Chapitre 4 : **Les sous programmes (Procédures et Fonctions)**

1. Introduction

- Lorsque l'on progresse dans la conception d'un algorithme, ce dernier peut prendre une taille et une complexité croissante. De même des séquences d'instructions peuvent se répéter à plusieurs endroits.
- Un algorithme écrit de cette façon devient difficile à comprendre et à gérer dès qu'il dépasse deux pages ⇨ La solution consiste alors à découper l'algorithme en plusieurs parties plus petites. Ces parties sont appelées des **sous-algorithmes**.
- Le sous-algorithme est écrit séparément du corps de l'algorithme principal et sera appelé par celui-ci quand ceci sera nécessaire.
- Il existe deux sortes de sous-algorithmes (programmes) qui sont: les procédures et les fonctions.

2. Les procédures

- Une procédure est un sous-programme identifié par *un nom*
- il peut être appelé dans *un autre programme* ou dans des différents lieux du même programme ou même dans un autre s*ous-programme*.
- il permet d'effectuer des actions par un simple appel comme une instruction en utilisant des données différentes.
- Une procédure renvoie plusieurs valeurs (par une) ou aucune valeur.

2.1 Déclaration d'une procédure :

Syntaxe :

```
Procédure nom_proc(liste et déclaration de paramètres);
Variables identificateurs : type ;

 Début
 |   Instruction(s) ;
 |
 FIN
```

- Une procédure a la même structure qu'un programme.
- Après le nom de la procédure, il faut donner la liste des paramètres (s'il y en a) avec leur type ***respectif.***
- Ces paramètres sont appelés ***paramètres formels***. Leur valeur n'est pas connue lors de la création de la procédure.

2.2 L'appel d'une procédure

- Pour déclencher l'exécution d'une procédure dans un programme, il suffit de l'appeler.
- L'appel de procédure s'écrit en mettant le *nom de la procédure*, puis la *liste des paramètres*, séparés **par des virgules**.
- A l'appel d'une procédure, le programme ***interrompt son déroulement*** normal, exécute les instructions de la procédure, puis ***retourne au programme*** appelant et exécute ***l'instruction suivante*****.**

Syntaxe :

```
Nom_proc (liste de paramètres) ;
```

Remarque :

- Les paramètres utilisées lors de l'appel d'une procédure sont appelés ***paramètres effectifs***.
- Ces paramètres donneront leurs valeurs ***aux paramètres formels***.
- Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de ***la partie principale*** (algorithme principal)**.**

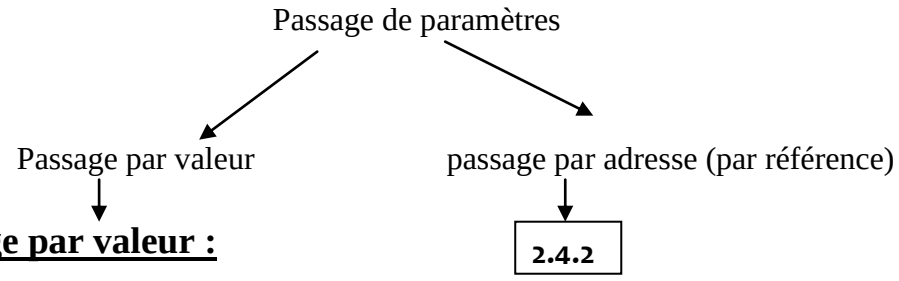
2.3 Notions de variables globales et de variables locales

En programmation informatique :

- une ***variable globale*** est une variable **déclarée** à ***l'extérieur*** du corps de toute ***Fonction*** ou **procédure** et pouvant donc être utilisée ***n'importe où*** dans le programme.
- une ***variable locale*** est une variable qui **ne peut être utilisée** que dans la ***Fonction*** ou le bloc où elle **est définie**.
- La variable **locale s'oppose** à la variable **globale** qui peut être utilisée dans **tout** le programme.

