C#/Asp.Net – Cours 1

# Configuration de Visual Studio

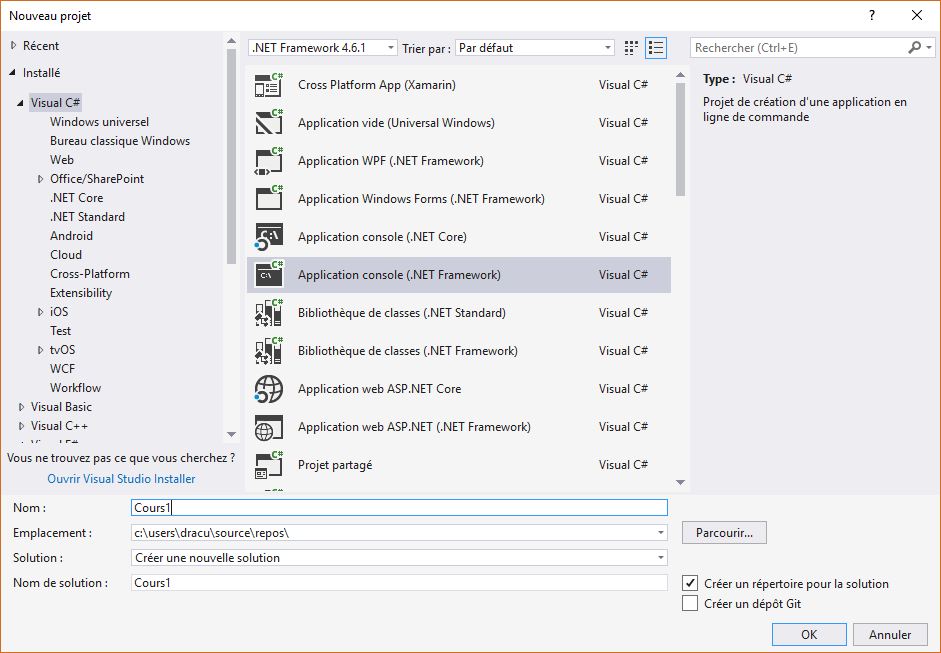
* Lancer Visual Studio 2015/2017
* Configurer Visual Studio avec le paramétrage "**Général**"

Visual Studio est extrêmement personnalisable, et peut-être complexe à manipuler. Pour ce cours, nous souhaitons tous utiliser un environnement proche dans son fonctionnement et sa manipulation. Nous allons donc initialiser son paramétrage sur un modèle déjà existant. Le choix de ce paramétrage vous est demandé au premier démarrage. Il faudra choisir l'option "**Général**".

Si cela n'était pas le cas, ou si un autre choix a déjà été fait par le passé, il est possible de le réinitialiser via le menu "Outils" puis "Importation et exportations de paramètres", puis l'option "Réinitialiser tous les paramètres".

*Pour information, ce choix n'a pas d'impact limitant les fonctionnalités de Visual Studio. Quel que soit le choix effectué, vous aurez accès aux mêmes possibilités. Il s'agit juste d'un ensemble de dispositions de fenêtres par défaut, de raccourcis clavier, et de proposition de types de projets qui sont modifiées*

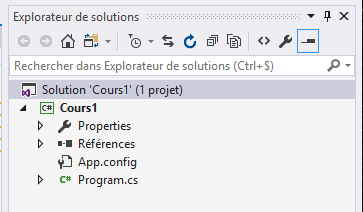
* Créer une nouvelle solution de type "Application console (.NET Framework)" en C#. Pour ce faire, rendez-vous dans le menu "Fichier", puis "Nouveau", puis "Projet". Puis dans la fenêtre qui s'ouvre, veillez à bien sélectionner "Visual C#" à droite, et "Application console (.NET Framework)" dans la partie centrale. Donnez le nom que vous souhaitez au projet (peu importe pour le moment).



* Une fois la solution créée, vous pouvez continuer avec la partie II.

# Organisation des fichiers

Avant toute chose, nous allons étudier ce que Visual Studio a créé pour nous. Tout commence dans l'écran "Explorateur de solution", qui est par défaut positionné à droite dans Visual Studio.



