Chapitre 1 : Procédure et fonction

1. Méthodologie de programmation

Le principe de la programmation est de **Découper** le problème initial en ensemble de sous-problèmes plus simples, distincts et dont certains sont répétitifs et utilisés à différents endroits dans l'algorithme avec des données différentes.

Chacun de ces sous-problèmes devient un nouveau problème à résoudre, si on considère que l'on sait résoudre ces sous-problèmes, alors on sait quasiment résoudre le problème initial.

Cette technique de décomposition impliquera :

- Une meilleure qualité de programmation.
- Un code plus court.
- Maintenance facile et rapide.

L'algorithme sera constitué d'un ensemble de modules. Chaque module correspond à un sous-problème. Les modules créés sont appelés des sous-programmes.

On distingue deux types de sous-programmes :

- ☒ les fonctions et
- ☒ les procédures.

Les fonctions ne sont qu'un cas particulier des procédures.

2. Organisation d'un algorithme

L'algorithme est organisé en plusieurs parties :

- déclaration des constantes.
- déclaration des variables.
- définition des sous-algorithmes. (fonctions et procédures)
- définition de l'algorithme principal: L'algorithme principal consiste en une suite d'opérations élémentaires faisant souvent appel à des fonctions ou procédures.
- L'algorithme principal est délimité par les mots-clefs **DEBUT**et **FIN**

En-tête de l'algorithme

Déclarations
- ✓ Constantes
- ✓ Variables globales
- ✓ **Fonctions**
- ✓ **Procédures**

Algorithme principal
DEBUT
- ✓ Actions

FIN

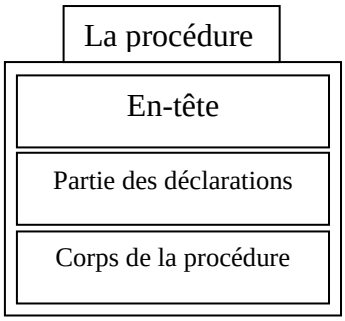
2. Procédure

2.1. Définition

Une procédure est une séquence d'instructions identifiée par un nom (le nom de procédure) et paramétrée, réalisant une tâche. Elle ne renvoie pas de résultat mais elle peut fournir un ou plusieurs résultats ou même aucun, ces résultats sont retourné à travers des paramètres. Une procédure peut avoir des paramètres qui peuvent être des paramètres d'entrée, de sortie ou à la fois d'entrée et de sortie.

2.2. Structure d'une procédure

Une procédure a la même structure qu'un programme principal :



En-tête

POCEDURE Nom_procédure ***(****param1****,*** *...****,*** *paramn*)

L'en-tête de la procédure permet d'identifier la procédure et de définir **les paramètres:** $param_1$ … $param_n$.

Exemple *:* PROCEDURE *exemple (x :* **ENTIER** *; y,z :* **REEL***)*

Les paramètres de la procédure sont un moyen de communication entre la procédure et l'extérieur (algorithme principal et les autres sous algorithmes).

Partie des déclarations

Dans cette partie on peut déclarer des variables, des constantes, des types et d'autres sous algorithmes. Tous ce qu'est déclaré dans cette partie est considéré locale, c-à-d on ne peut les manipulé que à l'intérieur de cette procédure et aussi dans les sous-algorithmes de cette procédure.

Corps de la procédure

La syntaxe est identique à celle du corps du programme principal.

2.3. Déclaration d'une procédure

Une procédure est un sous-algorithme qu'on déclare en général dans la partie des déclarations. La définition de procédure c'est l'écriture de son nom et de la séquences d'instructions qui la composer.

La définition d'une procédure se fait comme suit :

PROCEDURE <Nom_de_la_procédure> **[** (paramètres : type) **]:** Type_valeur_de_retour

VARIABLES : liste des variables locales

DEBUT {*corps de la procédure***}**

<Liste des actions>

FINPROCEDURE

Nom de la procédure

Une procédure a un nom qui est un identificateur unique (non utilisé pour une autre entité).

Les paramètres

Les arguments d'une procédure sont appelés ses paramètres. Le nom de la procédure est suivi de la liste des paramètres et de leurs types entre parenthèses.

