Chapitre 3 : Procédures et fonctions

Dès qu'on commence à écrire des programmes importants, il devient difficile d'avoir une vision globale sur son fonctionnement et de traquer les erreurs

Exemple: Il nous est demandé d'écrire un algorithme qui calcul la partie entière de deux nombres réels puis d'afficher la plus grande valeur des deux avant de calculer la somme et la moyenne des deux nombres entiers obtenus.

- Que faire ? décomposer le problème en sous problèmes et trouver une solution à chacun puis regrouper le tout dans un seul algorithme

En Algorithmique, chaque solution partielle donne lieu à un sous-algorithme qui fera partie d'un algorithme complet pour pouvoir être exécuté.

1- Définitions

Un sous-algorithme est un bloc faisant partie d'un algorithme. Il est déclaré dans la partie entête (avant le début de l'algorithme) puis appelé dans le corps de l'algorithme.

Étant donné qu'il s'agit d'un bloc à part entière, il possède éventuellement un en-tête, une série de traitements, et une gestion des résultats tout comme l'algorithme qui le contient.

Notes :

a- Un sous-algorithme utilise les variables déclarées dans l'algorithme (appelées variables globales). Il peut aussi avoir ses propres variables (dites locales) déclarées dans l'espace qui lui est réservé ; mais qui ne peuvent être utilisées que dans ce sous-algorithme et nulle part ailleurs car sa portée (visibilité) est limitée au bloc qui la contient. L'espace de ces variables locales n'est réservé que lorsque le sous-algorithme est appelé et est libéré dès la fin de l'exécution.

b- Un sous-algorithme est déclaré de manière générale c.-à-d qu'il peut être appelé plusieurs fois avec différentes valeurs grâce à des arguments. Ces derniers, bien qu'ils soient facultatifs, sont dits paramètres et sont clairement déclarés, au besoin, dans l'entête du sous-algorithme.

Un paramètre est une valeur du bloc principal dont le sous-algorithme a besoin pour exécuter avec des données réelles l'enchaînement d'actions qu'il est chargé d'effectuer. On distingue deux types de paramètres :

- Les paramètres formels sont la définition du nombre et du type de valeurs que devra recevoir le sous-algorithme pour se mettre en route avec succès. On déclare les paramètres formels pendant la déclaration du sous-algorithme.

- Les paramètres effectifs sont des valeurs réelles (constantes ou variables) reçues par le sous-algorithme au cours de l'exécution du bloc principal. On les définit indépendamment à chaque appel du sous-algorithme dans l'algorithme principal.

c- L'exécution d'un sous-algorithme (procédure ou fonction) se fait par une instruction d'appel (voir sections suivantes). L'application de cette instruction génère un saut vers le sous-algorithme appelé. La terminaison de ce sous-algorithme redémarre la suite d'instruction interrompue par l'appel.

2- Types de sous-algorithme

Un sous-algorithme peut se présenter sous forme de fonction ou de procédure.
Une fonction est un sous-algorithme qui, à partir de donnée(s), calcul et **rend** à l'algorithme **Un et Un seul résultat** alors qu'en général, une procédure **affiche le(s) résultat(s) demandé(s)**.

2-1 Procédure

Une procédure est un bloc d'instructions nommé et déclaré dans l'entête de l'algorithme et appelé dans son corps à chaque fois que le programmeur en a besoin.

Déclaration d'une procédure :

```
Procédure Nom_Procédure (Nom_Paramètre : Type_praramètre;……) ;
Déclaration
        Nom_variable : Type_variable ;   }   Variables locales
        …
Début
|       …
|       Instructions ;                   }   Corps de la procédure
|       …
Fin ;
```

L'appel d'une procédure peut être effectué en spécifiant, au moment souhaité, son nom et éventuellement ses paramètres ; cela déclenche l'exécution des instructions de la procédure.

