**Définition**

Un programme récursif c’est un programme qui s’appelle lui-même ou une fonction qui est définie par rapport à elle-même.

**Structure d’un programme récursif**

|  |
| --- |
| **P** |
| **\_** |
| **\_** |
| **Appel à P** |
| **\_** |

Procedure P

Begin

<instructions>

Appel à P

<instructions>

End ;

**Exemple 1** : Calcul de factoriel « n » ;

n !=n\*(n-1)\*(n-2)\*…………..\*3\*2\*1.

0 !=1

1 !=1

Fact (n)= n\*(n-1)\*(n-2)\*…………..\*3\*2\*1.

Cas général: Fact (n)= n\*Fact (n-1) .

Cas particuliers: Fact (0)=1 ; Fact (1)=1

Algorithme 1: Fact\_Réc

Début

/\* entrée : un nombre n\*/

/\* sortie: la factorielle de n\*/

Si n<=1 alors

Retourner 1

Sinon

Retourner n\* Fact\_Rec(n-1)

Fin

**Déroulement : exécution :**

Fact\_Rec (4)= ?

n=4

Retourne 4\* Fact\_Rec (3)

n=3

Retourne 3\* Fact\_Rec (2)

n=2

Retourne 2\* Fact\_Rec (1)

n=1

Retourne 1

Fact\_Rec (4)= 4\*3\*2\*1

**Remarque :**

La gestion de la mémoire se fait par pile.

**Règles**

* Ne jamais réappliquer l’algorithme sur des données plus grandes.
* Toujours effectuer un Test de Terminaison.

**Factorielle : version itérative :**

Algorithme Fact\_Iter

Début

/\* entrée : un nombre n\*/

/\* sortie: la factorielle de n\*/

f 1

pour i de 2 à n faire

f f\*i

Fin pour

Fin

**Déroulement : exécution :**

Fact\_Iter(0)=?; Fact\_Iter(1)=?

i=2

f 1\*2

i=3

f 1\*2\*3

i=4

f 1\*2\*3\*4

**Solution recursive ou iterative ?**

* Cela dépond de l’application
* Du coût en temps d’exécution et en place mémoire.
* Factorielle : équivalence des solutions itérative et récursive.
* Ce n’est pas toujours le cas.

**Exemple 2** : le nombre de fibonacci ;

Fibo : N N

Fibo(0)=1

Fibo(1)=1

Fibo(n)= Fibo(n-1)+ Fibo(n-2)

**Version récursive**

Algorithme Fibo\_Rec

Début

/\* entrée : un nombre n\*/

/\* sortie: le nombre de fibonacci\*/

Si n=0 ou n=1 alors

Retourner 1

Sinon

Retourner Fibo\_Rec (n-1)+ Fibo\_Rec (n-2)

Fin

**Complexité du nombre de Fibonacci récursif**

Opérations significatives : nombre d’appel de la fonction Fibo\_Rec

Le nombre d’appel croit exponentiellement en fonction de n.

Fibo\_Rec(0) : 1 appel

Fibo\_Rec(1) : 1 appel

Fibo\_Rec(2) : 3 appels

Ap(Fibo\_Rec(n)) =1+Ap(Fibo\_Rec(n-1) + Ap(Fibo\_Rec(n-2)

Fibo\_Rec(3) : 5 appels

Fibo\_Rec(4) : 9 appels

Fibo\_Rec(5) : 15 appels

**Complexité du nombre de Fibonacci itératif**

Algorithme Fibo\_Iter

Début

/\* entrée : un nombre n\*/

/\* sortie: le nombre de fibonacci\*/

A=1

B=1

Pour k=1 à n-1 faire

B=A+B

A=B-A

Retourner B

Fin

Complexité faible par apport à celle de la version récursive mais écriture non immédiate : elle demande de la réflexion.