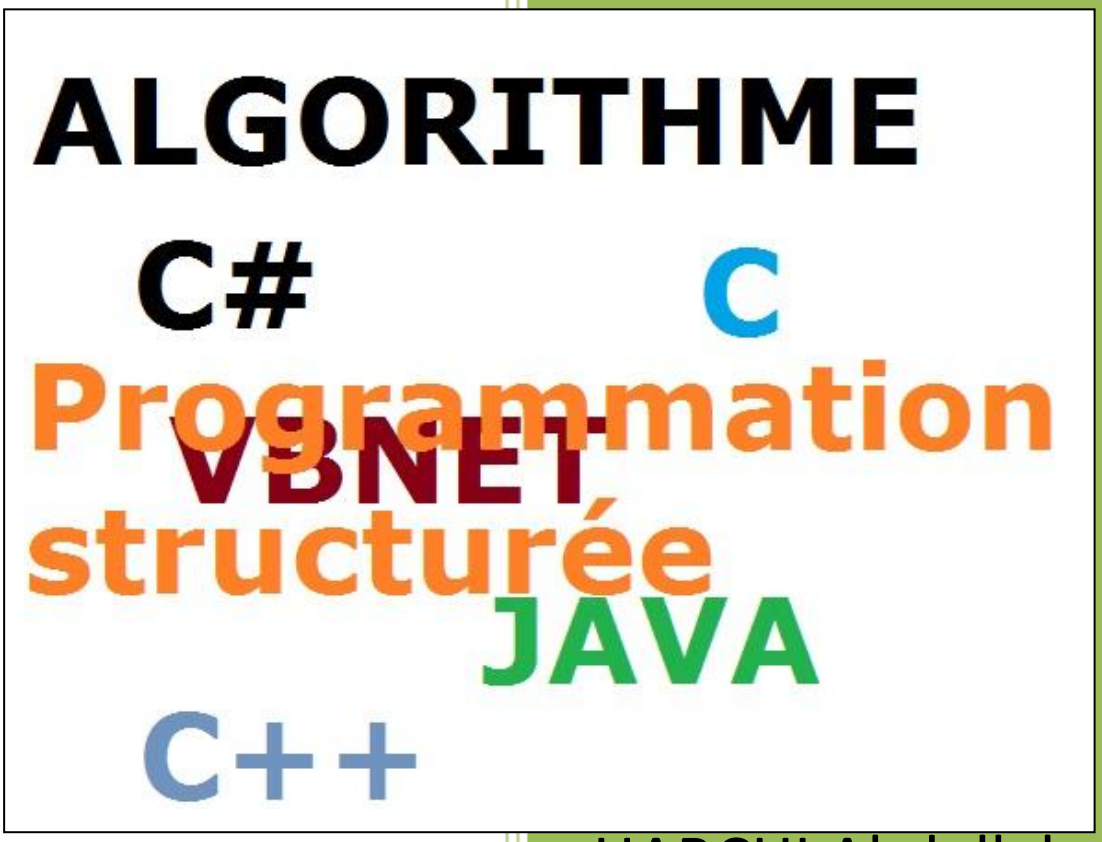
TDI 1ère Année

Eléments de base de C#

En mode Console



HARCHI Abdellah

Formateur NTIC

ISTA MAAMORA KENITRA

2011 - 2012

Table des matières

I- Ecrire la structure d'un programme C# en mode console 2

II- Déclarer et initialiser des variables de type simples (entier, réel, caractère) 3

III- Déclarer et utiliser des variables de type Chaînes de Caractères (String) ... 3

IV- Conversion d'une chaîne de caractère (String) en numérique et autres conversions 5

V- Tableau de conversion 6

Conversions étendues 6

Conversions restrictives 7

VI- Déclarer et manipuler des tableaux 7

VII- Utiliser les instructions arithmétiques 8

VIII- Utiliser Les opérateur LOGIQUES : (dans les conditions de test) 8

IX- Implanter les structures logique de bases (séquentielle , alternative , répétitive) 9

X- Les Entrées/Sorties : 10

XI- Utiliser une fonction ou une procédure 12

XII- Utiliser un espace de nom 13

I- Ecrire la structure d'un programme C# en mode console

```
//commentaires sur une ligne
/* commentaires sur plusieurs
lignes*/

using System;  // pour la définition de la console (bibliothèque System)

namespace Essai  // Nom du projet sur lequel on travail (espace de nom)

{
      // <summary>
      // Description Résumé de la classe Class
      //
      // </summary>
      class Class
      {
            // <summary>
            // Le point d'entrée principal de l'application.
            // </summary>
            static void Main()
            {
                  Console.WriteLine ("Bonjour tout le monde");
                  Console.ReadLine ();   //  pour interrompre le traitement
            }
      } /*fin bloc programme */
}
```

