# V. 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 d'un seul tenant 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 : les procédures et les fonctions.

## 2. Les procédures

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 algorithme ou dans un autre sous-algorithme.

Une procédure renvoie plusieurs valeurs (par une) ou aucune valeur.

### 2.1. Déclaration d'une procédure

⇒ ***Syntaxe :***

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

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.

⇒ ***Exemple :***

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

- ***Solution :***

```
Procédure Etoile()
Variables i : entier
Début
      Pour i Allant de 1 à 15 faire
         Afficher("*")
      FinPour
      //\n : retour à la ligne
      Afficher("\n")
FinProc
```

### 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)
```

Les paramètres utilisés lors de l'appel d'une procédure sont appelés paramètres effectifs. Ces paramètres donneront leurs valeurs aux paramètres formels.

⇒ ***Exemple :***

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

- ***Solution :***

```
Algorithme carré_étoiles
Variables j : entier

//Déclaration de la procédure Etoiles()
Procédure Etoile()
Variables i : entier
Début
  Pour i Allant de 1 à 15 Faire
     Afficher("*")
  FinPour
  Afficher("/n")
FinProc

//Algorithme principal (Partie principale)
Début
   Pour j Allant de 1 à 15 Faire
        //Appel de la procédure Etoiles
        Etoile()
   FinPour
Fin
```

⇒ ***Remarque 1 :***

Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de la partie principale (algorithme principal)

⇒ ***Remarque 2 :***

Lors de la conception d'un algorithme deux aspects apparaissent :

La définition (déclaration) de la procédure ou fonction.

L'appel de la procédure ou fonction au sein de l'algorithme principal.

### *2.3. Passage de paramètres*

Les échanges d'informations entre une procédures 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. 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 effectifs ne sera jamais modifiée.

⇒ ***Exemple :***

Soit l'algorithme suivant :

```
Algorithme Passage_par_valeur
Variables N : entier

//Déclaration de la procédure P1
Procédure P1(A : entier)
Début
   A ← A * 2
   Afficher(A)
FinProc

//Algorithme principal
Début
  N ← 5
  P1(N)
  Afficher(N)
Fin
```

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.

### *2.5. 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
   Afficher(A)
FinProc

//Algorithme Principal
Début
    N ← 5
    P1(N)
    Afficher(N)
Fin
```

A l'exécution de la procédure, l'instruction ***Afficher(A)*** permet d'afficher à l'écran 10. Au retour dans l'algorithme principal, l'instruction ***Afficher(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 :***

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.

## 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. Déclaration d'une fonction

⇒ ***Syntaxe :***

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

La syntaxe de la déclaration d'une fonction est assez proche de celle d'une procédure à laquelle on ajoute un type qui représente le type de la valeur retournée par la fonction et une instruction Retourner Expression. Cette dernière instruction renvoie au programme appelant le résultat de l'expression placée à la suite du mot clé Retourner.

⇒ ***Note :***

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

⇒ ***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
        Retouner X
    Sinon
        Retouner Y
    FinSi
FinFonction
```

### *3.2. L'appel d'une fonction*

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivie des paramètres effectifs. C'est la même syntaxe qu'une procédure.

A la différence d'une procédure, la fonction retourne une valeur. L'appel d'une fonction pourra donc être utilisé dans une instruction (affichage, affectation, ...) qui utilise sa valeur.

⇒ ***Syntaxe***

```
    Nom_Fonc(list de paramètres)
```

⇒ ***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
Début
    Si X > Y Alors
        Retourner X
    Sinon
        Retourner Y
    FinSi
```

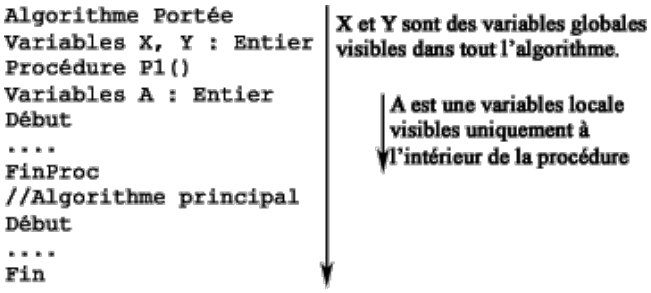
```
FinFonction

//Algorithme principal
Début
    Afficher("Donnez la valeur de A :")
    Saisir(A)
    Afficher("Donnez la valeur de B :")
    Saisir(B)
    //Appel de la fonction Max
    M ← Max(A,B)
    Afficher("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 la procédure.




⇒ ***Remarque :***

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

## 5. *La récursivité*

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

⇒ ***Exemple :***

Ecrire 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
```

Dans cet exemple, la fonction renvoie 1 si la valeur demandée est inférieur à 1, sinon elle fait appel à elle même avec un paramètre inférieur de 1 par rapport au précédent. Les valeurs de ces paramètres vont en décroissant et atteindront à un moment la valeur une (1). Dans ce cas, il n'y a pas d'appel récursif et donc nous sortons de la fonction.

⇒ ***Note :***

Toute procédure ou fonction récursive comporte une instruction (ou un bloc d'instructions) nommée "point terminal" permettant de sortir de la procédure ou de la fonction.

Le "point terminal" dans la fonction récursive Fact est : retourner 1.

## 1. *Avantages des procédures et fonctions*

Les procédures ou fonctions permettant de ne pas répéter plusieurs fois une même séquence d'instructions au sein du programme (algorithme).

La mise au point du programme est plus rapide en utilisant des procédures et des fonctions. En effet, elle peut être réalisée en dehors du contexte du programme.

Une procédure peut être intégrée à un autre programme, ou elle pourra être rangée dans une bibliothèque d'outils ou encore utilisée par n'importe quel programme.