Chapitre 1(partie 2): La récursivité

1. Notion de la récursivité :

- En programmation, la ***récursivité*** est une méthode qui permet à un sous-programme (procédure ou fonction) de s'appeler elle-même.
- C'est Dans la partie corps (instructions) on retrouve un appel à la procédure (fonction) elle-même.

Exemple : écrire un algorithme (fonction) permettant de calculer la factorielle d'un entier N donné ?

a) Solution itérative (classique) :

N !=1*2* 3*4*5*……*(N-1)*N.

Fonction fact (N : entier):Entier ;

R, i : entier ;

Debut

R ← 1;

Pour i allant 2 à N **faire**

R ← R* i ;

FinPour

Retourner (R) ;

Fin.

b) Solution récursive:

N !=N*(N-1) !

= N*(N-1)*(N-2) !

=N*(N-1)*(N-2)*……* 0!

- la fonction factorielle « ***fact*** » est définie comme suit :
  - ✓ Si N=0: fact(0)=1.
  - ✓ Si N<> 0: fact(N)=N*fact(N-1).
- En algorithmique la fonction ***fact*** est définie comme suit :

Fonction fact (N : entier):Entier ;

R : entier ;

Début

Si (N = 0) alors

R ← 1;

Sinon

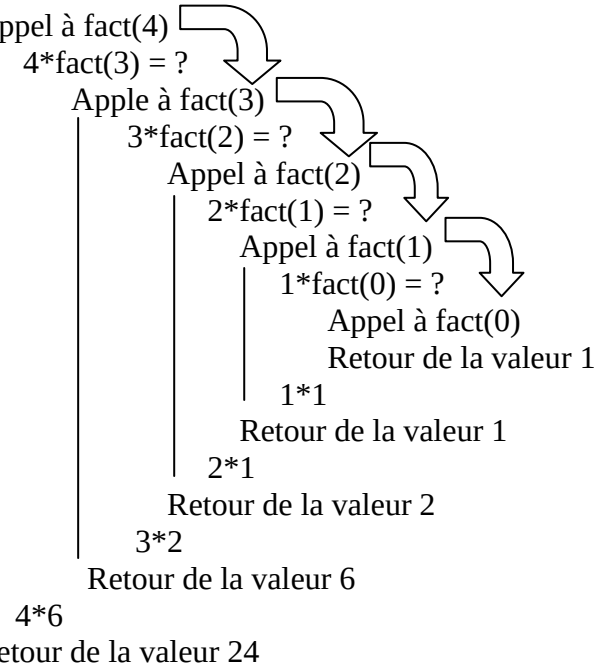
R ← N * fact (N - 1);

Appel récursive

Finsi

Retourner (R) ;

Fin.



2. Types de récursivité :

- On distingue en général deux types de récursivité :
  - ✓ La récursivité **simple**
  - ✓ La récursivité **croisée**

2.1. La récursivité simple :

- C'est lorsqu'un sous-programme (fonction ou procédure) s'appelle lui-même. C'est en effet le cas général de récursivité comme on l'a déjà vu dans l'exemple précédent avec la fonction **factorielle**.

Syntaxe :

Procédure P
Début
…
Appel de P ;
…
Fin ;

Exemple : calcul de la somme de **n** premiers nombres entiers positifs.

Somme(n) = 1 + 2 + 3 + … + (n-1) + n

Fonction Somme (n : entier) : entier ;
S: entier ;
Début
Si (n = 1) alors
S←1 ;
Sinon
S ← Somme (n-1) + n ;
Finsi
Retourner (S) ;
Fin.

2.2. La récursivité croisée :

- On appelle récursivité croisé le fait que **deux procédures** P1 et P2 s'appellent **mutuellement**, c.-à-d : lorsque **P1** s'exécute, elle fait appel à **P2**, et lorsque **P2** s'exécute, elle fait appel à **P1**.