Deuxième version : sans using System

```
namespace Essai  // Nom du projet sur lequel on travail (espace de nom)

{
      class Class
    {
        /*debut bloc programme */
        static void Main ()
        {
              System.Console.WriteLine ("Bonjour tout le monde");
              System.Console.ReadLine();
        }
    }
}
```

Remarque:
La première ligne (avec la directive using) signale que l'on fera appel à des ressources (fonctions) regroupées dans un espace appelé **System**.

WriteLine provoque systématiquement un saut de ligne après l'affichage.

Le programme affiche « Bonjour tout le monde » est se termine aussitôt sans que l'on est le temps de lire le message; on ajoute alors une instruction dite d'attente en dernière instruction: Console.ReadLine();

II- Déclarer et initialiser des variables de type simples (entier, réel, caractère)

Préalable :

Le langage C# fournit un ensemble très complet de types de données. Lorsque vous utilisez ces derniers, il est bon de ne pas oublier deux faits concernant l'environnement C# :

- Toutes variables, qu'elles soient définies par l'utilisateur ou intrinsèques (intégrées), sont de première classe. Cela signifie qu'elles peuvent être utilisées comme objet n'importe où dans le système.
- Toutes variables du système sont automatiquement fixées à des valeurs par défaut par le système au moment de leur déclaration.

En C#, les variables sont répartie en deux structures simples :

- les types valeur : variables auxquelles on affecte directement une valeur.
- Les types référence : variables qui doivent être affectées des interfaces via des méthodes ou des fonctions, si on veut accéder à leur données internes.

Les types de données par valeur :

<Type> <NomVariable> [= <Constante>] ;

Type	Valeur
sbyte	Entier signé de 8 bits (1 octet) –128 à +127
short	Entier signé de 16 bits (2 octets) –32768+32767
int	Entier signé de 32 bits (4 octets)
Long	Entier signé de 64 bits (8 octets)
byte	Entier non signé de 8 bits 0 à 255
ushort	Entier non signé de 16 bits 0 à 65535
uint	Entier non signé de 32 bits
ulong	Entier non signé de 64 bits
float	Type en simple précision à virgule flotante 7 décimals
double	Type en double précision à virgule flotante 15 décimales
bool	Type booléen; une valeur **bool** est soit vraie (true) soit fausse (false)
char	Type caractère; une valeur **char** est un caractère Unicode (extension de ANSI sur 16 bits)
decimal	Indique un type décimal :16 octets:128 bits contenant 28 chiffres significatifs valeur réelle sous forme d'un entier multiplié par une puissance de dix (28 maxi)

Les types de données par référence :

C# fournit également des types qui sont, de part leur nature, passés par référence. Ce sont les classes et les structures, qui sont des types définis par l'utilisateur en C#.

III- Déclarer et utiliser des variables de type Chaînes de Caractères (String)

Le type de données String (chaîne de caractère) est une classe de **type référence** dans l'espace de noms **System** de .NET Framework. Donc une chaîne de type string est un objet qui n'est utilisable qu'à travers les méthodes de la classe String.

Etant donné que cette classe est très utilisée les variables de type string bénéficient d'un statut d'utilisation aussi souple que celui des autres **types élémentaires par valeurs**. On peut les considérer comme des listes de caractères numérotés de 0 à n-1 (si n figure le nombre de caractères de la chaîne).

Toutefois un objet string de C# est immuable (son contenu ne change pas), on peut accéder à chaque caractère de la chaîne en la considérant comme un tableau de caractères en lecture seule. Si l'on souhaite se servir d'une string comme un tableau de char, il faut utiliser la méthode **toCharArray** qui convertit la chaîne en un tableau de caractères contenant tous les caractères de la chaîne. Enfin le type string possède des méthodes d'**insertion**, **modification** et **suppression :** méthodes **Insert**, **Copy**, **Concat**,... Attention ces méthodes ne modifient pas la chaîne objet qui invoquent la méthode mais renvoient un autre objet de chaîne différent, obtenu après action de la méthode sur l'objet initial.

