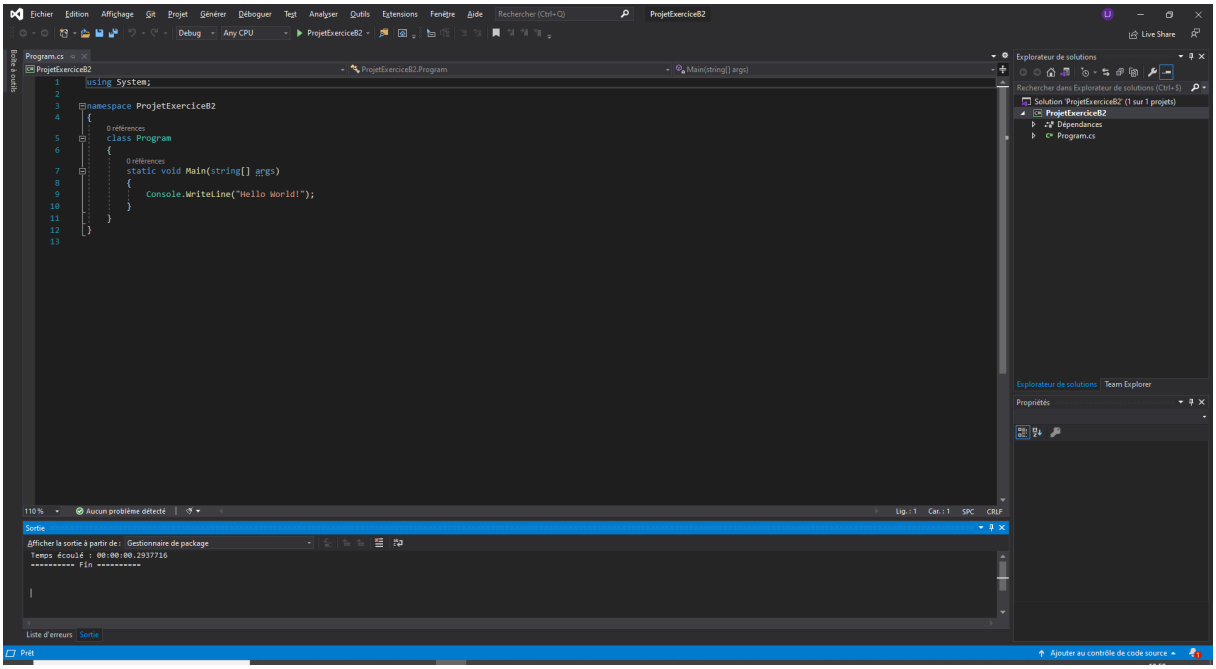
LANGAGE C# - Les Fondamentaux

Configuration d'un projet

Modèle à choisir : Application Console pour .NET Core
Choisir un nom pour votre projet, un emplacement puis valider.

Etat de Visual Studio après le lancement de votre projet :



Exercice 1 : Créer un nouveau projet sur Visual Studio 2019

Nous avons ici une solution qui contient le projet ProjetExerciceB2, qui est un projet vide.
Dans le fichier Program.cs, nous avons 3 parties très importante :

- Le Namespace : Il s'agit de l'ensemble du projet. Tout fichier de code qui contient le Namespace peut communiquer avec les autres fichiers du Namespace.
- La Classe Program : En C#, tout est « Objet » (tout comme en JAVA). Ici, notre classe est un peu spéciale car elle contient une fonction particulière, la fonction Main. Nous reparlerons des classes un peu plus tard dans le cours
- La Fonction Main : Cette fonction est OBLIGATOIRE dans un programme C#. C'est le point d'entrée de votre application. Dans la 1ière partie, nous allons exclusivement coder dans cette fonction, chose qui sera quasiment interdit dans la 2ième partie.

Rappel : Il ne peut avoir qu'une seule fonction MAIN par PROJET.

Déclaration de variables et de constantes

```
Type nomVariable = valeur;

const Type nomVariable = valeurConstante;
```

Le C# est un langage dit « typé ». Chaque variable aura un type qui lui est propre et ne pourra pas interagir avec des types différents. Par exemple, une variable de type Integer ne pourra pas interagir avec une variable de type String.

