# Bases du langage C#

## I. C# en mode console (mode texte)

✍ Avantages par rapport au mode graphique (Application Windows):
- C'est un mode plus proche de l'approche algorithmique (pas de notions de composants, de propriétés, d'événements, …). Le programmeur débutant maîtrise un peu mieux ce qu'il fait.
- Permet d'écrire simplement des programmes qui ne permettent qu'une interaction minime voire nulle avec l'utilisateur.

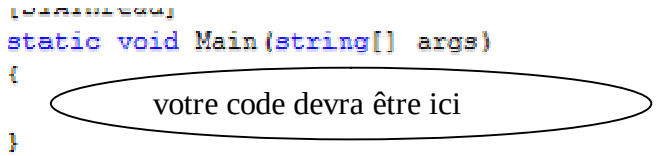
✍ Inconvénients :
- Ne reflète pas la réalité des applications de gestion actuelles (toutes avec interface graphique)
- Aspect plus rébarbatif (l'application se lance dans une fenêtre de commande)

✍ **Ecrire une application en mode Console**

Pour créer une nouvelle application en mode console, la seule différence avec la création d'une nouvelle application Windows c'est qu'il faut choisir comme type de modèle **Application Console** et non Application Windows dans la boîte de dialogue **Nouveau Projet.**

A l'écran apparaît alors le squelette du code de l'application. Evidemment, il n'y a pas de mode design pour créer une interface graphique.
Dans la fenêtre Explorateur de solutions, on remarque que plusieurs fichiers sont générés automatiquement. Tous ces fichiers sont indispensables, nous les étudierons par la suite, mais dans un premier dans vous n'aurez à travailler que sur un seul fichier, celui qui est ouvert par défaut, dans lequel vont être écrites les instructions. Ce fichier d'extension ".cs" (cs comme C sharp) est appelé par défaut Class1.cs. Ce type de fichier qui contient le code du programme s'appelle **fichier source**.

Les instructions de votre programme doivent être écrites à l'intérieur des accolades du bloc appelé `static void Main(string []args),` à la place des commentaires (les lignes commençant par //). Nous verrons par la suite que nous pourrons placer du code en dehors de Main (en créant d'autres blocs de code) ou même créer d'autres fichiers source concernant la même application.



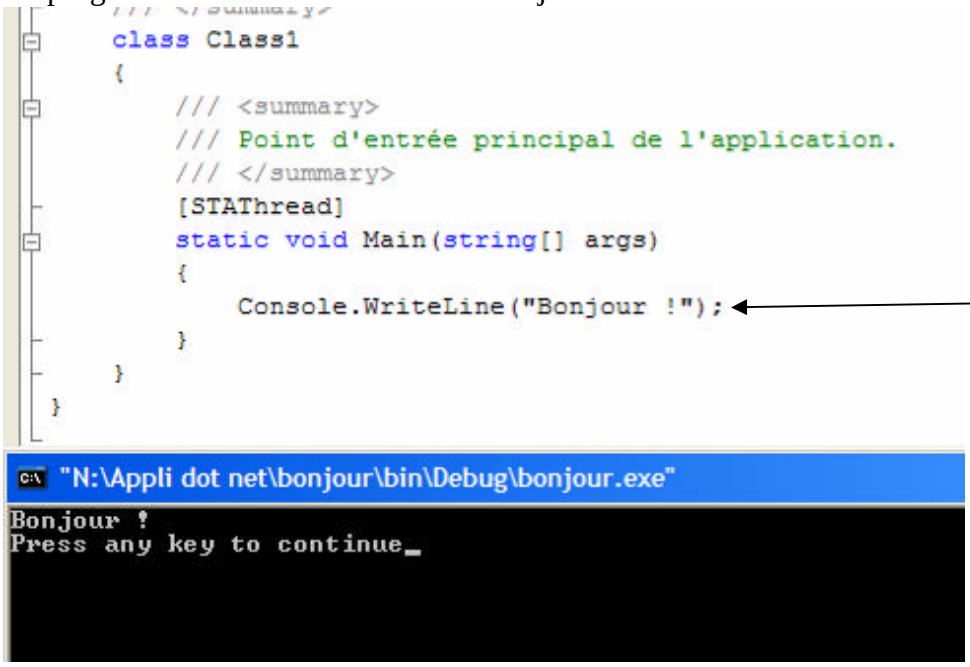

✍ *Tant que vous êtes novices en programmation et que vous ne maîtrisez pas les instructions qui sont générées automatiquement par l'environnement de développement, ne touchez pas à ces instructions, surtout ne modifiez et ne supprimez rien, votre application risquerait alors de ne plus fonctionner. Ceci est valable dans n'importe quel mode (console ou graphique).*

Pour exécuter un programme écrit en mode console, il faut d'abord le **générer**, ce qui correspond à la traduction des instructions écrites en C# en instructions compréhensibles par la machine, ce qu'on appelle aussi la **compilation**.