Déclaration d'une variable String :

string chaine ;

Déclaration d'une variable String avec initialisation :

string chaine = "abcdefghijk";

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 car = chaine [4] ; *// ici car contient la lettre 'e'*

Caractères de contrôle pouvant être intégrés dans une chaîne de caractères:

Le tableau suivant donne une liste des séquences d'échappement pour les caractères qu'il est possible de spécifier de cette manière :

Caractère	Séquence d'échappement
'	\'
"	\"
\	\\
Alerte	\a
Retour Arrière	\b
Saut de page	\f
Nouvelle ligne	\n
Retour chariot	\r
Tabulation horizontale	\t
Tabulation verticale	\v
Un caractère unicode spécifié par son numéro, par exemple \u200	\u
Un caractère unicode spécifié par son code hexadécimal, par exemple \xc8	\x
null	\0 (zéro)

Propriété principale :

Length : renvoie sous la forme d'un entier le nombre de caractères contenus dans une chaîne.

Exemple:

string s = "Salut";
Int lg = s.length;
Lg contient maintenant 5

Les fonctions principales (ou méthodes) spécialisées pour traiter un objet string sont :

IndexOf() :	Fonction d'exploration
Insert():	Fonction d'insertion
Compare() :	Fonction de comparaison
CompareTo():	Fonction de comparaison
Concat():	concaténation avec une autre chaîne
ToUpper():	Majuscule
Format():	Mise en forme de chaîne
StartsWith():	Fonction d'analyse
Substring():	Fonction d'extraction

Remarque : les variables string étant des objets, ils se manipulent avec la syntaxe suivante :

<nomObjet>.<NomMethode>(<NomParam-1>,<…>,….)
on obtient les méthodes dans l'EDI en tapant nomObjet.

Exemples :

```
public class ClassTest
{
public static void main (String[] args) throws Exception
{
        // manipulation de quatre chaînes
        string strP1 = "Boulons D45";
        string strP2 = "Vis chrome D45";
        string strP3, strP0 ;
        string strP4, strP5 ;
        // comparaison de deux chaîne donne I = 0;
        int i = strP1.CompareTo(strP2);
        // Recherche d'un caractère i =  8 ( ème position = D )
        int i=strP1.IndexOf('D');
        // Extraction à partir du 8ème caractère sur 3 de long
        strP3=strP1.Substring(8,3); // on obtient "D45"
        // Longueur de chaîne
        i= strP1.length;                // on obtient 11 la longueur
        // Mettre en majuscule
        strP3 = "bonjour" ;
        strP5=strP3.toUpperCase(); // strP5  majuscules, strP3 minuscules
        // Utilisation de la classe String
        int i = String.Compare(strP1, strP2); // I = 0
}
}
```

IV- Conversion d'une chaîne de caractère (String) en numérique et autres conversions

La classe **System.Convert** fournit un jeu complet de méthodes pour les conversions prises en charge. Elle constitue une façon, indépendante du langage, d'effectuer les conversions et est disponible pour tous les langages qui ciblent le Common Language Runtime (CLR). Alors

que divers langages peuvent recourir à différentes techniques pour la conversion de données, la classe **Convert** assure que toutes les conversions communes sont disponibles dans un format générique. Cette classe effectue des conversions restrictives, ainsi que des conversions dans des types de données dissociés.

Par exemple, les conversions des types **String** aux types numériques, des types **DateTime** aux types **String** et des types **String** aux types **Boolean** sont prises en charge. Pour une liste des conversions disponibles, consultez la liste de méthodes dans la classe **Convert**. La classe **Convert** effectue des conversions contrôlées et lève toujours une exception si la conversion n'est pas prise en charge. L'exception est souvent OverflowException. Pour une liste des conversions prises en charge, reportez-vous aux tableaux de conversion de types.

Vous pouvez passer la valeur que vous voulez convertir dans une des méthodes appropriées de la classe **Convert** et initialiser la valeur retournée dans une nouvelle variable.
Exemple 1 : le code suivant utilise la classe **Convert** pour transformer une valeur **String** en une valeur **numérique**.

```
string ch = "123,23";
double i = Convert.ToDouble(ch);
```

Affichage avec formatage

```
Console.WriteLine (Convert.ToString(d).Format(‘’# ##0,00‘’);
```

Exemple 2 : le code suivant utilise la classe **Convert** pour transformer une valeur **String** en une valeur **Boolean**.

```
string MyChaine = ‘’true’’;
bool MyBool = Convert.ToBoolean(MyChaine);

// MyBool prend la valeur vraie
```

V- Tableau de conversion

Une conversion étendue se produit lorsqu'une valeur d'un type est convertie en un autre type de taille égale ou supérieure. Une conversion restrictive se produit lorsqu'une valeur d'un type est convertie en une valeur d'un autre type de taille inférieure. Les tableaux de cette rubrique illustrent les comportements dont font preuve les deux types de conversion.

