CHAPITRE

10

La récursivité

I. Principe :

Un algorithme ou un sous-programme est dit récursif quand il contient un ou plusieurs appels à lui-même.

La récursivité est utilisée lorsqu'on peut décomposer un problème en sous problèmes de même nature et plus court à traiter. Dans la décomposition en sous problèmes, on doit toujours arriver à un cas où on peut résoudre le problème sans faire un appel récursif. Ce cas est appelé cas d'arrêt.

Remarque : Si un sous programme fait dans tous les cas un appel récursif, l'exécution ne se termine jamais.

1. Récursivité simple :

La récursivité est simple si elle contient dans son corps un seul appel récursif.

Exemple1 : Le calcul de la factorielle de N.

N != N*(N-1)*(N-2)*…*2*1 , on peut écrire ainsi N != N*(N-1)!

→ La factorielle de N est définie en fonction de la factorielle de N-1

→ La fonction a besoin d'elle-même pour donner un résultat : Pour calculer N!, il faut savoir calculer (N-1) ! et pour calculer (N-1)!, Il faut savoir calculer (N-2)! et ainsi jusqu'à 1! qui est égal à 1 et qui permet à la récursivité de s'arrêter après une série d'auto appels.

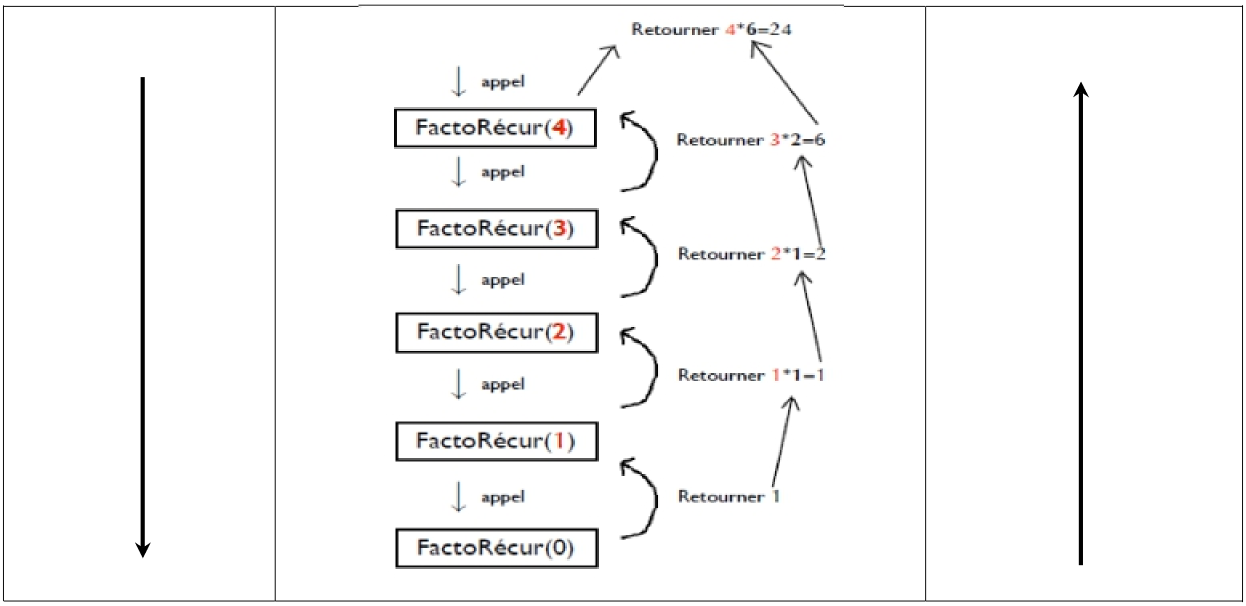
→ Il est donc impératif de prévoir une condition d'arrêt à la récursion sinon le programme ne s'arrête jamais.

<table>
<tr>
<td>Fonction factoriel (n : entier) : entier
Variables
i,f : entier
Debut
f ← 1
pour i de 2 à n faire
f ← f * i
finPour
factoriel ← f
fin</td>
<td>Fonction factoriel (n : entier) : entier
Variables
Debut
Si (n=0) alors
factoriel ← 1 //condition d'arrêt
sinon
factoriel ←n * factoriel(n-1) //appel recursif
finsi
fin</td>
</tr>
<tr>
<td colspan="2">⇒ A l'opposé de la récursion, l'itération utilise les structures de contrôle répétitives comme POUR, TANT QUE, REPETER JUSQU'À.</td>
</tr>
</table>

Exécution d'un appel récursif :

- L'exécution d'un appel récursif passe par deux phases, la phase de descente et la phase de remontée.
- Dans la phase de descente, chaque appel récursif fait à son tour un appel récursif.
- En arrivant à la condition terminale, on commence la phase de remontée qui se poursuit jusqu'à ce que l'appel initial soit terminé, ce qui termine le processus récursif.

Phase de descente   Phase de la remontée



Exemple2 : Algorithme récursif de calcul de PGCD de deux entiers :

```
Fonction PGCD (a, b : entier) : entier
Variables
Debut
 Si (a = 0 ou b = 0) alors
 PGCD ← 1 //condition d'arrêt
 Sinon
        Si (a = b)  alors PGCD ← a //condition d'arrêt
        Sinon
               Si (a > b) alors PGCD ← PGCD(a-b,b)// ou retourner PGCD(a-b,b)
                      Sinon PGCD ← PGCD(a, b - a)
               Finsi
        Finsi
Finsi
Fin
```

2. Récursivité mutuelle :

Des fonctions sont dites mutuelles récursives si elles dépendent les unes des autres.

