Avant propos

https://discord.gg/ahZZXpuwgZ

A propos de récursivité

La récursivité est un concept qui se base sur l'idée de décomposition d'un problème en sous-problèmes plus simples. Une fonction récursive est une fonction qui s'appelle elle-même dans son processus de résolution. Elle divise le problème initial en un ou plusieurs sous-problèmes de même nature, jusqu'à ce que le problème devienne suffisamment simple à résoudre.

Caractéristiques

Cas de base : Une fonction récursive doit avoir un cas de base qui spécifie quand arrêter la récursion. C'est une condition qui permet à la récursion de se terminer.

Cas général : La fonction récursive doit résoudre le problème en le divisant en sous-problèmes plus petits, qui sont similaires au problème d'origine.

Avantages

La récursivité constitue une solution élégante et concise pour résoudre certains problèmes. Cette approche est particulièrement utile pour résoudre des problèmes complexes, comme les arbres, les graphes….

Inconvénients

La récursivité peut être moins efficace en termes de performance que d'autres approches, car elle nécessite généralement plus de ressources mémoire et de temps d'exécution.
Une utilisation incorrecte de la récursivité peut entraîner des erreurs de débordement de pile (stack overflow) si la récursion ne se termine pas correctement.

Apprentissage

Il est important de comprendre comment concevoir des fonctions récursives correctes en définissant des cas de base appropriés et en évitant les erreurs de débordement de pile.

Prenons par exemple le calcul de la factorielle d'un nombre, une fonction mathématique qui pour une valeur entière positive, retourne le produit de tous les entiers entre 1 et cette valeur. Pour une valeur nulle, la fonction retourne 1. Par exemple, la factorielle de 5, que l'on note "5!", vaut 1*2*3*4*5 = 120.

On peut écrire la fonction factorielle sous la forme d'une simple boucle, de la manière suivante :

```
long  factorielle(int valeur)
{
  long total = 1;

for (int curValeur = 1; curValeur <= valeur; curValeur++)
     total *= curValeur;
  return total;
}
```

Il est cependant possible de donner une définition récursive de la fonction factorielle :

La factorielle d'un nombre N vaut 1 si N est égal à 0, et N multiplié par la factorielle de N - 1 sinon.
Cette définition est parfaitement équivalente à la précédente, et peut se traduire en code par une fonction récursive :

```
long factorielle(int valeur)
{
  if (valeur == 0)
    return 1;
  else
    return valeur * factorielle(valeur - 1);
}
```

On peut remarquer que le code de cette deuxième version est plus simple que la version avec une boucle, et qu'il peut se lire quasiment comme une définition.

La première version, qui utilise une boucle, est ce que l'on appelle une implémentation ***itérative*** de la fonction factorielle : on effectue un certain nombre d'itérations d'une boucle. La deuxième version s'appelle tout simplement l'implémentation ***récursive***.

Contexte d'apprentissage
https://dotnetfiddle.net/
Vous pouvez naturellement utiliser un IDE ou tout utre environnement qui vous satisfasse.

Itératif vers récursif
Un programme itératif se base sur des boucles pour traiter un certain nombre d'éléments. Un programme itératif simple peut donc ressembler à l'exemple suivant, qui affiche un certain nombre de fois un caractère :

```
using System;
public class Program
{

void afficheLigne(int nbAffichages, char caractere)
{
 for (int affichages = 0; affichages < nbAffichages; affichages++)
   Console.Write(caractere);
    Console.WriteLine("");

}

public void Main()
        {
                afficheLigne(10,'a');
        }
}
```

Pour écrire une version récursive de ce programme, on commence par se ***demander dans quel cas la boucle n'est plus utilisée.*** En l'occurence, il s'agit du cas où le paramètre nbAffichages vaut 0, donc qu'on ne fait qu'afficher le retour à la ligne.

