CHAPITRE 1

LA RECURSIVITE

1. Introduction :

La récursivité est une notion importante de la programmation qui permet de régler des problèmes extrêmement complexes avec seulement quelques lignes. C'est cependant une méthode avec laquelle il est facile de se perdre et d'avoir des résultats imprévisibles ou erronés. Lorsque bien utilisée, c'est cependant un outil simplifiant, lisible, efficace et souvent sous estimé.

La récursivité n'introduit aucune nouvelle instruction ou structure de données. Elle repose simplement sur une nouvelle façon d'utiliser les divers algorithmes solutionnant un problème donné.

2. Définitions :

- Un algorithme (une fonction, une procédure) est dit **récursif** si sa définition (son code) contient un appel à lui-même.
- Un algorithme qui n'est pas récursif est dit **itératif**.

Utilisations variées (liste non exhaustive) :
- Calcul de suite récursive (numérique, graphique, Fibonacci, Factorielle, etc.)
- Calcul de type « diviser pour régner » : recherche, tri, etc.
- Calcul sur des structures de données inductives (listes, arbres, etc.)

3. La récursivité en terme mathématique :

➢ ***Le produit factoriel d'un entier N≥0 (N !):***

$$F(N) = N! = \begin{cases} F(0)=1 \\ F(1)=1* F(0) \\ F(2)=2*F(1) \\ . \\ . \\ F(N)=N*F(N-1) \end{cases}$$

➢ ***Les nombres de Fibonacci:***

$$Fib = \begin{cases} Fib(0)=0 \\ Fib(1)=1 \\ Fib(2)=Fib(0)+Fib(1) \\ : \\ Fib(N)=Fib(N-2)+Fib(N-1) ; \quad \text{si } N\geq 2 \end{cases}$$

➢ ***Le calcule $f(n,x)=x^n$***

$$f(n,x)= \begin{cases} f(0)=1 \\ f(1)=x*f(0)=x \\ f(2)=x^2 =x*f(1) \\ : \\ f(n,x)=x^n= x*f(n-1) \end{cases}$$

4. Expression des fonctions et des procédures récursives :

4.1. Fonctions récursives :

```
Fonction F(A1,A2, …, An) : type :
{Déclarations locales à la fonction F }
Début
:
F := expression ;   cette expression contient un appel de la fonction F
:
Fin
```

4.2. Procédures récursives

```
Procédure P (A1,A2,…, An);
{Déclarations locales à la fonction P }
Début
:
P(B1,B2, …, Bn) ; appel récursif de la procédure P
:
Fin
```

5. Récursivité simple et croisé :

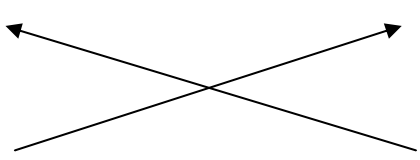
5.1. Récursivité simple

Dans la récursivité simple tous les appels apparaissent dans le corps de la fonction (procédure) récursive

```
Procédure P(x)
Début
:
P(f(x))
:
Fin
```

5.2. Récursivité croisée

Dans la récursivité croisée, les appels récursifs sont provoqués par l'exécution d'autres procédure ou fonction.



```
Procédure P(x)
Début
:
Q(f(x)) ;
:
Fin
```

```
Procédure Q(y)
Début
:
P(g(x))
:
Fin
```

6. Règles de conception d'un algorithme récursif

Première règle

Tout algorithme récursif doit distinguer plusieurs cas, dont l'un au moins ne doit pas comporter d'appel récursif. Sinon risque de cercles vicieux et de calcul infini.

Exemple :

```
f a c t (n ) :
f a c t (n) = n×f a c t (n − 1)
```

fact(1) ➔ 1· fact(0) ➔ 1 · 0· fact (-1)➔ . . . ➔ calcul infini

Condition de terminaison, cas de base

- Les cas non récursifs d'un algorithme récursif sont appelés cas de base.
- Les conditions que doivent satisfaire les données dans ces cas de base sont appelées conditions de terminaison.

Seconde règle

Tout appel récursif doit se faire avec des données plus *proches* de données satisfaisant une condition de terminaison

Exemple : Même avec un cas de base un algorithme récursif peut ne produire aucun résultat

```
f a c t (n ) :
        s i n = 0 a l o r s {condition  de terminison}
        f a c t ( 0 ) = 1 {cas de base}
        s inon
        f a c t (n) = f a c t (n + 1) / (n + 1)
        f i n s i
```

fact(1) ➔ fact(2)/2 ➔ fact(3)/(2 · 3) ➔ . . . ➔ calcul infini.

