Chapitre 5 (Algorithmique)

Les sous-programmes

I. Introduction

Informatiser une application c'est faire réaliser par un ordinateur, une tâche qui était effectuée par l'Homme.

Les étapes de résolution d'un problème :

1. Comprendre l'énoncé du problème.
2. Décomposer le problème en sous problèmes plus simple à résoudre.
3. Associer à chaque sous problème, une spécification (description).
   - Les données nécessaires,
   - Les données résultantes,
   - La démarche à suivre pour arriver aux résultats en partant d'un ensemble de données.
4. Implémenter les sous-programmes répondants à la spécification des différents sous problèmes
5. Coordonner le fonctionnement de tous les sous-programmes développés dans un programme principal.

II. L'analyse Modulaire

La conception d'un algorithme procède en général par des affinements successifs : On décompose le problème à résoudre en sous problèmes, puis ces derniers à leur tours, jusqu'à obtenir des sous problèmes « facile à résoudre », pour chaque sous problème on écrira un sous-programme.

La résolution du problème sera composée d'un algorithme principal et d'un certain nombre de sous-programmes. L'algorithme principal a pour but d'organiser l'enchaînement des sous-programmes.

Parmi les intérêts de la programmation modulaire nous pouvons citer :

- Répartir les difficultés,
- Faciliter la résolution d'un problème complexe,
- Pouvoir poursuivre l'analyse comme si les sous problèmes étaient résolus,
- Faciliter l'écriture de l'algorithme en évitant les duplications.
- Faciliter la localisation des erreurs.

III. Les Sous-programmes

Un sous-programme est une unité fonctionnelle formée d'un bloc d'instructions, nommée et éventuellement paramétrée. Que l'on déclare afin de pouvoir l'appeler par son nom en affectant s'il y a lieu des valeurs à ses paramètres.

Les données fournies au sous-programme et les résultats produits par ce dernier sont appelés des **arguments** ou des **paramètres**.

On distingue deux types de paramètres :

❶ **Paramètres formels** qui figurent lors de la définition du sous-programme, ces paramètres figurent dans l'entête de la définition du sous-programme, ils sont utilisés dans les instructions du sous-programme et là seulement. Ils correspondent à des variables locales.

❷ **Paramètres effectifs** Ils figurent dans l'instruction d'appel du sous-programme et sont substitués aux paramètres formels au moment de l'appel.

Remarque :

- Les paramètres effectifs et les paramètres formels doivent s'accorder du point de vue nombre et ordre.
- Leurs types doivent être compatibles selon le mode de passage des paramètres.

Un sous-programme peut être soit une procédure ou une fonction.

❶ **Une procédure** : Un sous-programme contenant un certain nombre d'instructions mais qui n'admet pas de valeur de retour.

❷ **Une fonction** : Un sous-programme contenant un certain nombre d'instructions et qui retourne un résultat unique.

III.1. Les Fonctions

III.1.a. Définition d'une Fonction

```
Fonction <nom_fonction>(<paramètres : types>) : <type_résultat>
        <Déclaration des objets internes>
Début
        <Instructions>
Fin
```

Exemple :

```
Fonction valeur_absolue(x : réel) : réel
Début
        si x > 0 alors retourner x
        sinon   retourner -x
        Finsi
```

Fin

III.1.b. Renvoi d'une valeur par une fonction

La fonction peut renvoyer une valeur (et donc se terminer) grâce au mot-clé **"retourner"**. Lorsque l'instruction "retourner" est rencontrée, la fonction évalue la valeur qui la suit, puis la renvoie au programme appelant (programme à partir duquel la fonction a été appelée).

Une fonction peut contenir plusieurs instructions "retourner", ce sera toutefois la première instruction "retourner" rencontrée qui provoquera la fin de la fonction et le renvoi de la valeur qui la suit.

