un tableau à 1 dimension initialisé
23. int[] entiers = new int[] { 0, 10, 20, 30 };
24. for (int i = 0; i < entiers.Length; i++) {
25.   Console.Out.WriteLine("entiers[{0}]={1}", i, entiers[i]);
26. }//for
27.
28. // un tableau à 2 dimensions initialisé
29. double[,] réels = new double[,] { { 0.5, 1.7 }, { 8.4, -6 } };
30. for (int i = 0; i < réels.GetLength(0); i++) {
31.    for (int j = 0; j < réels.GetLength(1); j++) {
32.         Console.Out.WriteLine("réels[{0},{1}]={2}",      i,      j,
réels[i, j]);
33.   }//for j
34. }//for i
35.
36. // un tableau de tableaux
37. string[][] noms = new string[3][];
38. for (int i = 0; i < noms.Length; i++) {
39.   noms[i] = new string[i + 1];
40. }//for
41. // initialisation
42. for (int i = 0; i < noms.Length; i++) {
43.   for (int j = 0; j < noms[i].Length; j++) {
44.         noms[i][j] = "nom" + i + j;
45.   }//for j
46. }//for i
47. // affichage
48. for (int i = 0; i < noms.Length; i++) {
49.    for (int j = 0; j < noms[i].Length; j++) {
50.         Console.Out.WriteLine("noms[{0}][{1}]={2}",      i,      j,
noms[i][j]);
51.   }//for j
52. }//for i
53. }//Main
54. }//class
55. }//namespace
```

A l'exécution, nous obtenons les résultats suivants :

1. *entiers[0]=0*
2. *entiers[1]=10*
3. *entiers[2]=20*
4. *entiers[3]=30*
5. *réels[0,0]=0,5*
6. *réels[0,1]=1,7*
7. *réels[1,0]=8,4*
8. *réels[1,1]=-6*
9. *noms[0][0]=nom00*
10. *noms[1][0]=nom10*
11. *noms[1][1]=nom11*
12. *noms[2][0]=nom20*
13. *noms[2][1]=nom21*
14. *noms[2][2]=nom22*

2.5 Les énumérations

Une énumération permet de définir un ensemble de constantes qui sont liées entre elles. La déclaration se fait de la manière suivante :

```
enum Semaine {lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche} ;
…
      Semaine jour ;
      jour=Semaine.mardi ;
```

Le compilateur associe à lundi la valeur 0, 1 à mardi, 2 à mercredi …

```
      if( jour==Semaine.jeudi)  ⇔  if((int)jour==3)

      jour=Semaine.mardi ;
      jour++ ;   //  jour devient mercredi
```

Il est possible d'affecter une valeur particulière à chaque membre de l'énumération :

```
      enum Couleur
      {
            Rouge=12,
            Bleu=46 ,
            Jaune=9
      } ;
```

2.6 Les structures

La syntaxe de déclaration d'une structure est la suivante :

```
struct Nom_typ {
public type1 Champ1;
public type2 Champ2;
public type3 Champ3;
…
public typeN ChampN;
}
```

Exemple :

```
struct Personne
{
      public string Nom ;
      public string Prenom ;
      public int Age ;
}
…

Personne Vperso;
Vperso=new Personne( );
Vperso.nom = "Ali" ;
Vperso.Age = 23;
```

2.7 Les opérateurs

2.7.1 Arithmétiques et de comparaison

Arithmétiques		
Opérateur	**Signification**	**Exemple**
+	Ajoute deux nombres	a + b
-	Soustrait deux nombres ou rend négatif un nombre	a - b ou -92
*	Multiplie deux nombres	a * b
/	Division	A/b
%	Reste de la division	A % B
Comparaison		
<	Inférieur	a < b (a plus petit que b)
<=	Inférieur ou égal	a <=
>	Supérieur	a > b
>=	Supérieur ou égal	a >= b
==	Egal	a = =b
!=	Différent	a != b

Pour une chaine + est l'opérateur de la concatenantion

2.7.2 Logiques

Opérateur	**Signification**
!	Opposé
&&	Et
||	Ou logique

2.7.3 Opérateurs abrégés

Opérateurs d'affectation :

Le langage C# offre la possibilité d'écrire des opérations avec une formule abrégée. La syntaxe est la suivante :

Variable Opérateur = Expression.

Ceci est équivalent à : *Variable = Variable Opérateur Expression*
Exemple :

A += 5 ; ⇔ *A = A + 5 ;*
*B *= 7 ;* ⇔ *B = B * 7 ;*

Opérateur d'incrémentation :

Pour incrémenter la valeur d'une variable on peut utiliser l'opérateur **++**. La syntaxe est la suivante : *Variable++* pour une incrémentation postfixée ou *++Variable* pour une incrémentation préfixée.
Exemple :

i++ ; ⇔ *i = i+1 ;*
++i ; ⇔ *i = i+1 ;*
x=y++ ; ⇔ *x=y ; y=y+1 ;*
x=++y ; ⇔ *y=y+1 ; x=y ;*

Opérateur de décrémentation :

Pour décrémenter la valeur d'une variable on peut utiliser l'opérateur **--**. La syntaxe est la suivante : *Variable--* pour une décrémentation postfixée ou *--Variable* pour une décrémentation préfixée.
Exemple :

i-- ; ⇔ *i = i-1 ;*
--i ; ⇔ *i = i-1 ;*
x=y-- ; ⇔ *x=y ; y=y-1 ;*
x=--y ; ⇔ *y=y-1 ; x=y ;*

C# présente d'autres structures, classes, méthodes et propriétés, (voir l'annexe ci-dessous).