2.4 Passage de paramètres

- Les échanges d'informations entre une procédure et le sous algorithme appelant se font par ***l'intermédiaire de paramètres***.
- Il existe ***deux principaux types*** de passages de paramètres qui permettent des usages différents.



2.4.1 Passage par valeur :

- Dans ce type de passage, le paramètre ***formel*** reçoit uniquement ***une copie de la valeur*** du ***paramètre effectif***.
- La valeur du paramètre effectif ne sera jamais modifiée.
- Exemple : Soit l'algorithme suivant :

```
AlGORITHME Passage_par_valeur ;
Variables N : entier
Procédure P1(A : entier) //*Déclaration de la procédure P1
Début
| A ← A * 2 ;
| écrire(A) ;
FinProc

Début                          //*Algorithme principal
| N ← 5 ;
| P1(N) ;
| écrire(N) ;
Fin
```

Questions : faire le tracé d'exécution de cet algorithme ?qu'est-ce que vous remarquez. ?

Explication : Cet algorithme définit une ***procédure P1*** pour laquelle on utilise le passage de paramètres ***par valeur***.

- Lors de l'appel de la procédure, la valeur du paramètre **effectif N** est **recopiée** dans le **paramètres formel A**.
- La procédure effectue alors le traitement et affiche **la valeur** de la **variable A**.
- Dans ce cas 10. Après l'appel de la procédure, l'algorithme affiche la valeur de la variable N.
- Dans ce cas **5**. La **procédure ne modifie** pas le paramètre qui **est passé par valeur**.

2.4.2 Passage par référence ou par adresse :

Dans ce type de passage :

- la procédure ***utilise l'adresse du paramètre effectif***.
- Lorsqu'on utilise l'adresse du paramètre, ***on accède directement à son contenu***.
- La valeur de la variable effectif sera *donc* ***modifiée***.
- Les paramètres passés par adresse sont précédés du mot **clé *Var***.

Exemple : Reprenons l'exemple précédent

```
ALGORITHME Passage_par_référence ;
Variables N : entier
//*Déclaration de la procédure P1
Procédure P1 (Var A : entier)
Début
|   A ⟵ A * 2
|   écrire(A)
FinProc
//*Algorithme Principal
Début
|   N ⟵ 5
|   P1(N)
|   écrire(N)
Fin
```

Question : faire le tracé d'exécution de cet algorithme ?qu'es que vous remarquez ?

Justification :

- A l'exécution de la procédure, l'instruction ***ecrirer(A)*** permet d'afficher à l'écran **10**.
- Au retour dans *l'algorithme* principal, l'instruction ***ecrirer(N)*** affiche également **10**.

- Dans cet algorithme le paramètre passé correspond à la ***référence (adresse)*** de la variable N.
- Elle est donc modifiée par l'instruction : A ← A ***2**
- **Remarque importante:** Lorsqu'il y a ***plusieurs paramètres*** dans la définition d'une procédure**,** il faut ***absolument*** qu'il y en ait ***le même nombre*** à l'appel et que ***l'ordre*** *soit* ***respecté.***

EXEMPLE :

```
PROCEDURE somme (x,y :entier) ;
    Variables
       S :entier ;
Début
   S ← x + y ;
Fin
```

- Pour appeler la *procédure somme* de cet exemple, on écrirait :

☐ Somme (2.5 ,7) → faux (le type des paramètres effectifs doit être entier).

☐ Somme (2, 5, 9) → faux (le nombre des paramètres effectifs doit être 2 et non pas 3).

☐ Somme (2, 5) → juste (le type et le nombre des paramètres sont adéquats).

3. LES FONCTIONS

Les Fonctions sont des sous algorithmes admettant des paramètres et retournant un seul résultat (une seule valeur) de type simple qui peut apparaître dans une expression, dans une comparaison, à la droite d'une affectation, etc.

