Chapitre IV

Les Sous programmes

I. Pourquoi les sous-programmes?

Dans un algorithme (ou programme informatique) on se retrouve souvent avec des bouts de code qui peuvent se répéter dans plusieurs emplacements. Cette redondance rend le code long, moins lisible et, par-dessus tout, si on souhaite modifier le comportement de ces blocs redondants alors on doit les réécrire tous, ce qui n'est pas du tout pratique.

Les sous-programmes permettent d'éviter cette redondance en factorisant le code qui peut se répéter. De cette façon nous n'aurons qu'un seul bloc de code que l'on peut réutiliser à volonté dans plusieurs endroits sans être obligé de le réécrire.

Les sous-programmes peuvent être des **fonctions** ou des **procédures**.

Une **fonction** est un sous-programme qui **retourne un seul résultat au moment de son appel**, alors qu'une **procédure ne retourne pas de résultats au moment de son appel**.
Pour utiliser un sous-programme, il faudrait la **définir**, puis **l'appeler**.

Exemples :

1. Écrire un programme qui calcule la formule suivante:

$$(x^2 + n!)/x^n$$

Nous notons que cette formule est constituée d'une fonction **puissance** et d'une **fonction factoriel**

2. Écrire un programme qui calcule la somme suivante:

$$\sum_{i=1}^{n} 2*i + n!$$

II. Définition d'un sous programme

Les fonctions et les procédures sont constitués des blocs suivants :

- un bloc formé du **nom** et **éventuellement des arguments**.
- un bloc pour déclarer les variables (qui ont souvent une portée locale),
- un bloc qui représente le **corps du sous programme**

A. Le bloc nom et éventuellement des arguments

1. Le nom d'un sous-programme

Le **nom** d'un sous programme représente son identificateur dans le programme donc il respecte toutes les règles de nommage d'un identificateur en algorithmique.

2. Les arguments d'un sous-programme

Les arguments ou paramètres formels d'un sous programme constituent les données à fournir au sous programme avant son exécution et éventuellement les résultats qu'il peut produire après son exécution.
On distingue trois types de passage de paramètres formels :

- Les **données(D) ou Entrée (E)**
- Les **Résultats(R) ou Sortie (S)**
- Les **données/Résultats(D/R) ou Entrée /Sortie(E/S)**

NB : Les échanges d'informations entre un sous programme et ses sous programme ou programme appelant se font par l'intermédiaire de paramètres formels.

a) Le Type passage par Donné(D)

Les paramètres de type **Donné** sont des variables dont les valeurs sont obligatoires pour l'exécution du sous programme.

Le type passage par donnée est aussi appelé **passage par copie de valeur.**

NB : Les valeurs des variables de type Donné sont non modifiables lors de l'exécution du sous programme .

b) Le Type passage par Résultat(R)

Les paramètres de type **Résultat** sont des variables utilisées par le sous programme pour stocker ses résultats produits lors de son exécution.

NB : Les variables de type résultat ont des valeurs qu' après l'exécution du sous programme .

c) Le Type passage par paramètres Donnee/ Resultat(D/R)

Les paramètres de type **Donné/Resultat** sont des variables qui stockent des valeurs nécessaires pour l'exécution du sous programme et qui peuvent être utilisées par le sous programme pour stocker des résultats.

Le type passage par **Donné/Resultat** est aussi appelé **passage par adresse .**

NB : Les valeurs des variables de type Donné/Resultat sont modifiables lors de l'exécution du sous programme .

Exemples : Donnez un nom et la liste des paramètres formels des formules définies ci-dessus.

3. Les variables locales d'un sous programme

Les variables locales d'un sous programme représentent l'ensemble des variables à déclarer pour réaliser les instructions du sous programme .

Exemples : Déclarer les variables des formules définies ci-dessus.

4. Le corps du sous programme

Le corps du sous programme regroupe toutes les instructions que devra exécuter le ce sous programmes au cours de son exécution.

III. Notion de fonction

Les **fonctions** sont des sous programmes 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.

Pour utiliser une fonction il faudra d'abord le définir puis l'appeler.

A. Syntaxe de définition d'une fonction

Syntaxe :

```
Fonction nom_Fonction (liste de paramètres) : type
Variables identificateur : type
Début
        Instruction(s)
        Retourner Expression
Fin
```

NB :
Les paramètres sont facultatifs, mais s'il n'y pas de paramètres, les parenthèses doivent rester présentes.

Exemple :

1. Définir une fonction qui renvoie la parité d'un nombre .
2. Définir une fonction qui renvoie le signe d'un nombre.

B. L'appel d'une fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivi des paramètres effectifs ou réels.Ces paramètres donnent leurs valeurs aux paramètres formels.

A l'appel d'une fonction, le programme interrompt son déroulement normal, exécute les instructions de la fonction, puis retourne le résultat au programme appelant et exécute l'instruction suivante.

NB:
L'appel d'une fonction pourra donc être utilisé dans une instruction (affichage, affectation, ...) qui utilise sa valeur.

Syntaxe

```
variable :type
          variable= Nom_Fonction (list de paramètres)
```

Exemple :

1. Faire l'appel des fonctions définies ci dessus

Remarques :

1. Lorsqu'il y a plusieurs paramètres Donnee ou Donnee/Resultat dans la définition d'une fonction , il faut absolument qu'il y en ait le même nombre à l'appel et que l'ordre soit respecté.