Exemple : Voici un algorithme utilisant une procédure qui calcule une somme de 100 nombres.

```
Algorithme essai;
Variable
        I, S : entier;
Procedure Somme ;
Debut /*Début de la Procédure*/
        S ←0 ;
        Pour I ← 1 a 100 Faire
                S ← S + i
        FinPour ;
        Ecrire ('La somme des 100 premiers nombres est', S) ;
Fin /*Fin de la Procédure*/
Debut /*Début de l'algorithme*/
        Somme
Fin. /*Fin de l'algorithme*/
```

2-2 Fonction

Une fonction est un bloc d'instructions qui retourne obligatoirement une et une seule valeur résultat à l'algorithme appelant. Une fonction n'affiche jamais la réponse à l'écran car elle la renvoie simplement à l'algorithme appelant.

Déclaration d'une fonction :

```
Fonction Nom_Fonction (Nom_Paramètre : Type_praramètre;……) : type_Fonction ;
Déclaration
        Nom_variable : Type_variable ;   }   Variables locales
        …
Début
        …
        Instructions ;                   }   Corps de la procédure
        …
        Nom_Fonction ← Résultat
Fin ;
```

Etant donné qu'une fonction a pour but principal de renvoyer une valeur, il est donc nécessaire de préciser le type de la fonction qui est en réalité le type de cette valeur.

Un appel de fonction est une expression d'affectation de manière à ce que le résultat soit récupéré dans une variable globale : *Nom_variable-globale ← Nom_Fonction (paramètres) ;*

Exemple : L'algorithme précédent, qui calcule une somme de N nombres, peut utiliser une fonction au lieu d'une procédure.

```
Algorithme essai;
Variable
        I, Som : entier;
Fonction Somme: entier ;
Variable
        S : entier ;
Debut  /*Début de la fonction*/
        S ←0 ;
        Pour I ← 1 a 100 Faire
                S ← S + I
        FinPour ;
        Somme ← S
 Fin /*Fin de la Fonction */
Debut /*Début de l'algorithme*/
        Som ← Somme ;
        Ecrire ('La somme des ', N, 'premiers nombres est', Som) ;
Fin. /*Fin de l'algorithme*/
```

Note : De même qu'une procédure, une fonction peut appeler d'autres sous-algorithmes à condition qu'ils soient définis avant elle ou qu'ils soient déclarés dans son entête.

3- Mode de passages de paramètres

Un sous-algorithme avec paramètres est très utile parce qu'il permet de répéter une série d'opérations complexes pour des valeurs qu'on ne connaît pas à l'avance. Il existe deux types de passage de paramètres : par valeur et par variable (dite aussi par référence ou encore par adresse).

3-1 Passage paramètres par valeur

C'est le mode de transmission par défaut, il y a **copie** de la valeur, des paramètres effectifs dans les variables locales issues des paramètres formels de la procédure ou de la fonction appelée.

Dans ce mode, le contenu des paramètres effectifs ne peut pas être modifié par les instructions de la fonction ou de la procédure ; car nous ne travaillons pas directement avec la variable, mais sur une copie. À la fin de l'exécution du sous-algorithme la variable conservera sa valeur initiale. Les paramètres dans ce cas sont utilisés comme données.

Syntaxe :

Procedure **nom_procédure** (**param1 :**type1 **; param2, param3 :**type2) ;
Fonction **<nom_fonction>** (**param1 :**type1 **; param2 :**type2) : Type_fonction ;

Exemple :

Soit l'algorithme suivant.

```
Algorithme pas-val ;
Déclaration
  M : entier ;
Procedure P1 (nombre : entier) ;
Debut
        Si nombre < 0 Alors
                nombre ← - nombre
        FinSi ;
        Ecrire (nombre)
Fin ;
Debut
        Lire (M) ;
        P1 (M) ;
        Ecrire (M)
Fin.
```

Exécutons cet algorithme pour la valeur (-6)

Avant l'appel de procédure : la seule variable déclarée est la variable globale (M)

M	Ecran
- 6	

Après l'appel de procédure : la variable-paramètre "nombre" est déclarée et reçoit en copie la valeur de M.