7. Exemples:

Exemple 1 : La somme des entiers

Construire une fonction récursive qui calcule la somme des entiers ≤N. Nous établissement facilement la relation de récurrence suivante :

$S_0 = 0$
$S_1 = S_0 + 1$
$S_2 = S_1 + 2$
:
$S_n = S_{n-1} + n$

```
Fonction Somme (donnée N : entier) : entier ;
Début
        Si N=0 alors            {condition de terminaison}
                Somme := 0      {cas de base}
        Sinon
                Somme :=N+Somme (N-1)
        Fsi
Fin
```

❖ Traces d'exécution pour N=5:

Somme (5)=
(5+Somme (4))
(5+ (4+ Somme(3)))
(5+ (4+ (3+ Somme (2))))
(5+ (4+ (3+ (2+ Somme(1)))))
(5+ (4+ (3+ (2+ (1+Somme(0))))))
(5+ (4+ (3+ (2+ (1+0)))))
(5+ (4+ (3+ (2+ (1)))))
(5+ (4+ (3+ (3))))
(5+ (4+ (6)))
(5+ (10))
(15)

Exemple 2 : Le factoriel d'un entier N≥0

```
Fonction Fact ( donnée N : entier) : entier
Début
        Si (N=0) alors
                Fact:=1 {initialisation du calcul}
        Sinon
                 Fact := N* Fact (N-1)
        Fsi
Fin
```

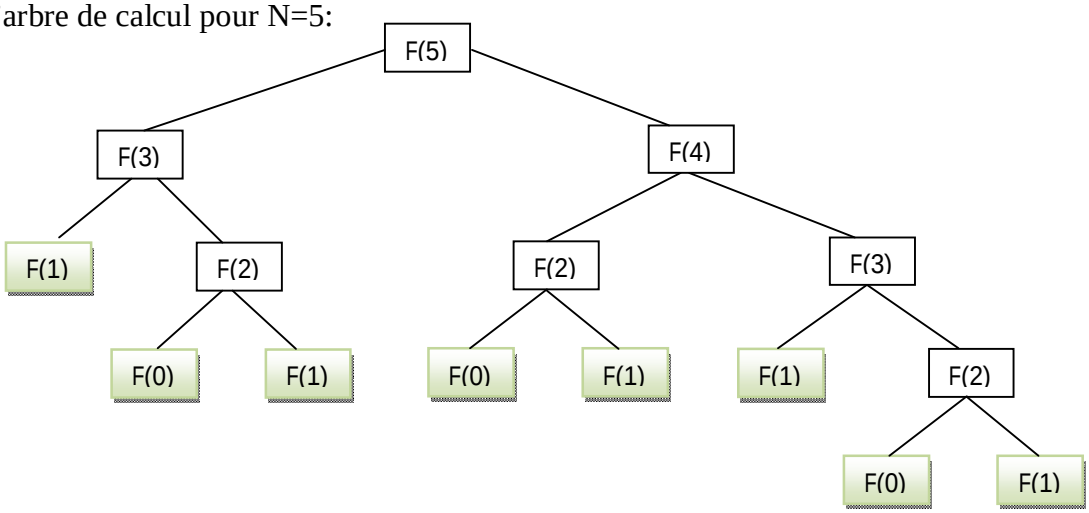
❖ Traces d'exécution pour N=5:

Fact (5)
(5* Fact(4))
(5*(4* Fact(3)))
(5*(4* (3*Fact(2))))
(5*(4* (3*(2*Fact(1)))))
(5*(4* (3*(2*(1*Fact(0))))))
(5*(4* (3*(2*(1*1)))))
(5*(4* (3*(2*1))))
(5*(4* (3*2)))
(5*(4* 6))
(5*24)
(120)

Exemple 3 : Le calcul de la suite Fibonacci (voir section II)

```
Fonction Fib ( donnée N : entier) : entier
Début
        Si (N=0) alors
                Fib:=0
        Sinon
                 Si N=1 alorss
                        Fib := 1
                Sinon
                        Fib := Fib (N-2)+Fib(N-1)
                Fsi
        Fsi
Fin
```

❖ L'arbre de calcul pour N=5:



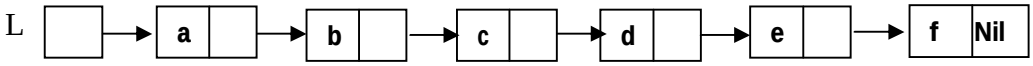
⇨ *Le calcul récursif de cette fonction n'est pas efficace.*

Remarque : la récursivité ne constitue pas toujours exécution efficace d'un programme. En effet, un processus peut effectuer des calculs déjà faits.