Une fois la génération réussie (pas d'erreur de syntaxe), il faut lancer l'**exécution** du programme par la commande **Exécuter sans débogage** (dans le menu déboguer ou par l'icône représentant un ! rouge, ou par Ctrl + F5).
✍ *Attention, si vous choisissez à la place la commande Démarrer (flèche ou F5) comme vous pouviez le faire avec les applications Windows, alors l'application va s'exécuter mais la fenêtre se refermera immédiatement après, et on n'aura pas le temps de voir le résultat.*

### Exemple de programme basique en mode console

Ce programme ce contente d'afficher Bonjour dans la fenêtre d'exécution.




code source du programme

seule instruction ajoutée, qui affiche Bonjour !

fenêtre d'exécution

#### ✍ L'affichage et la saisie en mode console

Pour l'affichage et la saisie, on utilise un composant invisible (une classe) qui s'appelle Console, et qui possède des méthodes pour afficher du texte et saisir des chaînes de caractère.

### L'affichage

Comme il n'y a pas d'interface graphique avec des contrôles pour écrire du texte, ce sont des instructions particulières qui permettent de l'affichage lors de l'exécution, en mode texte dans une fenêtre de commande.

```
Console.WriteLine( expression à afficher ) ;
```

Cette instruction affiche la valeur de l'expression indiquée entre parenthèses puis passe à la ligne.

```
Console.Write( expression à afficher );
```

Cette instruction fait la même chose mais sans passer à la ligne. L'affichage suivant se fera donc juste après ce qui est écrit à l'écran (sans même un espace).

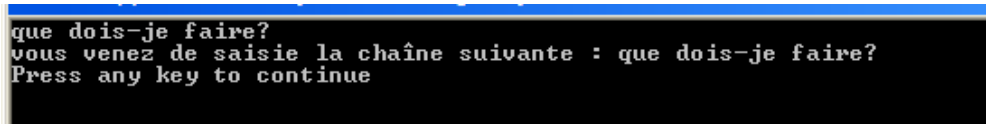
L'avantage des méthodes Write et WriteLine c'est qu'elles permettent d'afficher n'importe quel type d'expression, et pas seulement les expressions de type chaîne de caractère (contrairement au champ Text des contrôles graphiques). Aucune conversion n'est nécessaire !

## La saisie

La saisie est plus délicate que l'affichage.
Pour récupérer ce que l'utilisateur saisi au clavier, il faut d'abord déclarer une variable de type string (chaîne de caractère) qui va permettre de mémoriser la saisie.
A cette variable, on affecte le résultat de la méthode **Console.ReadLine( )**, qui est toujours une chaîne de caractère (même si l'utilisateur a tapé un nombre). Le résultat de cette méthode est tout simplement ce que l'utilisateur a tapé au clavier avant de valider par la touche Entrée.
ex :

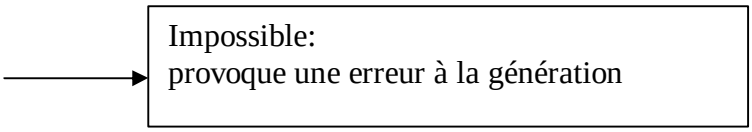
```
string chainesaisie ;
chainesaisie = Console.ReadLine();
Console.WriteLine("vous venez de saisir la chaîne suivante : "+ chainesaisie);
```

Résultat :




Si l'utilisateur saisit un nombre, celui-ci sera tout de même considéré comme une suite de caractère, et donc comme une chaîne (string). Autrement dit, il n'est pas possible de saisir directement une variable d'un autre type que string.

ex :
int unentier;
unentier = Console.ReadLine( ) ;




Pour pouvoir récupérer la valeur saisie dans une variable de type approprié, il faut effectuer une **conversion** par le composant (la classe) Convert et la méthode correspondante au type voulu. Plus exactement, on affecte à la variable du bon type, le résultat de la conversion de la chaîne saisie par Convert.To…

Exemple : saisie d'un entier et affichage de son carré

```
string chnb;                    //Déclarations
int nb;

Console.WriteLine("tapez un nombre entier ");
chnb = Console.ReadLine();                  //Saisie dans une variable de type string

nb = Convert.ToInt32(chnb);    //Conversion de cette variable en type int

nb = nb*nb;         //Calcul du carré

Console.Write("voilà votre nombre au carré :");         //Affichage du résultat
Console.WriteLine(nb);
```

Les principales méthodes de conversion sont les suivantes :
**ToString** pour convertir une expression numérique en **chaîne**.