Reste à gérer le cas où il y a des choses à afficher. Le principe de la fonction récursive est qu'elle s'occupe d'une seule étape, et laisse les étapes suivantes pour les appels imbriqués. Dans le cas où il y a des caractères à afficher, la fonction doit donc afficher un caractère, puis se rappeler, avec comme paramètre le nombre de caractères restant à afficher. Il s'agit de la valeur qu'on lui a transmise, diminuée de 1.

```
using System;
public class Program
{

void afficheLigneRecursif(int nbAffichages, char caractere)
{
  if (nbAffichages == 0)
    Console.WriteLine("");
  else
  {
     Console.Write(caractere);
    afficheLigneRecursif(nbAffichages-1, caractere);
  }
}

          public void Main()
          {
                    afficheLigneRecursif(10,'b');
          }
}
```

Exercice 1
Écrivons un programme appelant une procédure récursive *afficherEntiersDeNAM* qui reçoit deux entiers *N* et *M* en entrée avec *N* <= *M*, et qui affiche dans l'ordre, tous les entiers de *N* à *M*, séparés par des espaces.

```
public void Main()
          {
afficherEntiersDeNAM(10, 20);
          }
}
```

Exercice 2
Soit le prototype de la fonction Puissance

double Puissance(double baseNombre, int exposant)
dont l'appel suivant

```
Console.WriteLine(3 + "^" + 4 + " = " + Puissance(3, 4));
```

donne pour résultat

```
3^4 = 81
```

Ecrivez la fonction Puissance sans recourir à une boucle ni à aucune variable locale à la fonction.

Exercice 3

Soit le programme suivant :

```
using System;
using System.Collections.Generic; //Necessaire pour l'usage de List

class Program
{
   int SumRecursiveList(List<int> lst)
   {

                return lst[0] + SumRecursiveList(lst.GetRange(1, lst.Count - 1));
   }

   public void Main()
   {
      List<int> largeList = new List<int>(1000);
      for (int i = 0; i < 1000; i++)
      {
         largeList.Add(i);
      }

Console.WriteLine(SumRecursiveList(largeList));
   }
}
```

Sans executer le code source, identifiez le problème avec la fonction *SumRecursiveList*
Corrigez le problème et vérifiez en executant le code.

Exercice 4

Soit le prototype de fonction suivante

int TrouverMax(int[] tableau, int index, int max)

qui appelée de la sorte

```
int[] tableau = { 5, 8, 3, 12, 7, 2, 10 };
 Console.WriteLine("La valeur maximale dans le tableau est : " + TrouverMax(tableau, 0, tableau[0]);
```

Retourne la valeur 12

Ecrire la fonction *TrouverMax*. Celle-ci ne devra contenir ni boucle ni variable locale.

Exercice 5

Soit le prototype de fonction suivante

long sommeDesCarres(int n)

qui calcule la somme des n (strictement positif) premiers carrés. Pour n valant 4, ce sous-programme calculera 1^2+ 2^2+ 3^2+ 4^2.

Exercice 6
Soit la fonction suivante qui retourne la valeur 'inversée' d'une chaine de caractères.

```
String InverserChaine(string chaine)
   {
    if (chaine.Length == 0|| chaine.Length == 1)
      {
         return chaine ;
      }
      else
      {
                  // Inversons la sous-chaîne en enlevant le premier caractère (chaine[0])
                  //et en inversant  le reste(chaine.Substring(1))
         return InverserChaine(chaine.Substring(1)) + chaine[0] ;
      }
   }
```

Inspirez-vous de la fonction *InverserChaine (**sans l'utiliser**) pour écrire une fonction récursive*

bool EstPalindrome(string chaine)

*qui vérifie si **chaine est un palindrome.***

Exercice 7
Le c# pratique le 'boxing' (adaptation du typage au contexte)
Ainsi :

```
int chiffre = '9' - '1';
Console.WriteLine("L'entier chiffre vaut " + chiffre);
```

Affichera

```
L'entier chiffre vaut 8
```

Utilisez cette propriété pour écrire la fonction

int ValeurNumerique(string chaine)

qui calcule la valeur numérique d'une chaîne de caractères composée de chiffres.

Exercice 8
Ecrire la fonction récursive (sans variable locale)

int CompterOccurences(char caractere, string chaine)

qui retourne le nombre d'occurrences d'un caractère dans une chaine.

Exercice 9
Ecrire la fonction récursive

string SupprimerCaractere(string chaine, char caractereASupprimer)

dont l'appel suivant

```
Console.WriteLine("La chaine LaMadone sans le caractere a :  " +
                  SupprimerCaractere( "LaMadone",'a'));
```

Affichera

```
La chaine LaMadone sans le caractere a :  Lmdone
```

Exercice 10
Ecrire la fonction récursive

<u>string MelangerChaines(string chaine1, string chaine2)</u>

dont l'appel suivant

Console.WriteLine("Les chaines Bleu et Blanc mélangées " + MelangerChaines("Bleu","Blanc"));

Affichera

```
Les chaines Bleu et Blanc mélangées  BBlleaunc
```