2. Pour exécuter un programme qui contient des sous programmes , il faut commencer l'exécution à partir de la partie principale (programme principal)

3. Lors de la conception d'un programme deux aspects apparaissent :
   - La définition (déclaration) de la procédure ou fonction.
   - L'appel de la procédure ou fonction au sein de l'algorithme principal.

Exercices Application :

Exercice 1:

Définir une fonction qui renvoie le plus grand de deux nombres différents puis l'appeler dans le programme principal.

Exercice 2:

Définir une fonction qui vérifie si un nombre est premier ou pas puis l'appeler dans le programme principal.

Exercice 3:

Soit la formule suivante:

$$M=(x^2 + n!)/x^n$$

Définir les fonctions ci dessous :

a. une fonction puissance
b. une fonction factoriel
c. une fonction qui calcule le résultat de la formule M
d. Définir le programme principal

Exercice 4:

Soit la formule suivante:

$$F = \sum_{i=1}^{n} 2*i+n!$$

Définir les fonctions ci dessous :

a. une fonction qui calcule le résultat de la formule F
b. Définir le programme principal

IV. Notion de Procédure

Une procédure est une série d'instructions regroupés sous un nom, qui permet d'effectuer des actions par un simple appel de la procédure dans un programme ou dans un autre sous-programme.
Une procédure ne renvoie pas de résultat.

Pour utiliser une fonction il faudra d'abord le définir puis l'appeler.

A. Syntaxe de définition d'une Procédure

```
Procédure nom_Procedure(liste de paramètres)
            Variables identificateurs : type
            Début
                      Instruction(s)
            FinRoc
```

NB :
Les paramètres sont facultatifs, mais s'il n'y pas de paramètres, les parenthèses doivent rester présentes.

Exemple :

```
Écrire une procédure qui affiche à l'écran une ligne de 15 étoiles puis passe à la ligne suivante.
```

B. 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_procedure(liste de paramètres)
```

Les paramètres utilisés lors de l'appel d'une procédure sont appelés paramètres effectifs.

Exemple :

```
2. Faire l'appel de la procédure définies ci dessus
```

Exercices Application :

Exercice 1:

En utilisant la procédure Étoiles déclarée dans l'exemple précédent, écrire un algorithme permettant de dessiner un carré d'étoiles de 15 lignes et de 15 colonnes.

Reprendre les exercices d'application des fonctions avec les procédures

V. Passage de paramètres

Les échanges d'informations entre les sous programmes se font par l'intermédiaire de paramètres.
Il existe deux principaux types de passages de paramètres qui permettent des usages différents :

A. 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.

B. Passage par référence ou par adresse :

Dans ce type de passage, la sous programme **utilise l'adresse** du paramètre effectif. Lorsqu'on utilise l'adresse du paramètre, **on accède directement à son contenu**. La valeur de la variable effective sera donc modifiée.

VI. 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:

- **Dans la partie déclarative du programme principal**
- **Dans la partie déclarative d'un sous programme**

A. Variables de portée globale

Une variable déclarée dans la partie déclaration de programme principale est appelée variable globale. Elle est accessible de n'importe où dans le programme , même depuis les procédures et les fonctions.
Elle existe pendant toute la durée de vie du programme.

B. Variables de portée locale

Une variable déclarée à l'intérieur d'un sous programme est dite locale. Elle n'est accessible qu'à de ce sous programme au sein de laquelle elle définie, les autres sous programmes n'y ont pas accès.
La durée de vie d'une variable locale est limitée à la durée d'exécution de la procédure.

Remarque :
Les variables globales sont à éviter pour la maintenance des programmes.

VII. La récursivité

Une procédure (ou une fonction) est dite récursive si elle s'auto-appelle.

Exemple :

Écrire une fonction récursive permettant de calculer la factorielle d'un entier positif.

Solution :

n != n * (n-1) ! : la factorielle de n est n fois la factorielle de n-1 :

```
//Déclaration de la fonction Factorielle (Fact)
Fonction Fact(n : entier) : entier
Début
        Si n > 1 Alors
                Retourner (fact(n-1)*n)
        Sinon
                Retourner 1
        FinSi
FinFonction
```

Exercice Application :

1. Écrire une fonction récursive permettant de calculer x a la puissance n.
2. Écrire une fonction récursive permettant la suite

$$U_n = U_{n-1} + U_{n-2} \text{ avec } U_{0=1} \text{ et } U_{1=1}$$

VIII. Série d'exercice

Exercice 1 :

Écrire un programme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul).

Exercice 2 :

Écrivez un programme permettant à l'utilisateur de saisir une série de N notes d'une classe.

Le programme, une fois la saisie terminée, renvoie:

1. Le nombre de notes saisies
2. La moyenne des notes.
3. La note la plus grande
4. La note la plus petites

NB : Une note est comprise entre 0 et 20.

Exercice 3 :

Écrivez un programme permettant à l'utilisateur de saisir un nombre N positif puis le décomposer en produit de facteur premier .

Exercice 4 :

Écrivez un programme permettant à l'utilisateur de saisir un nombre N en base de 10 le convertit en binaire.