M	Nombre	Ecran
- 6	- 6	
- 6	6	
-6	6	6

Au retour à l'algorithme (au niveau de l'appel) il ne reste que la variable globale avec sa valeur initiale

M	Ecran
- 6	
-6	-6

3-2 Passage paramètres par variable

Ici, il s'agit non plus d'utiliser simplement la valeur de la variable, mais également son emplacement dans la mémoire (d'où l'expression « par adresse »). En fait, le paramètre formel se substitue au paramètre effectif durant le temps d'exécution du sous-programme et à la sortie il lui transmet sa nouvelle valeur.

Un tel passage de paramètre se fait par l'utilisation du mot-clé **Var**.

Syntaxe :

Procedure **nom_procédure** (Var **param1 :**type1, **param2, param3 :**type2) ;

Fonction **<nom_fonction>** (Var **param1 :** type1, **param2 :**type2) : Type_fonction ;

Note : Les paramètres passés par valeur et par adresse peuvent cohabiter à l'intérieur d'un même sous-algorithme. Il suffit de partager les deux types de passage par un (;).

Syntaxe :

Procedure **nom_procédure** (Var **param1 :**type1 **; param2, param3 :**type2) ;

Dans ce cas ***param1*** est passé par référence alors que les deux autres ont par valeur

Fonction **<nom_fonction>** (**param1 :**type1 **; Var param2 :**type2) : Type_fonction ;

Dans ce cas ***param1*** est passé par valeur alors que le deuxième est passé par valeur

Exemple :

Soit l'algorithme précédent modifié dans le type de passage de paramètre

```
Algorithme pas-val ;
Déclaration
  M : entier ;
Procedure P1 (Var nombre : entier) ;
Debut
        Si nombre < 0 Alors
                nombre ← - nombre
        FinSi ;
        Ecrire (nombre)
Fin ;
Debut
        Lire (M) ;
        P1 (M) ;
        Ecrire (M)
Fin.
```

Exécutons cet algorithme toujours pour la valeur (-6)

Avant l'appel de procédure : la seule variable déclarée est la variable globale (M)

M	Ecran
- 6	

Après l'appel de procédure : la variable-paramètre *nombre* se substitue à la variable *M*

(~~M~~) nombre	Ecran
- 6	
6	
6	6

Au retour à l'algorithme il ne reste que la variable globale avec sa nouvelle valeur.

M	Ecran
6	
6	6

4- Exemples

Exemple 1 :

Un algorithme qui calcule et affiche la valeur absolue d'une valeur en utilisant une fonction

```
Algorithme exemple1 ;
Declaration
        a, b : Entier ;
Fonction abs (unEntier : Entier) : Entier ;
Declaration
        valeurAbsolue : Entier ;
Debut
        si unEntier ≥ 0 alors
                valeurAbsolue ← unEntier
        sinon
                valeurAbsolue ← - unEntier
        finsi ;
        abs← valeurAbsolue
fin ;
Debut
        Ecrire ('Entrez un entier : ') ;
        Lire (a) ;
        b ← abs (a) ;
        Ecrire ('la valeur absolue de ', a, ' est ', b)
Fin.
```

Lors de l'exécution de la fonction ***abs***, la variable ***a*** et le paramètre ***unEntier*** sont associés par un passage de paramètre en entrée : La valeur de ***a*** est copiée dans ***unEntier***.

Exemple 2 :

Il est demandé d'écrire un algorithme qui demande à l'utilisateur d'entrer une valeur entière positive appelée (Valeur) puis

a. qui indique à l'utilisateur si Valeur est un nombre à 2 chiffres,

b. qui affiche la factorielle de Valeur,

c. et qui saisit des valeurs au nombre de Valeur puis affiche la plus grande valeur saisie.

C'est à vous de jouer sur ce coup !