Cet écran permet de visualiser la hiérarchie des projets de la solution courante. La capture plus haut représente donc :

* Une solution appelée "Cours1" (l'élément sélectionné dans la liste), qui contient…
* Un projet C# appelé "Cours1" lui aussi

*Pour info, si par défaut Visual Studio créé à la fois une solution et un projet avec le même nom, ce n'est qu'une convention, rien n'oblige à conserver cela. D'ailleurs, une solution a pour vocation de référencer plusieurs projets comme nous le verrons plus tard.*

* Le projet "Cours1" contient 2 "dossiers" un peu spéciaux "Properties" et "Références" (que nous analyserons plus tard - pas important pour le moment), ainsi que deux fichiers : "App.config" et "Program.cs"
  + Le premier est peu important pour le moment, c'est un fichier de configuration qui peut contenir différents paramétrages (par exemple, connexion à une base de données, sécurité, etc.)
  + Le second est le point d'entrée de l'application, et est ouvert par défaut si vous venez de créer un nouveau projet et c'est celui sur lequel nous allons travailler aujourd'hui.
* Si nous analysons le répertoire qui a été créé sur le disque dur (en faisant un bouton droit sur la **solution**, puis "Ouvrir le dossier dans l'explorateur de fichier"), on peut voir que :
  + Le répertoire racine de la solution contient un fichier "NomDeMaSolution.sln", ainsi qu'un répertoire "NomDeMonProjet".
  + Le fichier ".sln" est ce qui définit la solution, et référencie (entre autres) un projet ".csproj" présent dans le sous-répertoire portant son nom (vous pouvez ouvrir le fichier sln avec notepad++ par exemple)
  + Le répertoire portant le nom du projet contient les fichiers "App.config", "Program.cs", mais aussi un fichier XML appelé "NomDeMonProjet.csproj", ainsi que trois répertoires : "bin", "obj" et "Properties"
  + Le répertoire "bin" ne contient aucun fichier, seulement un sous-répertoire "debug"
* Dans Visual Studio, nous allons générer le projet courant, via le menu "Générer" puis "Générer la solution"
* Dans la fenêtre "sortie" en bas, il devrait apparaitre le message suivant : "======= Génération : 1 a réussi, 0 a échoué …". Cette opération s'appelle "la compilation".
* *Notez dans un coin de votre tête que si ce message n'apparaissait pas (et cela sera le cas durant vos développements), il ne sert à rien de continuer avant d'avoir résolu les soucis qui empêchent la compilation. Il ne faut surtout pas accepter la proposition de Visual Studio qui vous proposera de lancer tout de même la dernière version qui a fonctionné, cela n'a aucun intérêt.*
* Dans le répertoire "NomDeMonProjet\bin\debug", un fichier "nomdemonprojet.exe" (".exe" = exécutable) est apparu. Double-cliquez dessus, une fenêtre s'ouvre puis se ferme immédiatement. C'est normal, votre programme ne faisant rien actuellement, il se quitte de suite.
* Dans "Program.cs", entre les 2 accolades centrales (fonction "Main"), ajoutez les deux lignes suivantes :

Console.WriteLine("Bonjour !");

Console.ReadKey();

* Votre code devrait donc ressembler à cela :

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Bonjour !");

Console.ReadKey();

}

}

* Recompilez votre programme, puis lancez le manuellement, il devrait maintenant vous afficher un message puis disparaitre au premier appui de touche
* Il est possible de compiler puis de lancer directement le programme sans devoir le faire à la main via le menu "Déboguer" puis "Démarrer le déboguage" (touche **F5**). C'est cela que nous ferons dorénavant.

Si vous avez terminé, ne pas faire le "III" de suite.

# C# - types

## Variables - déclaration et affectation

* Toujours dans le projet précédent, pour chaque type usuel, déclarez et affectez une variable avec des noms et des valeurs de votre choix
* Puis créez une énumération "JourDeLaSemaine" contenant tous les jours de la semaine
  + Cette énumération doit être créée **en dehors** de la classe "Program" (cf. slides)
* Puis dans la méthode Main, ajoutez une déclaration et affectation d'une variable de type "JourDeLaSemaine" que vous venez de créer
* Tentez d'affecter à des variables des valeurs d'un autre type, et notez que la plupart ne sont pas possibles (erreur de compilation, surlignage en rouge), mais certaines le reste : il est par exemple possible d'assigner une valeur entière à une variable d'un nombre à virgule flottante, mais pas l'inverse
  + *C'est ce qu'on appelle une conversion implicite. C# en possède un petit nombre, uniquement les conversions n'ayant pas de perte de précision. Il est possible de créer ses propres conversions si nécessaire*
* Utilisez la méthode "Console.WriteLine" en lui fournissant entre les parenthèses une variable pour l'afficher dans la console.
* Utilisez les opérateurs numériques classiques ("+", "-", "/" etc.), ainsi que les parenthèses pour gérer les priorités. Essayez d'utiliser ces opérateurs entre différents types.
* Petite énigme, aux vues de ce que vous venez de tester, et sans le saisir pour en connaitre la réponse, quelle serait selon le vous le résultat de l'affichage de la variable "maChaine" si je la déclare et l'affecte comme cela :

var maChaine = "Début " + 1 + 2;

Et comme cela ?

var maChaine = "Début " + (1 + 2);

Pourquoi ?

