Chapitre 2 :

La récursivité

Activité

Ecrire un algorithme d'une fonction **Fact** permettant de calculer la factorielle d'un entier **n** supérieur ou égal à 0.

On rappelle que la factorielle d'une entier positif **n** notée en mathématiques **n!** est calculée comme suit :

$$\begin{cases} 0! = 1 \\ n! = n * (n-1) * (n-2) * \ldots\ldots\ldots 3 * 2 * 1 \quad \text{(pour } n>0\text{)} \end{cases}$$

Exemple : $5! = 5 * 4 * 3 * 2 * 1 = 120$

fonction Fact (n : entier) : entier
Debut
f ← 1
Pour i de 2 a n faire
f ← f * i
finpour
retourner f

Solution intérative

La factorielle d'un entier peut être calculée d'une autre façon.

Exemple : Pour n = 5

Fact (5) = **5 *** $\underbrace{4 * 3 * 2 * 1}$
5 * Fact (4)
4 * $\underbrace{3 * 2 * 1}$
4 * Fact (3)
3 * $\underbrace{2 * 1}$
3 * Fact (2)
2 * $\underbrace{1}$
2 * Fact (1)
1 * Fact (0)
1 * **1**

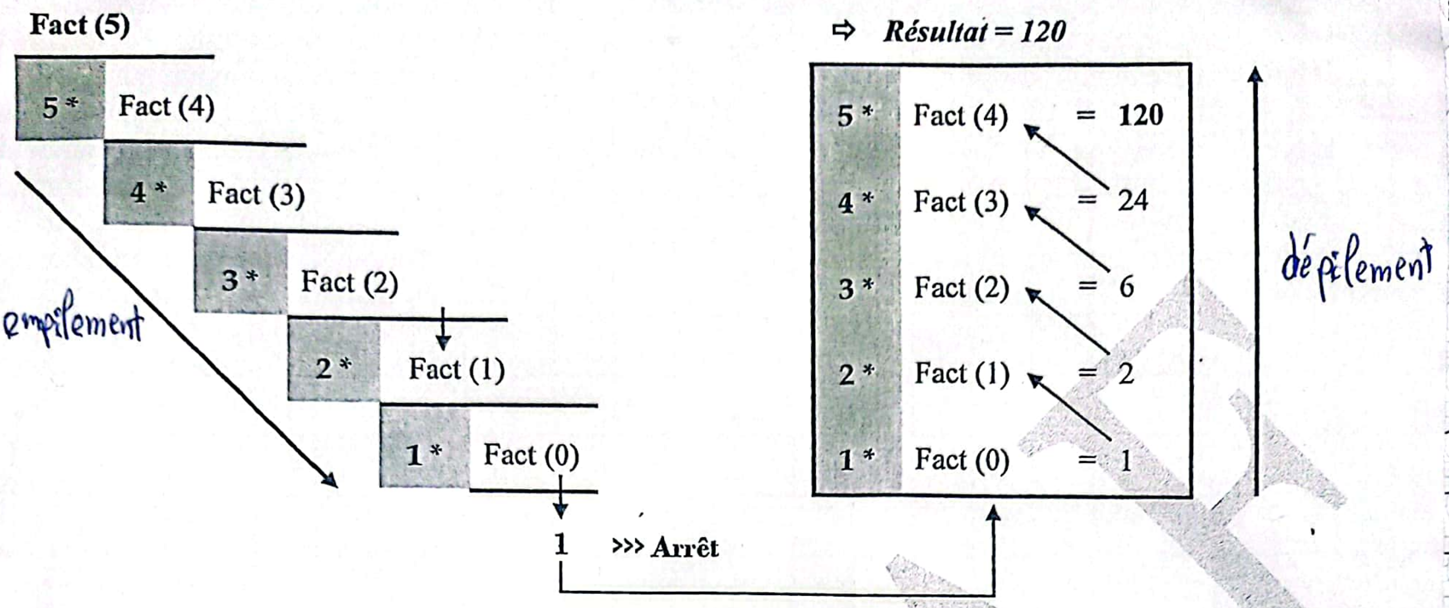
A partir de cet exemple, proposer une autre formule pour calculer la factorielle de n.

Formule 2 :

0! = 1

n! = n * (n-1)!

Explication :



Solution récursive :

```
fonction fact(n:entier):entier
Debut
  si n=0 alors
     retourner 1
  sinon
     retourner n*fact(n-1)
  finsi
fin
```

Définitions

- La récursivité est une méthode algorithmique qui consiste à appeler un module dans son propre corps.
- On appelle récursive toute fonction ou procédure qui s'appelle elle-même.
- Un module récursif s'appelle lui-même **avec de nouvelles valeurs de paramètres effectifs** dans chaque appel jusqu'à atteindre une condition d'arrêt.
- Un traitement récursif doit comporter :
  - Une (ou plusieurs) cas d'arrêt de tout appel récursif.
  - Un cas général (ou plusieurs cas généraux) dans lequel un ou plusieurs appels seront effectués (en changeant les paramètres)

Applications

1) Calcul de la puissance d'un entier

Ecrire un algorithme d'un module permettant de calculer la puissance n^e d'un entier $n \geq 0$.

2) Calcul du pgcd de deux entiers (principe des différences)

Ecrire un algorithme d'un module permettant de calculer le PGCD de deux entiers strictement positifs en utilisant le principe des différences.

3) Somme des chiffres d'un entier

Ecrire un algorithme d'un module permettant de calculer la somme des chiffres d'un entier positif.

4) Somme des chiffres d'une chaîne numérique :

Ecrire un algorithme d'un module permettant de calculer la somme des chiffres d'une chaîne numérique.

5) Vérification si une chaîne est un palindrome :

Ecrire un algorithme d'un module permettant de vérifier si une chaîne est palindrome ou non.

```
1) fonction puiss(a, e:entier):entier
   Debut
     si e=0 alors
        retourner 1
     sinon
        retourner a * puiss(a, e-1)
     finsi
   fin
```

```
2) fonction pgcd(a,b:entier):entier
   Debut
     si a=b alors
        retourner a
     sinonsi a>b alors
        retourner pgcd(a-b,b)
     sinon
        retourner pgcd(a,b-a)
     finsi
   fin
```

```
3) fonction somme(n:entier):entier
   Debut
     si n=0 alors
        retourner 0
     sinon
        retourner n mod 10 + somme(n div 10)
     finsi
   fin
```

4)
```
fonction somech(ch:chaine):entier
Debut
    si long(ch)=0 alors
        retourner 0
    sinon
        retourner valeur(ch[0])+somech(sous_chaine(ch,1,long(ch)))
    finsi
fin
```

5)
```
fonction paland(ch:chaine):booleen
Debut
    si long(ch)=0 ou long(ch)=1 alors
        retourner vrai
    sinon
        si ch[0] ≠ ch[long(ch)-1] alors
            retourner faux
        sinon
            retourner paland(sous_chaine(ch,1,long(ch)-1))
        finsi
    finsi
fin
```