Récursivité

on dit d'un processus qu'il est récursif lorsqu'il est défini par ses propres termes. Donc un objet est récursif s'il est utilisé lui-même dans sa définition.

Une fonction est dite récursive lorsqu'elle contient une instruction d'appel à elle-même.

Donc l'apparition de l'identificateur d'une fonction dans une expression située dans le corps même de cette fonction implique une exécution récursive de celle-ci.

La récursivité est une technique particulièrement puissante pour des définitions mathématiques.

Une définition récursive doit toujours avoir une définition explicite pour une ou plusieurs valeurs particulières de ses arguments.

Comme les instructions répétitives, les fonctions récursives doivent examiner le problème de leur terminaison. Les appels récursifs sont contrôlés par une condition qui devient fausse au bout d'un certain temps.

Ex : la fonction factorielle de n. On se base sur la relation de récurrence.

n ! = n*(n-1)

```
int factiterative(int n)
{int i, res = 1 ;
 for (i=1 ; i<=n ;i++)
    res=res * i ;
 return res ;
}

int factrecursive(int n)
{if (n==0)
  return 1 ;
 else
  return n*factrecursive(n-1) ;
 }
```

L'exemple de la factorielle, s'il est très simple, est néanmoins un mauvais emploi de la récursivité, car une définition récursive est plus économique. Il faut donc éviter d'utiliser la récursivité lorsqu'on peut la remplacer par une définition itérative. l'utilisation d'une telle fonction est extrêmement couteuse.

les problèmes qu'il faut résoudre en utilisant la récursivité sont les problèmes typiquement récursifs et non itératifs.

Pour exprimer un problème sous forme récursive, il faut le décomposer en trois étapes :

a) **Paramétrage : faisant apparaitre les différents éléments dont dépend la solution et en particulier la taille du problème à résoudre, qui dans des cas favorables devrait décroitre à chaque appel récursif.**
b) **Recherche d'un cas trivial et de sa solution : (étape clé de l'algo) un cas trivial est un cas qui peut être réglé directement sans appel récursif. Ce sera souvent le cas où la taille du problème est égale à une valeur particulière.**
c) **Décomposition du cas général : visant à le ramener à un ou plusieurs problèmes, en principe plus simple (de taille plus petite).**

<u>Exemple des tours de Hanoï</u> :

On dispose de 3 piquets A, B et C et de n disques de différentes tailles. Les disques peuvent être empilés sur les piquets. Les n disques initialement sont sur le piquet A et en ordre de tailles décroissantes.
Le jeu consiste à déplacer les n disques du piquet A au piquet B de telle sorte qu'ils se retrouvent dans l'ordre de départ, en utilisant les conditions suivantes :

a) **A chaque étape un seul disque est déplacé d'un piquet à un autre.**
b) **Un disque ne peut pas être placé au dessus d'un disque plus petit.**
c) **Le piquet C peut être utilisé comme piquet intermédiaire.**

Bien qu'il soit difficile de résoudre ce problème par d'autres moyens, la solution est presque triviale si on pense la résoudre récursivement.
La solution est de réduire le problème en un autre plus petit de même type. Premièrement on déplace (n-1) disques. Ce déplacement est réalisé par un appel récursif. Après on déplace le dernier disque là où il doit être placé. Ensuite on déplace les (n-1) disques au dessus du dernier disque, récursivement.

```
void hanoi(int n, int piq1, int piq2, int piq3)
{if (n==1)
   printf(‘’déplacer un disque du piquet %3d au piquet %3d\n”, piq1, piq2) ;
 else
    {hanoi(n-1, piq1, piq3, piq2) ;
     printf(‘’déplacer un disque du piquet %3d au piquet %3d\n”, piq1, piq2) ;
     hanoi(n-1, piq3, piq2, piq1) ;
     }
}
```

Exemple :

Ecrire une fonction récursive qui lit une série d’entiers et les affiche dans l’ordre inverse de la lecture. La série se termine par la valeur nulle.

```
void imprime()
{int n ;
 scanf(‘’%d”, &n) ;
 if (n != 0)
   {imprime() ;
    printf(‘’%5d”,n) ;
    }
}
```