## Calculatrice - Niveau I

Nous allons maintenant commencer un programme que l'on reprendra petit à petit au cours des exercices. Nous souhaitons pouvoir disposer d'une calculatrice pour résoudre des opérations mathématiques simples (addition, soustraction, multiplication, division, puissance...). Pour se simplifier la vie, pour débuter :

* on ne s'intéressera qu'aux nombres entiers
* on ne gèrera pas de suite les éventuelles erreurs de saisies clavier.
* on ne fera que des calculs avec 2 opérandes et un opérateur (a + b, a - b, ab etc.)

Pour débuter par quelque chose de simple, réalisez un petit programme qui :

* Indique à l'utilisateur de saisir un premier nombre (la première opérande)
  + *Via l'utilisation de Console.WriteLine qui prends en paramètre la chaine de caractère à afficher*
* Capture sa saisie clavier et affecte le texte saisi dans une 1ère variable
  + *Via l'utilisation de Console.ReadLine qui attends que l'utilisateur ait appuyé sur "entrée" et renvoit le texte saisi.*
* En fait de même pour la seconde opérande affectée à une autre variable
* Converti chacune des saisies utilisateurs dans une variable de type entier ("int")
  + *Via l'utilisation de Int32.Parse qui prends en paramètre une chaine de caractère (par exemple, celle saisi) et renvoi une nouvelle valeur sous la forme d'un entier.*
* Affiche le résultat de l'addition, soustraction, multiplication, et division des deux opérandes.

Le pseudo code est donc de la sorte

soit une première variable "operandeGauche" de type chaine de caractère, et une seconde variable "operandeDroite" de type chaine de caractère

Afficher("Veuillez entrer l'opérande de gauche")

Affecter à la variable "operandeGauche" la valeur saisie par l'utilisateur

Afficher("Veuillez entrer l'opérande de droite")

Affecter à la variable "operandeDroite" la valeur saisie par l'utilisateur

Soit une variable "valeurOperandeGauche" de type entier, et une variable "valeurOperandeDroite" de type entier

Affecter à la variable "valeurOperandeGauche" la conversion de la chaine de caractère "operandeGauche"

Affecter à la variable "valeurOperandeDroite" la conversion de la chaine de caractère "valeurOperandeDroite"

calculer (valeurOperandeGauche + valeurOperandeDroite) et l'afficher

calculer (valeurOperandeGauche - valeurOperandeDroite) et l'afficher

calculer (valeurOperandeGauche \* valeurOperandeDroite) et l'afficher

calculer (valeurOperandeGauche / valeurOperandeDroite) et l'afficher

calculer (valeurOperandeGauche valeurOperandeDroite ) et l'afficher

# Instructions conditionnelles et opérations booléennes

Dans l'exercice précédent, rajouter différents tests pour gérer certains cas atypiques, notamment :

* Si la valeur de l'opérande de droite vaut 0, on ne tente pas d'effectuer la division mais on affiche un message à l'utilisateur
* Si la valeur de l'opérande de droite est inférieure à 0, on n'effectue par la mise en puissance
* Si la valeur de droite ainsi que la valeur de gauche sont égales à 0, on n'effectue aucun calcul

# Gestion de l’opérateur

Puis, modifiez votre code pour demander à l'utilisateur de saisir l'opérateur à utiliser (ex. "+", "-", "/", "^", "\*"), et n'effectuez que celui qui correspond à son choix, ou ne faites rien si l'opérateur n'est pas reconnu.

# Méthodes

Nous allons procéder à la modification du code source existant pour le clarifier en mettant en place un certain nombre de méthodes.

Le 1er but du développement de méthodes est de clarifier au maximum le code en capturant un comportement technique dans une méthode représentant une « intention ». C’est ce que nous allons faire :

Créer une méthode « DemanderOperandeALUtilisateur » qui :

* Prends en paramètre le message à afficher
* Affiche ce message, puis capture la saisie utilisateur
* Renvoie cette capture

Puis utilisez cette méthode pour demander les 2 opérandes

Le code obtenu à la fin devrait être beaucoup plus lisible. En bonus, il est beaucoup plus maintenable (pas de répétition de code).

Dans un second temps, nous allons aussi extraire le code qui analyse la saisie de l’opérateur par l’utilisateur.

* Rajoutez donc une énumération qui contient les différents opérateurs connus (Addition, Soustraction, Multiplication, etc. ainsi que le choix « OperateurNonReconnu »)
* Créez une méthode « DemanderOperateurALUtilisateur » qui :
  + Demande à l’utilisateur de saisir un opérateur
  + Capture sa saisie
  + Analyse la saisie et renvoie l’opérateur correspondant
* Puis remplacez le code correspondant dans le corps de la méthode Main, et utilisez une instruction switch sur le résultat de l’appel à « DemanderOperateurALUtilisateur » pour exécuter la bonne opération.

Y-a-t’il encore du code en doublon ? Si oui et si c’est pertinent, remplacez-le par une méthode appropriée.