CHAPITRE5

LES PROCÉDURES ET LES FONCTIONS

1.Problématique :

Les problèmes jusque là traités peuvent être classés dans la catégorie des problèmes faciles ; c'est à dire que leur résolution se ramène à une suite finie d'actions simples, qui, si elles sont correctement exécutées, fourniront le(s) résultat(s) souhaité(s).

En réalité, les problèmes rencontrés sont plus complexes et leur solution n'est pas immédiate : pour obtenir le résultat final, on doit passer par des résultats intermédiaires solutions de sous problèmes inclus dans le problème principal. C'est pour cette raison, que pour résoudre un problème, on doit le décomposer en sous problèmes. Si la complexité persiste, on continuera la décomposition jusqu'à arriver à un niveau de décomposition où tous les sous problèmes trouvés seront faciles à résoudre.

Ce concept de décomposition s'appelle ***analyse descendante*** ; en programmation, on parle de programmation procédurale. Ainsi, pour chaque sous problème on va lui associer son algorithme, lui même on lui fait correspondre un sous-programme.

On distingue alors deux catégories de sous-programmes : ***Les fonctions et les procédures.***

Dans le même ordre d'idées, Descartes avait dit : « Diviser les difficultés en autant de parcelles qu'il se peut afin de les mieux résoudre ».

A titre d'exemple un programme de gestion d'un restaurant peut être découpé en plusieurs modules gestion des tables, des clients, des employés, du stock, etc.

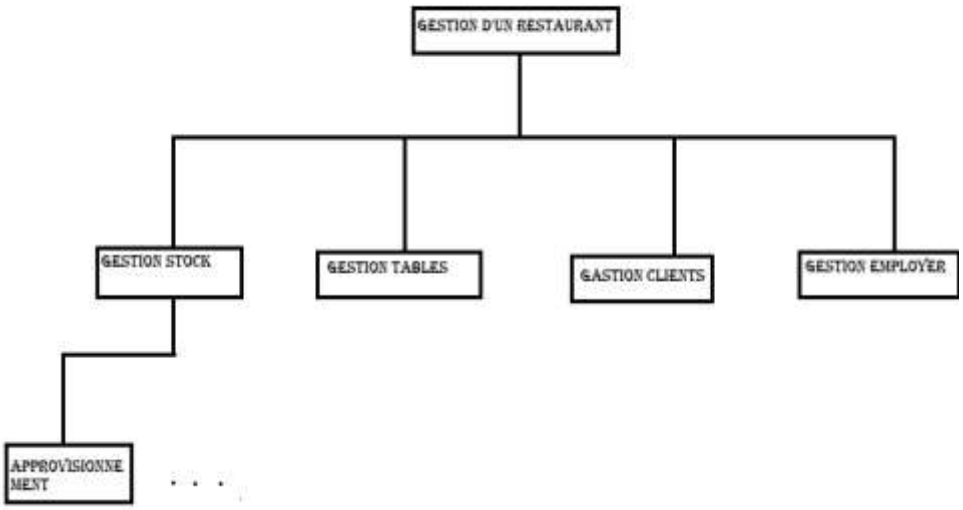

Figure 11- Décomposition du problème de gestion d'un restaurant en sous problèmes

On a donc décomposé l'algorithme en plusieurs sous algorithmes appelés modules. Le recours à ce type de décomposition permet de structurer la solution et de l'optimiser puisqu'un module peut être appelé une ou plusieurs fois.

Intérêt de la décomposition :

La décomposition d'un problème en sous problèmes faciles à résoudre, permet d'obtenir des algorithmes qui sont :

- Lisibles et faciles à comprendre.
- Facile à maintenir : détection rapide de l'erreur et correction sans difficultés.
- Facile à faire évoluer : ajout facile d'autres fonctionnalités.
- Réutilisable : dans la résolution d'un problème, on peut constater qu'une suite d'actions revient plusieurs fois. Dans ce cas, il serait judicieux de l'écrire une seule fois, et de l'utiliser autant de fois que c'est nécessaire.

2. Les procédures :

2.1 Définition:

Une procédure est une structure autonome qui permet que réaliser une tache bien précise et qui permet de transmettre un ou plusieurs résultats. Il existe deux types de procédure : procédure sans paramètres et procédure avec paramètres.