Inutile de convertir une chaîne dans une autre chaîne !!
**ToDouble** pour convertir une chaîne ou un entier en **réel** (double).
**ToInt32** pour convertir une chaîne ou un réel en **entier**.
Si la chaîne ne correspond pas à un nombre entier, il y aura erreur à l'exécution.
Convertir un réel (double) en entier enlève la partie décimale (après la virgule).

# II. Les variables en C#

### Rappel d'algorithmique

Les données d'un programme doivent être stockées dans la mémoire vive de l'ordinateur afin d'être traitées ou comparées. Les différents espaces mémoires utilisables par le programmeur sont appelés **variables,** nommées ainsi puisqu'elles stockent des valeurs qui peuvent varier selon le cours d'exécution du programme.
Le contenu d'une variable peut donc changer au cours de l'exécution du code. Cependant, une variable ne peut contenir qu'une seule donnée à la fois.

### Déclaration d'une variable

Le programmeur doit procéder à la **déclaration de la variable** avant de pouvoir l'utiliser.
La déclaration d'une variable indique son nom et son type.
Le type d'une variable spécifie 2 choses :

- la nature de l'information qui sera contenue au sein de cette variable, ce qui détermine les **opérations possibles** sur celle-ci
- et la **taille de l'espace mémoire** nécessaire pour contenir cette information.

#### Les principaux types primitifs

Les principaux types de base de C# sont :

| Type de données | Equivalent en Algo | Description |
|---|---|---|
| byte | entier | Pour les petits entiers de 0 à 255<br>Représente un entier non signé sur 8 bits. |
| int | entier | Pour les entiers signés de -200 milliards à 200 milliards environ.<br>Représente un entier signé sur 32 bits. |
| float | réel | Représente un nombre à virgule flottante sur 32 bits (simple précision). |
| double | réel | Représente un nombre à virgule flottante sur 64 bits (double précision). |
| decimal | réel | Pour les nombres très grands, par exemple les valeurs monétaires.<br>Représente une valeur décimale sur 96 bits. |
| char | caractère | Représente un caractère Unicode sur 16 bits. |
| string | chaîne | représente une chaîne de caractères immuable à longueur fixe de caractères Unicode. |
| bool | booléen | représente une valeur booléenne *true* ou *false*. |

#### Règles concernant le nom des variables

Le nom d'une variable déclarée en C# doit respecter les critères suivants afin d'être valable :

• Le premier caractère est obligatoirement une lettre ou le caractère de soulignement "_"
• Le nom peut être constitué de lettres, chiffres et du caractère de soulignement "_".
• Une distinction est faite entre les majuscules et les minuscules (on dit que C# est sensible à la casse). TOTO, Toto et toto seront donc des noms de variables distincts.
• Les accents sont tolérés par C# mais non conseillés par les bonnes techniques de programmation.
• Les mots réservés du langage C# ne peuvent pas être utilisés comme noms de variables (int, for, if, …). On se rend compte qu'on utilise un mot réservé du langage car celui-ci apparaît d'une couleur différente des autres variables.

## Syntaxe de la déclaration

| En algorithmique | En C# |
|---|---|
| *nomvariable : type* | *type nomvariable ;* |
| ex :<br>Var<br>x : entier<br>bidule : chaîne | <br><br>int x;<br>string bidule; |
| Les variables sont déclarées au début après le mot clé Var | Les variables peuvent être déclarées n'importe où dans le code |

## Conversion de type

Les fonctions de conversions permettent de transformer une variable d'un type donné dans un autre type. Cela est en particulier utile pour manipuler des valeurs numériques saisies dans des contrôles de texte.
En effet, le texte des contrôle est toujours du type string, même si l'utilisateur y saisi un nombre. Et de la même manière, une variable affectée à la propriété Text d'un contrôle doit être de type string.

Pour convertir toute variable en string on utilise la fonction suivante :

**Convert.ToString(*variable à convertir*);**

De la même manière, on peut convertir une variable dans n'import quel autre type.
Par exemple :
conversion en entier : **Convert.ToInt32**(*variable à convertir*);
conversion en réel : **Convert.ToDouble**(*variable à convertir*);

Si la variable ne peut être convertie une erreur est renvoyée. Par exemple, si l'utilisateur tape du texte alphabétique dans une zone de saisie et que ce texte doit être converti en entier, le programme s'arrête en affichant une boite de dialogue d'erreur.

---

Exemple complet :

Supposons que nous ayons une zone de saisie appelée txtBox et une étiquette lblResultat

```
int nombreEntier;

//conversion du texte de la zone de saisie pour l'affecter à un nombre entier
nombreEntier = Convert.ToInt32(txtBox.Text) ;

//conversion du nombre en chaîne pour l'écrire dans le Label
lblResultat.Text = Convert.ToString(nombreEntier) ;
```