La syntaxe de l'instruction "retourner" est :

Retourner <valeur ou variable ou expression>

Le type de valeur retourné doit correspondre à celui qui a été précisé dans le prototype (l'entête ou la signature) de la définition de la fonction.

La fonction valeur_absolue() peut être aussi écrite comme suit :

```
Fonction valeur_absolue(x : réel) : réel
Début
    si x > 0 alors
        retourner x
    Finsi
    retourner -x
Fin
```

III.1.c. Appel de Fonction

L'appel de fonction est une expression, dont la valeur est le résultat retourné par cette fonction, son appel s'effectue donc comme si on va évaluer une expression.

<Variable> ← <nom_fonct>([paramètres effectifs])

Ou encore

Écrire (<nom_fonct>([paramètres effectifs]))

Remarque : Une fonction peut ne pas avoir de paramètre, par exemple

```
Fonction unEntier( ) : réel
var nb : réel
Début
    Répéter
        Écrire (" Donner un réel");
        Lire (nb);
    Jusqu'à (nb>= 1 et nb <=100)
    retourner nb
Fin
```

Appel de la fonction unEntier() :

x ← unEntier()

Exemple d'utilisation des fonctions en algorithmique: Affichage de la valeur absolue d'un réel en utilisant la fonction "**Fonction** valeur_absolue(x : réel) : réel"

```
Algorithme absolue
Var v,y: réel
Fonction val_absolue(x : réel) : réel  /* Déclaration de la fonction */
   Var    a : réel   /* variable interne pour la fonction */
   Début
      si x > 0 alors        a ← x
      sinon       a ← -x
      Finsi
      Retourner(a)
   Fin
Début
Écrire(" Donner un réel ")
Lire(v)
y=val_absolue(v);      /* Appel de la fonction */
Écrire("la valeur absolue de ", v , "est:", y)
Fin
```

Paramètre formel

Paramètre effectif

Application : Écrire un algorithme qui permet d'afficher si un entier donné est premier ou non. Pour ce faire, utiliser une fonction "est_premier(x)" qui permet de faire le test de parité d'un entier x.

III.2. Les Procédures

III.2.a. Définition d'une Procédure

Procédure <nom_procédure>(<liste des paramètres>)

<Déclaration des objets internes>

Début

<Instructions>

Fin

Exemple :

Procédure Affiche_somme(a : entier, b : entier)

Var

S : entier

```
Début
        S ← a + b
        Écrire(S)
Fin
```

III.2.b. Appel de Procédure

L'appel à une procédure consiste à indiquer le nom de la procédure suivi d'un couple de parenthèses contenant la liste des paramètres effectifs nécessaires.

Exemple :

```
Algorithme affiche
        Var
        x, y : entier
Début
        Écrire(" Donnez un entier ")
        Lire(x)
        Écrire("Donnez un autre entier")
        Lire(y)
        Affiche_somme(x, y)
Fin
```

Application : Écrire une procédure qui permet d'afficher les entiers compris entre 1 et un certain entier strictement positif passé en paramètre.

IV. Passage de Paramètres

Ils existent deux modes de passage des paramètres : passage par valeur et passage par variable (appelé aussi par adresse).

IV.1. Passage par valeur *(appelé aussi passage ou transmission par copie)*

Dans ce mode de passage, le paramètre formel (décrit lors de la définition du sous-programme) prend une copie de la valeur du paramètre effectif (figurant au moment de l'appel) qui lui est associé. Par conséquent la valeur du paramètre effectif reste inchangée après l'exécution du sous-programme appelé.

Le passage par valeur consiste alors à recopier la valeur du paramètre effectif au sein du sous-programme . Par suite, ce sous-programme appelé s'exécute en utilisant ces valeurs.