Exemple : La définition de la parité.

Pair (n) = { vrai si n = 0 ; impair (n − 1) sinon }   ET   impair (n) = { faux si n = 0 ; pair (n−1) sinon }

Fonction pair(n : entier) : boolean Si (n=0) alors retourner vrai sinon retourner impair(n-1) finsi Fin	Fonction impair(n : entier) : boolean Si (n= 0) alors retourner faux sinon retourner pair(n-1) finsi Fin

3. **Récursivité imbriquée :** Elle consiste à faire un appel récursif à l'intérieur d'un autre appel récursif.

Exemple : La fonction d'Ackermann.

$$A(m, n) = \begin{cases} n + 1 \text{ si } m = 0 \\ A(m-1, 1) \text{ si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m, n-1)) \text{ sinon} \end{cases}$$

<u>Fonction Ackermann(m : entier, n : entier) : entier</u>
<u>Debut</u>
Si m = 0 alors
    Retourner n+1
Sinon
    Si n = 0 et m > 0 alors
        Retourner Ackermann(m-1, 1)
    Sinon
    Retourner Ackermann(m-1, Ackermann(m, n-1))
    Finsi
Finsi
<u>Fin</u>

m = 5, n = 2 A(5,2) = A(4,A(5,1)) = A(4, 2) = 2
A(5,1) = A(4,A(5,0)) = A(4, 2) = A(3, A(4,1)) = A(3, 2) = A(2, A(3,1)) = A(2,2) =2
A(5,0) = A(4,1) = A(3,A(4, 0)) = A(3, 2) = A(2,A(3,1)) = A(2,2) =2
A(4, 0) = A(3 , 1) = A(2,A(3,0)) = A(2,2) = A(1,A(2,1)) = A(1, A(2,0)) = A(1,3) = A(0,1) = 2
A(3,0) = A(2, 1) = A(1,A(2,0)) = A(1,3) = A(0,1) = 2
A(2,0) = A(1,1) = A(0, A(0,1)) = A(0,2) = 3
A(0,1) = 2

Algorithme récursif de la recherche du maximum dans un tableau :

Soit Tab un tableau à n éléments, écrire une fonction récursive permettant de rechercher l'indice du maximum dans Tab.

```
Fonction  maximum  (T:tab,  indDeb : entier,  indFin: entier) : entier
Variables
      M, K1, K2 : entier Debut
Debut

        Si (indDe =  indFin ) alors
                maximum ← indDeb
        sinon
                m ← (indDeb + indFin)div2
                //  division  du  problème  en  deux  sous problèmes
                K1  ← maximum(T,  indDeb,  m) //régner sur le 1ier sous-problème
                K2 ← maximum(T,  m + 1, indFin)  //   régner   sur   le   2ieme   sous-problème

                Si (T[K1] > T[K2]) alors
                        maximum ←K1
                sinon
                        maximum ← K2
                Finsi
        Fin Si

Fin
```


Algorithme récursif de la recherche dichotomique :

Soit Tab un tableau à ***n*** éléments trié dans ordre croissant. La recherche par dichotomie compare l'élément cherché ***x*** avec l'élément en position ***m*** situé en position du milieu du sous-tableau :

- Si Tab[***m***] = ***x*** → alors on a trouvé l'élément x en position m.
- Si Tab[***m***] > ***x*** → il est possible que ***x*** se trouve avant la position m du tableau. Il faut uniquement chercher dans la moitié inférieure du tableau.
- Enfin Si Tab[***m***] < ***x*** → il est possible que ***x*** se trouve après la position ***m***. Il faut traiter uniquement la moitié supérieure du tableau.

On continue ainsi la recherche jusqu'à trouver l'élément ou bien aboutir à un tableau de taille= 0, dans ce cas ***x*** n'est pas présent et la recherche s'arrête.

```
Fonction dicho_recursif (T:tab, borneinf, bornesup : entier ; x : entier): booleen
Variable :
mil : entier
Debut
Si (borneinf > bornesup) Alors
        dicho _recursif ← faux // x n'existe pas
Sinon
        // borneinf<= bornesup
        mil ← (borneinf + bornesup) div 2
        Si  (T[mil] = x) alors
         dicho _recursif ←vrai
         sinon
                si (T[mil] > x) alors
                        dicho _recursif  ← dicho_recursif(T, borneinf, mil-1, x)
                sinon
                        dicho _recursif ← dicho_recursif(T, mil+1, bornesup, x)
                finsi
        finsi
 Finsi
Fin
```

Trace de l'algorithme :

1	2	3	4	5	6	7	8	9	10
45	50	62	63	70	85	100	140	180	192

X

x = 72

mil = (1+10)div2 = 5

dicho _recursif (T, 1, 10 , 72) =

dicho _recursif (T, 6, 10 , 72) mil = (6+10) div2=8

dicho _recursif (T, 6, 7, 72) mil = (6+7) div2 = 6

dicho _recursif (T, 6, 5 , 72)

borneinf > bornesup alors faux // x n'existe pas

x = 70

mil = (1+10)div2 = 5

Vrai // x existe.