2.2 Procédure sans paramètres:

Syntaxe :

```
Procédure  Nom_proc

Variables

<déclaration des
variables>

Début

Bloc d'instructions

Fin
```

Remarques :

1- On remarque que la squelette d'une procédure est presque identique à celle d'un algorithme une procédure est elle aussi constituée de 3 parties : entête, partie déclaration et corps.

2- L'appel de la procédure sans paramètre peut se faire au niveau de l'algorithme principal ou au niveau d'une autre procédure ou fonction cet appel se fait tout simplement en écrivant le nom de la procédure.

3- Les variables déclarés au niveau de la procédure s'appellent des variables locales et ne sont utilisables que seulement à l'intérieur de celle-ci.

4- Si une variable est déclarée au niveau de l'algorithme principal on l'appelle variable globale celle-ci peut être utilisé au niveau de l'algorithme principale ainsi que les différents modules.

Exercice d'application 1

Ecrire un algorithme qui permet d'afficher un carrée d'" * " cet algorithme fait appel a une procédure Carree qui elle-même fait appel à une procédure Ligne qui permet de dessiné une ligne d'"*".

Correction

```
Algorithme Dessin
    Procédure Ligne
       Variables
        i: entier
```

```
Début
        Pour i de 1 à 10 Faire
          Ecrire ("*")
       Fin Pour
   Fin
   Procédure Carree
      Variables
      i: entier
       Début
          Pour i de 1 à 10 Faire
            Ligne
          Fin Pour
   Fin

Début
   Carree
Fin
```

2.3 Procédure avec paramètres:

Lors de l'appel d'une procédure l'algorithme principal ou autre module a souvent besoin d'échanger des informations avec cette dernière cet échange se fait à l'aide de paramètres.

Syntaxe

```
Procédure Nom_proc(paramètre1 : type, paramètre 2 : type, …)

Variables

<déclaration des variables>

Début

Bloc d'instructions

Fin
```

Remarques

1- Lors de la déclaration d'une procédure paramétré la seule différence avec la déclaration d'une procédure sans paramètres et qu'à la suite du nom de la procédure on ajoute une liste de paramètres qu'on appelle « paramètres formels ».

2- Lors de l'appel d'une procédure paramétrée on écrit le nom de la procédure suivit d'une liste de paramètres appelé paramètres effectifs qui remplacent les paramètres formels.
3- L'ordre des paramètres formels doit être respecté lors de l'appel avec les paramètres effectifs.
4- Les paramètres effectifs doivent avoir les mêmes types que les paramètres formels.

Exercice d'application 2 :

Refaire l'exercice précédent mais cette fois au lieu d'afficher un carée de dimension 10 "*" la dimension du carrée sera saisi au clavier au niveau de l'algorithme principal et passé comme paramètre à la procédure.

Correction :

```
Algorithme Dessin
    Variables
      N :entier
   Procédure Ligne(N :entier)
      Variables
      i: entier
     Début
          Pour i de 1 à N Faire
            Ecrire ("*")
        Fin Pour
    Fin
   Procédure Carree(N :entier)
      Variables
      i: entier
       Début
          Pour i de 1 à N Faire
            Ligne(N)
          Fin Pour
    Fin
Début
    Ecrire(donner la dimension du carrée )
    Lire(N)
   Carree(N)

 Fin
```

2.4 Mode de passage des paramètres:

La substitution des paramètres formels par des paramètres effectifs s'appelle passage de paramètres. Elle correspond à un passage d'information entre le module appelant et le module appelé.

Les paramètres d'une procédure peuvent être classés en 3 catégories :

1- Paramètres d'entrée ou (données) ils offrent les informations nécessaires au fonctionnement de la procédure. Leurs valeurs avant et après l'exécution de la procédure reste les mêmes. Exemple pour le calcul d'un salaire on a besoin de connaitre le nombre d'heures travaillées ainsi que le tarif de l'heure après le calcul du salaire ces deux données garderons les mêmes valeurs.

2- Paramètres de sortie ou (résultats) ils vont contenir les résultats de l'exécution de la procédure leurs valeurs initiales n'ont aucune importance. Le même exemple pour le calcul du salaire un paramètre sal va contenir le résultat du calcul et sa valeur initiale n'a aucune importance.