Syntaxe :

Procédure P1
Début
…
Appel de P2 ;
…
Fin ;
Procédure P2
Début
…
Appel de P1 ;
…
Fin ;

Exemple :

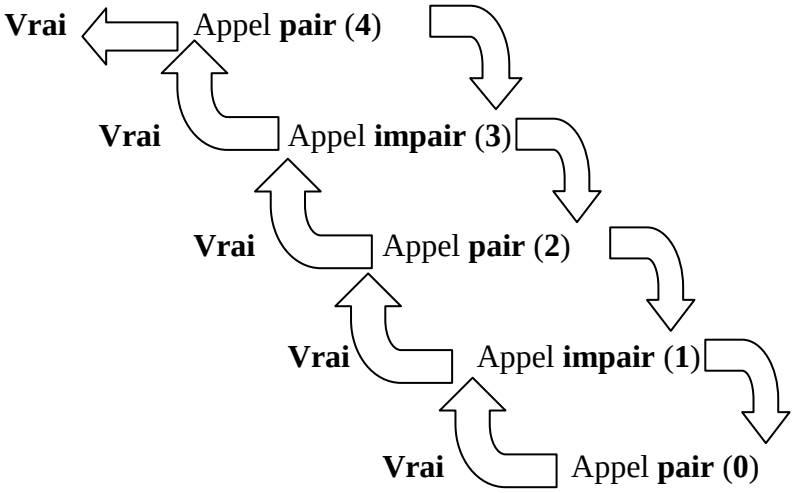
- Un nombre entier positif n peut être soit :

Pair → n = 2*k
Impair → n = 2*k+1

- Si on considère deux fonctions **Pair** (n) et **Impair** (n) à valeurs logiques (**booléen**) alors on aura :
  Si **Pair** (n) = vrai alors **Impair** (n) = faux
  Si **Impair** (n) = vrai alors **Pair** (n) = faux

Fonction Pair (n : entier): booléen ; R: entier ; **Début** **Si (**n = 0) alors R ← vrai ; **Sinon** R ← Impair (n-1) ; **Finsi** Retourner (R) ; Fin ;	**Fonction** Impair (n : entier): booléen; R: entier ; **Début** **Si (**n = 0) alors R ← faux ; **Sinon** R ← Pair (n-1) ; **Finsi** Retourner (R) ; **Fin.**

Application : on veut vérifier **pair (4) :**



3. Règles de conception :

a) Première règle :

- Chercher à décomposer le problème en plusieurs sous problèmes de même type mais de taille inférieure.

b) Deuxième règle :

- Tout algorithme récursif doit distinguer plusieurs cas, dont l'un au moins ne doit pas comporter d'appel récursif (***Condition d'arrêt ou terminaison***). C'est le cas **trivial**, sinon risque de boucler infiniment.

Condition de terminaison :

- ✓ 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**.

Exemple :

Fonction fact (N : entier):Entier ;
R : entier ;
Début
Retourner (N * fact (N - 1)) ;
Fin.

Pas de conditions De terminaison ?!

Appel à fact(4)
4*fact(3) = ?
Apple à fact(3)
3*fact(2) = ?
Appel à fact(2)
2*fact(1) = ?
Appel à fact(1)
1*fact(0) = ?
Appel à fact(0)
0*fact(- 1) = ?
…

Solution :

```
Fonction fact (N : entier):Entier ;
R : entier ;
Début
Si ( N = 0 ) alors
R ← 1;
Sinon
        R ← N * fact (N - 1);
Finsi
   Retourner (R) ;
Fin.
```

Cas de Base

Remarque :

- Puisqu'une fonction récursive s'appelle elle-même, il est impératif qu'on prévoie une condition d'arrêt à la récursivité, sinon le programme ne s'arrête jamais.
- On doit toujours tester en premier la condition d'arrêt, et ensuite, si la condition n'est pas vérifiée, lancer un appel récursif.

c) Troisième règle :

- Tout appel récursif doit se faire avec des données plus ‘’ proches ‘’ de données satisfaisant une condition de terminaison.

Avantages de la récursivité:

- Simplifier l’écriture des programmes, car les calculs à effectuer ne sont pas définis explicitement.
- Faciliter la tâche du programmeur qui n’aura plus à préciser le nombre de répétitions de la même action, ni à gérer les valeurs des différentes variables utilisées.