Conversions étendues

Le tableau suivant décrit les conversions étendues qui peuvent être effectuées en toute sécurité et sans perte d'information.

Type	Peut être converti en toute sécurité en
Byte	UInt16, Int16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal
SByte	Int16, Int32, Int64, Single, Double, Decimal
Int16	Int32, Int64, Single, Double, Decimal
UInt16	UInt32, Int32, UInt64, Int64, Single, Double, Decimal
Char	UInt16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal
Int32	Int64, Double, Decimal
UInt32	Int64, Double, Decimal
Int64	Decimal
UInt64	Decimal
Single	Double

Certaines conversions étendues en valeurs **Single** ou **Double** peuvent entraîner une perte de précision. Le tableau suivant décrit les conversions étendues qui peuvent parfois entraîner une certaine perte d'information.

Type	Peut être converti en
Int32	**Single**
UInt32	**Single**
Int64	**Single**, **Double**
UInt64	**Single**, **Double**
Decimal	**Single**, **Double**

Conversions restrictives

Une conversion restrictive en **Single** ou **Double** peut entraîner une perte d'information. Si le type cible ne peut pas exprimer correctement la valeur de la source, le type qui en résulte prend la valeur de la constante **PositiveInfinity** ou **NegativeInfinity**. La valeur de la constante **PositiveInfinity** est le résultat de la division d'un nombre positif par zéro et est retournée lorsque la valeur de type **Single** ou **Double** dépasse la valeur du champ **MaxValue**. La valeur de la constante **NegativeInfinity** est le résultat de la division d'un nombre négatif par zéro et est retournée lorsque la valeur de type **Single** ou **Double** est inférieure à la valeur du champ **MinValue**. Une conversion d'un **Double** vers un **Single** doit donner pour résultat **PositiveInfinity** ou **NegativeInfinity**.

Une conversion restrictive peut également entraîner une perte d'informations pour d'autres types de données. Cependant, si la valeur d'un type en cours de conversion tombe en dehors de la plage spécifiée par les champs **MaxValue** et **MinValue** du type cible et que la conversion est vérifiée par le runtime pour assurer que la valeur du type cible n'excède pas ses valeurs **MaxValue** ou **MinValue**, une exception OverflowException est levée. Les conversions effectuées avec la classe System.Convert sont toujours vérifiées de cette façon. Pour obtenir des informations sur l'exécution de conversions vérifiées sans l'utilisation de **System.Convert**, consultez la section consacrée à la conversion explicite.

Le tableau suivant reprend les conversions qui lèvent une exception **OverflowException** en utilisant **System.Convert** ou toute autre conversion contrôlée si la valeur du type converti se situe en dehors de la plage définie du type qui en résulte.

Type	Peut être converti en
Byte	Sbyte
SByte	Byte, UInt16, UInt32, UInt64
Int16	Byte, SByte, UInt16
UInt16	Byte, SByte, Int16
Int32	Byte, SByte, Int16, UInt16, UInt32
UInt32	Byte, SByte, Int16, UInt16, Int32
Int64	Byte, SByte, Int16, UInt16, Int32, UInt32, UInt64
UInt64	Byte, SByte, Int16, UInt16, Int32, UInt32, Int64
Decimal	Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64
Single	Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64
Double	Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64

VI- Déclarer et manipuler des tableaux

Tableaux à une dimension

Le type d'un tableau déclaré est donné tout d'abord par le type des éléments de base qu'il peut contenir et d'autre part par le nombre de ses dimensions. Les tableaux unidimensionnels ont une seule dimension (comme une ligne ou une file d'attente). Ils sont déclarés à l'aide de crochets, par exemple :

```
int[] i = new int[100];
```

Cette ligne de code déclare la variable i comme étant un tableau d'entiers de taille 100. Elle contient la place pour stocker 100 éléments entier, allant de i[0] à i[99].
Pour remplir un tableau, il est possible de spécifier la valeur de chaque élément comme dans le code suivant :

```
int i = new int[2];
i[0] = 1;
i[1] = 2;
```

Il est également possible d'effectuer simultanément la déclaration du tableau avec l'affectation de valeurs aux éléments en utilisant ;

```
int[] i = new int[] {1,2};
```

Ou même en utilisant une version abrégée de cette dernière :

```
int[] i = {1,2};
```

Par défaut tous les tableaux commencent par défaut avec 0 comme borne inférieure (nous vous conseillons de vous en tenir à la valeur par défaut). Néanmoins en utilisant la classe System.Array du .NET Framework il est possible de créer et manipuler des tableaux ayant une autre borne inférieure.