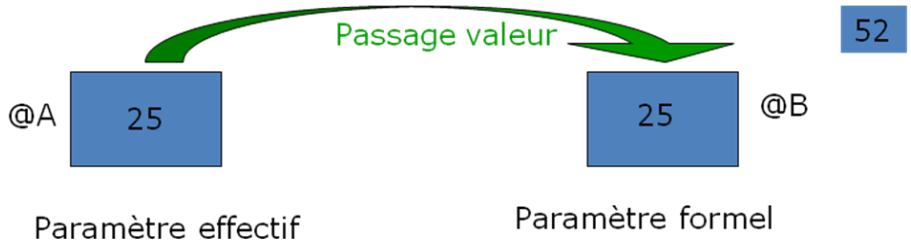

Figure 1 : Illustration du passage par valeur

- Si le paramètre formel change, le paramètre effectif associé ne change pas.
- Le paramètre formel est donc local dans le sous programme auquel il appartient.
- Le paramètre effectif doit posséder une valeur.

IV.2. Passage par variable *(dit aussi passage par adresse, par pointeur ou encore par référence)*

Dans ce mode de passage le paramètre formel prend l'adresse du paramètre effectif qui lui est associé. Le paramètre formel contient donc une référence vers le paramètre effectif et non pas sa valeur, par conséquent tout changement du contenu de cette référence affecte le paramètre effectif correspondant.
Un paramètre transmis par variable est toujours précédé par le mot clé **var** au moment de sa déclaration dans l'entête de la définition du sous-programme.

Dans le cas de la transmission par adresse, il n'y a plus copie d'une valeur, mais transmission de son adresse, donc de son emplacement dans le programme appelant. Dans ces conditions, le sous programme fonctionne, non plus sur une copie, mais sur la valeur d'origine elle-même. Elle peut donc éventuellement la modifier.
Lorsque les paramètres passent par variable, ils doivent être précédés par le mot VAR au moment de la déclaration du sous-programme.

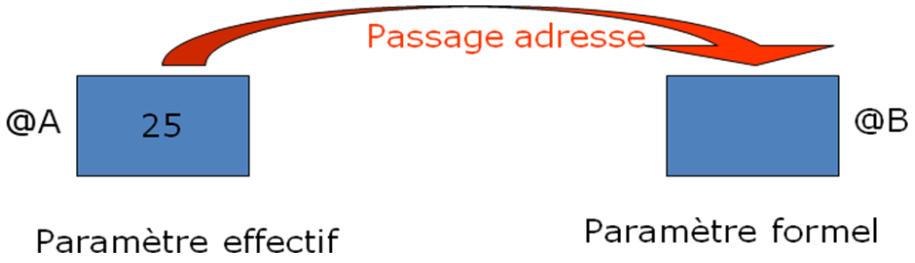

Figure 2 : Illustration du passage par adresse

- La variable B ne contient pas une copie de la valeur de A, elle contient l'adresse de la variable A.
- Modifier la variable B revient donc à modifier le contenu de la variable A.
- Le changement d'un paramètre formel provoque le changement du paramètre effectif associé.

Remarque : On détermine le mode de passage d'un paramètre **effectif** en consultant le paramètre **formel** correspondant. Il s'agit d'un passage de paramètre par variable si le paramètre formel est précédé de **VAR**, sinon c'est le mode de passage par valeur qui est utilisé.

Applications :

1- Donner le résultat d'exécution de ces deux algorithmes

Algorithme passage **Var** x : entier **Procédure** triple(i : entier) **Début** i ← i*3 **Fin** **Début** x←3 triple(x) Écrire(x) **Fin**	**Algorithme** passage **Var** x : entier **Procédure** triple(**var** i : entier) **Début** i ← i*3 **Fin** **Début** x←3 triple(x) Écrire(x) **Fin**

2- Écrire un algorithme qui permet de définir :

- Une procédure qui permet de lire deux entiers a et b strictement positifs
- Une procédure qui permet de permuter les valeurs de a et b
- Une fonction qui retourne le maximum de a et b
- Une fonction qui test si un entier est paire

Écrire l'algorithme utilisant ces sous programmes et qui permute a et b si a est paire, et affiche le maximum de a et b si a est impaire.