Rappel IMPORTANT : Une déclaration de variable en C# se termine TOUJOURS par un point-virgule !!!

Les Types

Voici une liste non-exhaustive de type dit « primitifs » :

- bool : Permet de manipuler un booléen (0 ou 1)
- byte : Permet de manipuler un octet (8 bits)
- char : Permet de manipuler un caractère unique
- int : Permet de manipuler un nombre entier (codé sur 32 bits)
- float : Permet de manipuler un nombre à virgule flottante (codé sur 32 bits)
- double : Permet de manipuler un nombre à virgule flottante (codé sur 64 bits)
- long : Permet de manipuler un nombre entier (codé sur 64 bits)
- String : Permet de manipuler une chaîne de caractères.

Exercice 2 : Manipulation de variables

Instancier deux variables de type int et les ajouter entre eux. Le résultat sera dans une nouvelle variable de type int.
Instancier une variable de type float et retrancher le résultat de type int de l'opération précédente et affecter la valeur à ce même int. Que constatez-vous ?

Instancier une variable de type string, puis ajouter lui la chaine de caractères suivante : « et ceci est concaténé ».

Afficher la variable string dans la console. *(La ligne de code est Console.WriteLine(nomdevotrevariable))*

LANGAGE C# - Les Fondamentaux

Les inputs et les outputs

```
Console.WriteLine("Hello World!"); //Affiche Hello World dans la console
String input = Console.ReadLine(); //On attend une entrée clavier de la part de
l'utilisateur
```

Comme déjà aborder dans l'exercice précédent, nous retrouvons ici la ligne de code suivante : Console.WriteLine

Cette fonction prend en paramètre une variable de type String. Ici, le fait d'écrire littéralement la phrase Hello World entre guillemets (qui permet de déclarer une chaine de caractères) est intrinsèquement considéré comme une variable de type String.

Nous avons ensuite une autre fonction : Console.ReadLine()

Cette fonction, à la différence de celle d'écriture, ne prend pas de paramètre en entrée. Dans l'exécution d'une application console, le programme se fige à cette ligne tant que l'utilisateur n'a pas validé une instruction dans le prompteur (ou terminal).
Cette fonction retourne une valeur de type string qui peut être contenue dans une variable, ici input.

Définition d'une fonction

```
public int MaFonction()
{
    return 1;
}

public void MaProcedure()
{
    Console.WriteLine("Hello World!");
}

public int MaFonctionAvecPassageDeParamètres(int a)
{
    return a + 1;
}
```

Précédemment, nous avons parlé de fonctions. Les fonctions que nous avons utilisées sont des fonctions prédéfinies par la bibliothèque de classes .NET.
Nous pouvons écrire aussi nos propres fonctions.

Pour déclarer une fonction, il faut définir les éléments suivants :

- La visibilité de la fonction : La fonction peut être public, private ou protected. Ceci n'est pas obligatoire pour l'instant, nous verrons plus tard lorsque nous aborderons les classes.

- Le type de retour de la fonction : Une fonction retourne soit « void » ou un type bien défini. Une fonction qui retourne « void » est une *procédure*.
- Le nom de la fonction : C'est grâce à cela que l'on va pouvoir appeler la fonction.
- Des paramètres : Ils sont définis dans les parenthèses. Il peut y avoir X paramètres, séparés d'une virgule. La définition d'un paramètre est le suivant : Type NomDuParamètre
- Une accolade ouvrante et fermante : On mettra entre les accolades le code que l'on souhaite exécuter lors de l'appel de la fonction

Appel d'une fonction

```
int a = MaFonction();

MaProcedure();

int b = MaFonctionAvecPassageDeParamètres(1);
```

Pour appeler une fonction, il suffit d'écrire le nom de la fonction et d'y ajouter les paramètres.
Comme pour une variable, l'instruction doit être suivie d'un point-virgule.
Le retour d'une procédure n'a pas besoin d'être récupéré dans une variable, à la différence d'une fonction.