3.1 Les types de fonction :

- Une fonction intégrée dans une bibliothèque pouvant être appelé depuis n'importe quel programme principal pour calculer une valeur**.**
- ***Le type de cette fonction peut être prédéfinis*** qui est entier, réel, booléenne, caractère, chaine de caractère.

3.2 Déclaration d'une fonction

Syntaxe :

```
Fonction nom-de-fonction (Déclaration Des Paramètres):Type du résultat;
```

Exemple : Définir une fonction qui renvoie le plus grand de deux nombres différents**.**

Solution :

```
  //*Déclaration de la fonction Max
Fonction Max(X: réel, Y:réel) : réel ;
Début
    Si X > Y Alors
        ecrire(X)
    Sinon
        ecrire(Y)
    FinSi
Fin
```

3.3 Appel d'une fonction dans l'algorithme principal

- On utilise une fonction dans une expression ou un appel de procédure, exactement ***comme une variable***, mais en indiquant *ses paramètres* entre parenthèses et séparés par des virgules pour nous renvoyer un seul résultat.
- pour renvoyer la valeur calculée au l'algorithme ou programme principal, il faut écrire l'instruction suivante:

```
NomDeFonction ← Valeur Du Resultat;
```

Exemple : Ecrire un algorithme appelant, utilisant la fonction Max de l'exemple précédent.
Solution :

```
Algorithme Appel_fonction_Max
    Variables A, B, M : réel
    // * Déclaration de la fonction Max
    Fonction Max(X: réel, Y: réel) : réel
    DEBUT
        Si X > Y Alors
            max(x,y) ← X
        Sinon
            max(x,y)  ←  Y
        FinSi
    FIN

   //*Algorithme principal
    DEBUT
     Ecrire ("Donnez la valeur de A :")
     lire(A)
    Ecrire ("Donnez la valeur de B :")
     lire (B)

     //*Appel de la fonction Max
       M ← Max(A,B)

    Ecrire ("Le plus grand de ces deux nombres est : ", M)

   FIN
```

4. Portée des variables

- La portée d'une variable désigne le domaine de visibilité de cette variable.
- Une variable peut être déclarée dans deux emplacements distincts.
- Une variable déclarée dans la partie déclaration de l'algorithme principale est appelée variable globale :
  - Elle est ***accessible de n'importe où dans l'algorithme***, même depuis les procédures et les fonctions.
  - Elle existe pendant *toute* ***la durée de vie*** du programme.
- Une variable déclarée **à l'intérieur** d'une procédure (ou une fonction) est ***dite locale***.
- Elle n'est accessible qu'à la procédure au ***sein de laquelle elle définie***, les autres procédures ***n'y ont pas accès***.
- La durée de vie d'une variable locale est limitée à **la durée d'exécution** de sa procédure ou sa fonction.
- Il est possible de déclarer dans la procédure un identificateur (variable) utilisé dans un niveau englobant. Dans ce cas, ***la localité*** *masque* ***la globalité.***

```
Algorithme portée
 Variable x,y :entier ;
Procedure p1() :
  Variables
    A :entier
Debut
….
Finproc
//Algorithme principal
Début
  …..
Fin
```

```
X et Y sont des variables
globales  visibles dans tout
l'algorithme


A est une variable locale
visible uniquement à
l'intérieur de la
procédure
```

5. AVANTAGES DES PROCEDURES ET FONCTIONS

- ✓ Les procédures ou fonctions permettant *de* ***ne pas répéter plusieurs fois*** une même séquence d'instructions au *sein du programme (algorithme).*
- ✓ Structurent un algorithme en modules et augmentent *sa lisibilité* et sa *compréhensibilité*.
- ✓ facilitent la mise au point du programme (càd que *la compilation* et *la détection des erreurs* sera plus rapide en utilisant les procédures et/ou les fonctions).
- ✓ peuvent être même réalisées en dehors du contexte du programme, autrement dit, elles peuvent être rangées dans une bibliothèque d'outils et utilisées par n'importe quel autre programmes.