3- Paramètres d'entrée/sortie ou (données/résultats) lors de l'appelle de la procédure ils auront des valeurs initiales après l'exécutions de la procédure leurs valeurs initiales peuvent changer. Exemple on suppose qu'on connait le prix d'un article et qu'on va réaliser une réduction sur cet article de 10%. La procédure remise ; on va lui passer comme paramètres le prix de l'article ainsi que la valeur de la remise après l'exécution de cette procédure dans le paramètre prix on ne va plus trouver l'ancien prix mais le pris après la remise.

Pour chaque type de paramètres il y'a un mode de passage bien définies le tableau ci-dessous indique pour chaque type de paramètres son mode de passage de paramètres.

Type de paramètre	***Mode de passage de paramètre***
Entrée	Passage par valeur
Sortie	Passage par variable ou par adresse
Entrée/Sortie	Passage par variable ou par adresse

- Le mode de passage par valeur signifie que le paramètre garde la même valeur initiale.
- Le mode de passage par variable signifie que le contenu du paramètre peut être modifié après l'exécution de la procédure.

- Pour distinguer entre un paramètre passé par variable et un paramètre passé par valeur on ajoute le mot var devant le paramètre passé par variable.

Exemple :

Procédure Nom_proc (paramètre 1 :type, var paramètre 2 :type, etc…).

Exercice d'application 3 :

Ecrire un algorithme qui calcul le salaire d'un employer cet algorithme fait appel à une procédure calcul.

Correction :

```
Algorithme Salaire
      Variables
      Nh,Th,sal :réel
      Procédure Calcul (n, t :réel, var s: réel)

           Début
            s← n*t
           Fin
    Début
      Ecrire ("donner le nombre d'heures travaillées")
      Lire(Nh)
      Ecrire ("donner le tarif de l'heure")
      Lire(Th)
     Calcul(Nh,Th,sal)
     Ecrire ("le salaire est",sal)
    Fin
```

Exercice d'application 4 :

Ecrire un algorithme qui calcul le prix d'un article après une remise.

Correction :

```
Algorithme Prix
   Variables
    prix, remise : réel
   Procédure Calcul (r : réel, var p: réel)
     Début
      p← p-(p*r)
     Fin

  Début
    Ecrire ("donner le nombre de l'article")
    Lire (prix)
    Ecrire ("donner la remise ")
    Lire (remise)
   Calcul (remise, prix)
   Ecrire ("le nouveau prix est",prix)
  Fin
```

Remarque :

- Dans le mode de passage par valeur les modifications faites sur les paramètres formels n'affectent pas les valeurs des paramètres effectifs contrairement au cas où les paramètres sont passés par variables.

Exemple :

Procédure Permut1(a, b : entier)	Procédure Permut2(var a, b : entier)
Variables c : entier Début c← b b← a a← c Fin	Variables c : entier Début c← b b← a a← c Fin

- après l'exécution de Permut 1 a et b garderont leurs valeurs initiales alors qu'après l'exécution de Permut 2 les valeurs de a et b seront permutées.

3. Les Fonctions :

3.1 Définition:

Une fonction est un sous-programme contenant un certain nombre d'instructions, qui retourne un résultat unique affecté à son nom. On peut dire autrement : Une fonction est un sous programme qui retourne obligatoirement une valeur. Cette dernière sera stockée dans une variable qui porte le même nom que la fonction.

Une fonction possède un type qui est celui de son résultat.

Contrairement à une procédure dans une fonction le mode de passage de paramètre se fait toujours par valeur.

Syntaxe :

```
Fonction Nom_fonct(paramètre1 : type, paramètre2 : type,
etc…) :Type
Variables
<liste des variables>
Début
Block instructions
Nom_fonct←résultat
Fin
```

Exercice d'application 5 :

Ecrire une fonction qui calcul le factoriel d'un entier n.

Correction :

```
Fonction Factoriel (n :entier) :entier
    Variables
     i, f : entier
    Début
      f←1
        Pour ide 2 à n Faire
          f←f*i
        Fin Pour
       Factoriel←f
      Fin
```

Exercice application 6 :

Ecrire une fonction qui calcul la somme de deux entiers a et b.

Correction :

```
Fonction Somme (a, b : entier) : entier
    Début
      Somme←a+b
    Fin
```

Remarque :

L'appel d'une fonction se fait par son nom suivit de la liste des paramètres effectifs il est toujours réalisé dans une expression et non seul comme pour les procédures.

Exemples

- x ←Somme (a, b) ; x doit être de type entier le même type que la fonction Somme.
- Ecrire ("la somme de a et b est ",Somme(a, b)).