La propriété Length, en lecture seule, d'un tableau contient le nombre de ses éléments selon toutes ses dimensions. Comme un tableau unidimensionnel n'a qu'une seule dimension, cette propriété contient la longueur de la seule dimension. Ainsi, étant donnée la définition ci-dessus du tableau i, i.Length vaut 2.

Exemples :

```
int [] t = new int[5];                  // déclaration d'un tableau de 5 entiers
int[] tval = {0,200,300,400,500};       // déclaration + initialisation
tval[0] = tval[1]/2 ;                   // on obtient 100


char[]  tcar = {'a','b','c'};           // déclaration d'un tableau de caractères
for ( int i=0;i < 3;i++)
        Console.WriteLine(tcar[i]);     // listage des valeurs du tableau

string[] tNom = {"Toto","Julie"};       // tableau de chaînes de caractères.

for ( int i=0;i < 2;i++)
        Console.WriteLine(tNom[i]);// listage des chaînes
```

VII- Utiliser les instructions arithmétiques

+	Addition
-	Soustraction
*	multiplication
/	Division
( )	Parenthèses algébriques

VIII- Utiliser Les opérateur LOGIQUES : (dans les conditions de test)

Egalité	= =
Différent	!=
Inférieur ou égale	<=
Supérieur ou égale	>=

Et	&&
Ou	||
non	!

IX- Implanter les structures logique de bases (séquentielle , alternative , répétitive)

Séquentielle

Les instructions sont séparées par ;

Un bloc d'instruction est encadré par **{ }**

Alternative

```
if ( condition )
        {block 1}      ou   instruction-unique ;
    [else      //    [ ] = facultatif
        {block 2}      ou    instruction-unique ; ]
```

Exemple :

```
 if (delta > 0)
   {
        x1= -(b/2*a) + racine (delta) ;
        x2= -(b/2*a) - racine (delta) ;
   }
   else
        Console.WriteLine("1seule racine") ;
...
```

```
switch (variable)
    {
        case valeur-1 : {block instructions} ou instruction-unique ;
        break;
        case valeur-2 : [ { block instructions } ou instruction-unique ] ;
        break;
        default :       {block instructions} ou instruction-unique ;
    }
```

Exemple :

```
    char cCateg ;
                                        // Acquisition du code catégorie
    switch ( cCateg )
    {
        case 'c' : celibat(iMatric) ;            // fonction à programmer
        break;
        case 'm' : couple(imatricH,iMatricF);        // idem
        break;
        default : Console.WriteLine("erreur");
    }
```

Répétitives : 4 formules

Le TANT QUE :

```
while (condition) // 0-N : tant que condition vrai
{
        block instructions
}
ou instruction-unique ;
```

Exemple :

```
string strLigne = f.read();             // lecture premier enregistrement
while ( strLigne != null )              // tant que pas fin de fichier
 {
        Console.WriteLine(strLigne);          // impression ligne
        strLigne = f.read();                  // lecture enregistrement suivant
 }
```

Le JUSQU'A (faire tant que)

```
do
{
        block instructions  ou instruction-unique ;
 }
while (condition); // 1-N : Jusqu'a condition faux
```

Exemple :

```
string strLigne=f.read();                     // lecture premier
 {
        Console.WriteLine(strLigne);          // impression ligne
        strLigne=f.read();                    // lecture suivante
 }
while ( strLigne != null )                    // tant que pas fin de fichier
```

Le POUR :

```
for (expression-1;expression-2;expression-3)
{
        block instructions ou instruction-unique ;
}
        // N connu (utilisé pour les tableaux notamment)

         expression-1 : initialisation
         expression-2 : tant que condition vrai
         expression-3 : incrémentation
```

Exemple :

```
for(int I=0 ;I<10 ;I++)      // calcul totaux des 10 1er nb paires et impaires
{
        iTotPair=iTotPair+(2*I);
        iTotImpair=iTotImpair+(2*I)+1;
}
System.out.println("Paires " + iTotPair + " Impaires : " + iTotImpair );
```

X- Les Entrées/Sorties :