Exercice 3 : Déclarations et appels de fonctions

Ecrire une procédure sans passage de paramètres qui affiche un texte.
Ecrire une procédure avec passage d'un paramètre de type String qui affiche un texte.
Ecrire une fonction renvoyant une valeur de type float qui fait la division euclidienne de deux paramètres de type int et qui retourne le reste de la division. Afficher ensuite le résultat. (Attention à la division par zéro !).
Ecrire une fonction avec un paramètre de type String qui retourne le pluriel du paramètre (Veuillez gérer les exceptions !) puis une autre fonction avec deux paramètres de type String qui affichera la phrase suivante : « Le pluriel de -1er paramètre- est -2^{ième} paramètre- »

Les conditions et les boucles

Nous allons maintenant voir les conditions et les boucles en C#. Voici nos possibilités :

IF…ELSE IF…ELSE

```
if(condition)
{

}
else if (condition2)
{

}
else
{

}
```

SWITCH

```
switch (valeur)
{
    case cas1:
        break;
    case cas2:
        break;
    default:
        break;
}
```

WHILE

```
while (condition)
{
    //La condition doit devenir fausse
    //pour sortir de la boucle
}
```

DO…WHILE

```
do
{
    //La condition doit devenir fausse
    //pour sortir de la boucle
    //Le code dans boucle est effectué une première fois
    //avant de regarder la condition
} while (condition);
```

FOR

```
for (int i = 1; i < 10; i++)
{
    //Insérer le code ici
}
```

FOREACH

```
int[] tab = new int[10];
foreach (int a in tab)
{
    //Insérer le code ici
}
```

Les Tableaux

```
//Définit un tableau avec 10 valeurs de type int
//Chaque valeur est par défaut égal à 0
int[] tableauDeInt = new int[10];

//Définit un tableau avec un 4 valeurs de type int
//Les valeurs sont définis à l'instanciation
int[] tableauDeIntDefini = { 1, 2, 5, 7 };
```

Le symbole clé dans une déclaration de tableau est []. Après un type ou une classe, il permet de définir un tableau. Un tableau est quelque chose d'immutable, c'est à dire que sa taille allouée en mémoire ne pourra pas changer.
Nous avons deux façons de déclarer un tableau suivant l'image ci-dessus :

- La première est de définir une taille. Vous remarquerez le mot-clé new, il permet d'instancier un tableau. On le retrouvera plus tard dans notre cours. La valeur numérique entière permet ici de dire que notre tableau aura au maximum 10 entrées qui seront de type int. On pourra ensuite associer des valeurs aux différents int composant le tableau. Par défaut, chaque valeur dans le tableau est égale à 0.
- La deuxième est de créer directement le tableau avec des valeurs. Ici, notre tableau aura comme valeurs 1, 2, 3 et 5 et intrinsèquement, aura une taille de 4.

```
int[] tableauDeInt = new int[10];

tableauDeInt[2] = 10;

tableauDeInt[11] = 5; //Erreur à l'exécution

int[] tableauDeIntDefini = { 1, 2, 5, 7 };

Console.WriteLine(tableauDeInt[5]);

Console.WriteLine(tableauDeIntDefini[2]);
```

Pour associer une valeur à un endroit précis du tableau, on y accède grâce à l'index. Dans la ligne de code tableauDeInt[0] = 2 ; , on met la valeur de l'index 0 du tableau à 2.
Attention, on ne peut pas accéder à une valeur en dehors des index disponibles comme nous montre la ligne tableauDeInt[11] = 10. Cependant, Visual Studio et le compilateur nous laisse écrire le code. Mais à l'exécution, notre programme plantera.
Pour accéder à une valeur, on a juste besoin de l'index de la valeur.

Exercice 4 : Les tableaux

Déclarer un tableau de de type int avec les valeurs suivantes : 6, 2, 5, 11, 9. Puis incrémentez de 1 chaque valeur dans le tableau.
En utilisant le tableau juste avant, échanger la première valeur du tableau avec la dernière.
Dans un nouveau tableau qui prendra la taille du tableau utilisé précédemment, triez le tableau dans l'ordre croissant.