Exemple 4 : La longueur d'une liste chaînée L

```
Fonction Longueur  ( donnée Pointeur :liste) : entier ;
Début
        Si Pointeur=Nil alors
                Longueur :=0 {initialisation du calcul}
        Sinon
                Longueur :=1+Longueur( suivant (Pointeur))
        Fsi ;
Fin
```

❖ Traces d'exécution pour L:



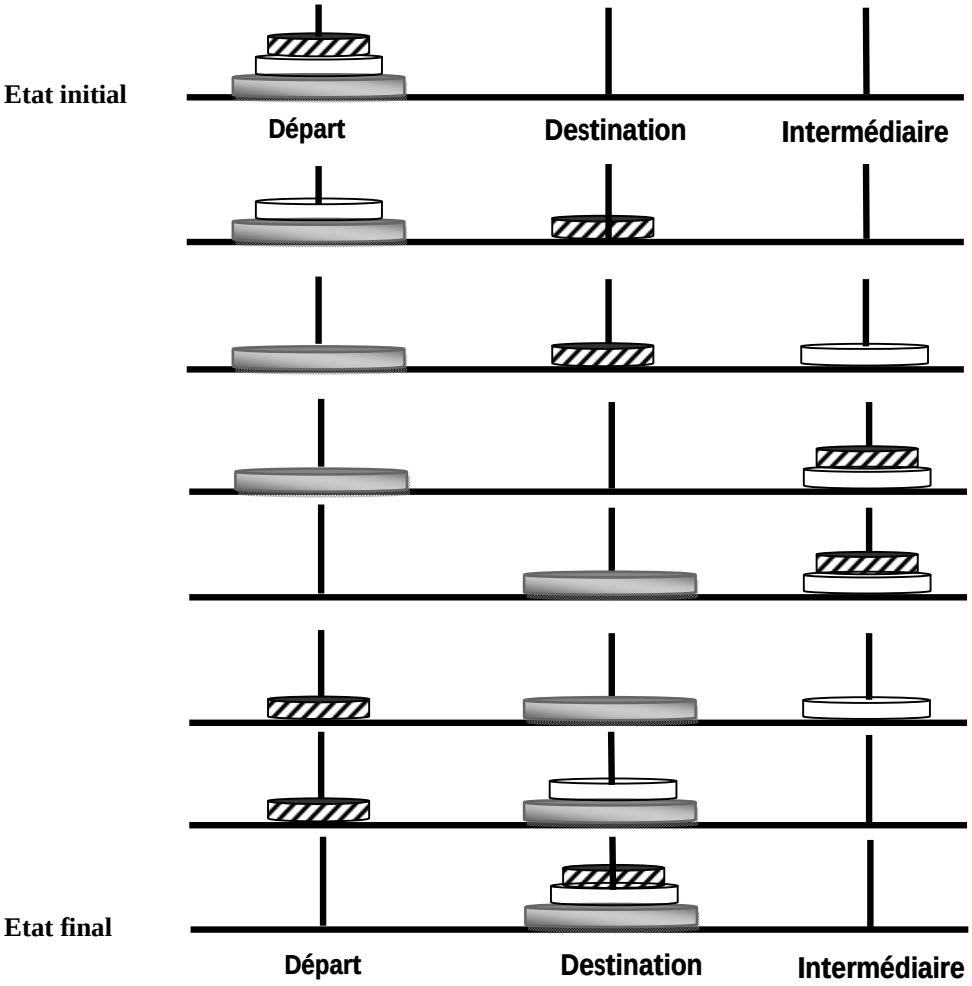
L=<a,b,c,d,e,f,Nil>
(1+<b,c,d,e,f, Nil >)
(1+(1+<c,d,e,f, Nil >))
(1+(1+ (<d,e,f, Nil >)))
(1+(1+ (1+<e,f, Nil >))))
(1+(1+ (1+(1+<f, Nil >)))))
(1+(1+ (1+(1+(1+(1+< Nil >))))))
(1+(1+(1+ (1+(1+(1+(0)))))) ➔ initialisation du calcul récursif
(1+(1+(1+ (1+(1+(1))))))
(1+(1+(1+ (1+(2)))))
(1+(1+(1+ (3))))
(1+(1+(4)))
(1+(5))
(6)

Exemple 5 : La tour de Hanoi

Il s'agit de déplacer les N disques d'une tour départ (D), tous de diamètres différents, vers une autre tour destination (A) en respectant les contraintes suivantes :

- On ne déplace qu'un seul disque à la fois.
- On ne place jamais un disque sur un autre de diamètre inférieur.

Initialement les disques sont empilés sur la tour D dans l'ordre décroissant des diamètres. Une troisième tour T peut être utilisée comme tour de transit.




```
fonction hanoi (nb_disques, Départ, Destination, Intermédiaire)
Début
    Si nb_disques = 1 alors
        Déplacer le disque 1 de Départ vers destination
    sinon
        hanoi (nb_disques -1, Départ, Intermédiaire, Destination)
        Déplacer le disque nb_disques de Départ vers Destination
        hanoi (nb_disques -1, Intermédiaire, Destination, Départ)
    fsi
Fin
```

- Traces d'exécution pour : pour ***hanoi***(3,1,3,2), la fonction affiche :
  - Déplacer le disque 1 de la tour 1 à la tour 3.
  - Déplacer le disque 2 de la tour 1 à la tour 2.

- Déplacer le disque 1 de la tour 3 à la tour 2.
- Déplacer le disque 3 de la tour 1 à la tour 3.
- Déplacer le disque 1 de la tour 2 à la tour 1.
- Déplacer le disque 2 de la tour 2 à la tour 3.
- Déplacer le disque 1 de la tour 1 à la tour 3.