Afficher à l'écran :

Méthodes de la classe **Console** de **System**

Avec

using System ;

```
Console.Write(<Nomobjet>);              // pas de saut de ligne
Console.WriteLine(<Nomobjet>);          //  saut de ligne
```

Rem : N'importe quel type de variable peut être imprimée mais il est judicieux d'utiliser un type String (ou plusieurs concaténées avec +)

Exemple :

```
string strLigne = "Bonjour ";
Console.WriteLine (strBonjour  + "  tout le monde");
```

Lire au Clavier (saisie)

Utilisation de la méthode **ReadLine**() de la classe **Console** de **System**.
On lit toujours une chaîne de caractères que l'on convertit ensuite dans le bon type

Exemples :

// lecture d'une chaîne de caractères

```
string s1 = Console.ReadLine();
Console.WriteLine (s1);
```

// lecture d'une chaîne numérique entier et conversion

```
string s2 = Console.ReadLine();
int i = Convert.ToByte(s2);
i = i + 1;
Console.WriteLine (Convert.ToString(i));
```

// lecture d'une chaîne numérique entier et conversion

```
string s3 = Console.ReadLine();
```

// remarque : saisir au clavier avec une , (virgule) et avec un . (point) les chiffres décimaux.

```
double d = Convert.ToDouble(s3);
d = d + 0.50;
Console.WriteLine (Convert.ToString(d));
```

Avec formatage de la sortie

```
Console.WriteLine (Convert.ToString(d).Format(''# ##0,00'');
```

Imprimer :

Pas de méthode spéciale, il faut soit :

- Rediriger la sortie standard à l'exécution (java Prog > LPT1: en dos).
- Ecrire dans un fichier séquentiel à imprimer via le spool ultérieurement

XI- Utiliser une fonction ou une procédure

Une procédure :

D'un même programme s'appelle par son NOM suivi des PARAMETRES qu'on lui transmet. Elle exécute les instructions qu'elle contient puis redonne le contrôle à l'instruction suivante du programme appelant.

NomProc([param1,[param2]….]);

Exemple :
Afficher("Bonjour");

Une Fonction :

Est une procédure qui a un TYPE (donnée en retour). Elle s'exécute puis retourne une VALEUR (de son TYPE) avant de rendre le contrôle à l'instruction suivante du programme appelant.

Type NomVariable = NomFonction([param1,[param2]….]);

Exemple :
int a = LireClavier() ;

Remarque : Lorsque ces procédures ou fonctions appartiennent à un objet technique prédefini (String par exemple) on les nomme " méthodes " et on les utilise de la manière suivante :

<nomObjet>.<NomMethode>(<NomParam-1>,<…>,….)

Remarque : *on obtient les méthodes dans l'EDI en tapant nomObjet puis .*

Exemple :

Voir Objet String ; chaîne de caractères

Exemple d'utilisation d'une fonction :

L'exemple montre l'utilisation d'une fonction (Addition) devant faire la somme de deux nombres passés en paramètres.

La fonction est définie en dehors la structure principale du programme.

```
public class Calcul
    {
      // Structure Principale du Programme
          public static void Main ()
          {
              // Déclaration des Variables
                double dChiffre1 , dChiffre2 ,dSomme ;
                  // Entrées des Valeurs au Clavier
                  // Valeur 1
                Console.WriteLine(" Valeur 1 = ");
                dChiffre1 = Console.ReadLine();
                // Valeur 2
                Console.WriteLine (" Valeur 2 = ");
                dChiffre2 = Console.ReadLine();
                // Utilisation de la fonction Addition
                dSomme = Addition (dChiffre1,dChiffre2) ;
```

```
                // Affichage du résultat
            Console.WriteLine (dChiffre1 + " + " + dChiffre2 + " = " + dSomme);

      }

      // Structure de la FONCTION Addition
          public static double Addition (double dChiffreA, double dChiffreB)
          {
                double dTotal ;
                dTotal = dChiffreA + dChiffreB ;
                return dTotal ;
      }
}
```

XII- Utiliser un espace de nom

Même si vous n'en déclarez pas explicitement, un espace de noms par défaut est créé. Cet espace de noms sans nom, parfois appelé espace de noms global, est présent dans chaque fichier. Tout identificateur dans l'espace de noms global est disponible pour être utilisé dans un espace de noms nommé.
Les espaces de noms disposent implicitement d'un accès public et cela ne peut pas être changé.
Using <Nom> ;

Avant le mot clef Class (début de fichier source)