CHAPITRE 5 : LES SOUS PROGRAMMES

I. L'analyse modulaire

1. Définition

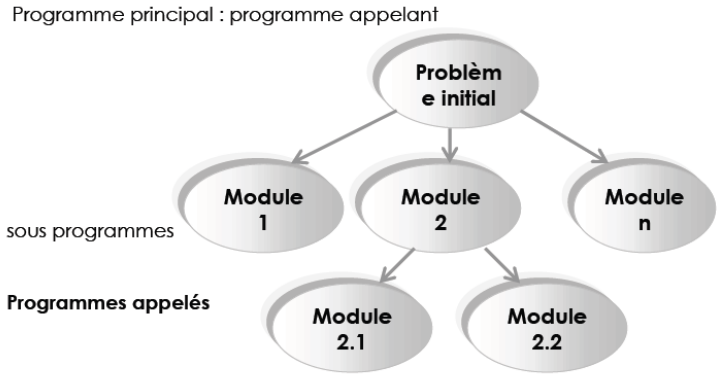
L'analyse modulaire consiste à **diviser** un problème en **des sous problèmes de difficultés moindres**. Ces derniers peuvent être à leur tour divisés en d'autres sous problèmes jusqu'à atteindre un niveau de difficulté abordable.

2. Avantages

- Faciliter la résolution des problèmes.
- Améliorer l'écriture de l'algorithme en évitant les répétitions.
- Faciliter la détection et la localisation des erreurs.
- Permettre la réutilisation des instructions.
- Faciliter l'évolution et la mise à jour des programmes.

3. Notion de sous programme

- Le **problème initial** sera résolu sous forme d'**un programme principal** qui va **appeler les sous programmes** relatifs aux **modules**.
- le programme principal est un **programme appelant**.
- Les sous programmes des **programmes appelés**.



.....

- Un sous-programme peut faire appel d'autres sous programmes
- un sous-programme peut être une procédure ou une fonction

II. Les procédures

1. Définition

Une procédure : est un sous-programme qui peut avoir plusieurs résultats.

2. Appel d'une procédure.

L'appel d'une procédure se fait en écrivant son nom suivi d'entre parenthèses la liste des paramètres effectifs séparés par des virgules en respectant la syntaxe suivante :

PROC Nom_procédure (paramètre effectif1, paramètre effectif 2, ... , paramètre effectif n)

3. Syntaxe.

Dans l'analyse

```
Résultat : .....
Traitement :
        ..... .....
        .....
        .....
```

En Algorithme :

```
0)  DEFPROC nom_procédure (Param formel1, Param formel2:type1;  Param Formel3: type2;...; Param formeln:typen)   ← Entête

1) Instruction 1
2) Instruction 2
3) ..... .....
4) ..... .....
n-1) Instruction n-1
n) Fin nom_procedure
```

Corps de la procédure.

En Pascal

```
Procedure nom_procedure (Param formel1, param formel2:type1;  param Formel3: type2;...;  Param formeln:typen) ;   ← Entête

{Déclaration  des éventuels  objets locaux}

Begin
        Instruction 1 ;
        Instruction 2 ;
        ..... ;                                  } Corps de la procédure.
        ..... ;
        Instruction n-1 ;
End ;
```

III. Déclaration, accès aux objets et mode de transmission

1. Types d'objets

a. Objets globaux

Un objet global est **déclaré au sein du programme principal**. Il peut être utilisé dans un sous-programme sans être déclaré dans celui-ci.

b. Objets locaux

Un objet local est **déclaré dans un sous-programme**. Cet objet est accessible et connu ou sein du module ou il est déclaré ou par un sous-programme appelé.

2. Les paramètres et leurs modes de transmission

a. Types de paramètres

- **Paramètres effectifs**

  Un **paramètre effectif** est celui **utilisé dans une instruction d'appel** du module.

- **Paramètres formels**

  Un **paramètre formel figure dans l'entête de la définition d'un sous-programme**.

Remarques :

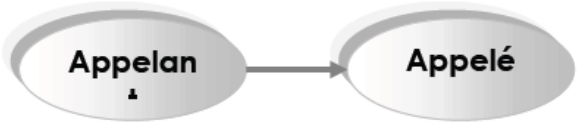
- Les paramètres formels et les paramètres effectifs doivent se correspondre en :
  - **Nombre**
  - **Ordre**
  - **Type**
- les identificateurs des paramètres formels et effectifs peuvent être différents.

b. Modes de transmission des paramètres.

Lors de l'appel d'un module les paramètres effectifs substitueront les paramètres formels, c.à.d. le transfert de données entre le programme appelant et le programme appelé. On parle, ainsi de **passage de paramètres,** Il existe de modes de passage :

- **Passage par valeur**

L'échange de données se fait dans un seul sens :

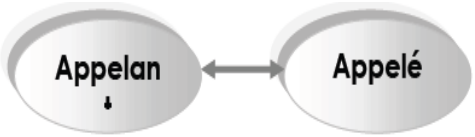


La modification d'un paramètre formel n'entraîne pas la modification du paramètre effectif correspondant.

Remarque : les paramètres d'une fonction sont passés par valeur.

- **Passage par variable.**

L'échange de données se fait dans les deux sens :



Le paramètre formel subit obligatoirement une modification qui permettra un changement de la valeur du paramètre effectif.

Dans ce cas le paramètre formel doit obligatoirement être précédé par le mot clé **var.**

IV. Les fonctions

1. Définition

Une **fonction** est un sous-programme qui retourne un seul résultat de type simple. Ce type est celui de la fonction.

Remarque : Une fonction est un cas particulier d'une procédure.

2. Appel d'une fonction.

L'appel d'une fonction engendre le retour d'une valeur, qu'on peut :

- **L'affecter dans une variable :**

  Nom_Variable← FN nom fonction (param eff1, param eff2, …, param effn)

- **L'afficher directement :**

  écrire (FN nom fonction (param eff1, param eff2, …, param effn)=

- **La faire figurer dans une expression arithmétique ou logique :**

  si FN nom fonction (param eff1, param eff2, …, param effn)) alors

  …

3. Syntaxe.