La liste des paramètres est facultative. Mais quand elle existe, ces paramètres sont déclarés de la même façon qu'on déclare les variables de différents types.

Variables locales

Les variables déclarées à l'intérieur de la procédure sont inutilisables à l'extérieur de la procédure.

Corps de la procédure

Dans le corps de la procédure on peut écrire une séquence d'instruction qui réalisent la tâche de la procédure, on peut utiliser n'importe quelle instruction étudiée en algorithmique.

2.4. Appel de procédure

Lorsque la déclaration de la procédure est faite, il est possible d'appeler la procédure dans l'algorithme principal ou dans un autre sous algorithme pour déclencher l'exécution de la liste des actions définies à l'intérieur de la procédure. L'algorithme principal ou le sous algorithme qui appel une procédure s'appel "L'appelant".

Remarque

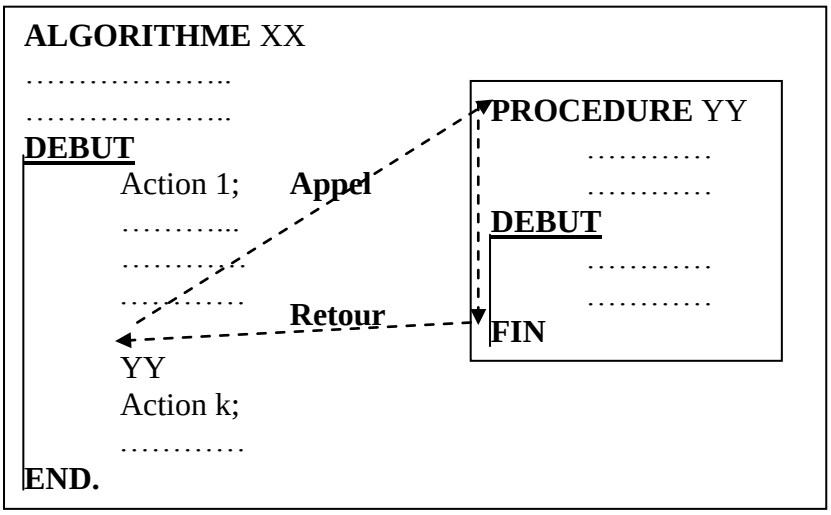
Une procédure peut appeler d'autres sous-algorithmes déjà définis.

Syntaxe de l'appel d'une procédure

Il suffit d'écrire le nom de la procédure suivi par les paramètres dans l'algorithme principal ou dans un autre sous algorithme**.**

Fonctionnement de l'appel d'une procédure

A la rencontre du nom de la procédure dans un algorithme, on dit qu'on a un appel de procédure. Il y à une rupture de séquence dans l'algorithme **appelant** et un branchement automatique est réalisé vers la procédure **appelée.**



Exemple: On utilise cet exemple dans tout le chapitre.

Ecrire un algorithme qui calculer la combinaison $C_N^P = \frac{N/}{P/(N-P)/}$

Solution1

Cette première solution sert à calculer la combinaison sans utilisation de sous algorithmes.

ALGORITHME Solution1
VARIABLE
N, P, Dif_NP, Fact_N, Fact_P, Fact_Dif_NP, Comb **: ENTIER**

DEBUT

LIRE (N, P)
Dif_NP ← N-P
Fact_N ← 1

TANTQUE (N > 1**)**
Fact_N ← Fact_N * N
N ← N – 1

FINTQ

Fact_P ← 1

TANTQUE (P > 1)
Fact_P ← Fact_P * P
PN ← P – 1

FINTQ

Fact_Dif_NP ← 1

TANTQUE (Dif_NP > 1)
Fact_Dif_NP ← Fact_Dif_NP * Dif_NP
Dif_NP ← Dif_NP – 1

FINTQ

Comb ← Fact_N / (Fact_P * Fact_Dif_NP)

ECRIRE ('la valeur de comnbinaison est =', comb)

FIN.

Solution2

On remarque dans la solution précédente qu'il y à des blocs d'instructions similaires qui ce fait sur des donnée différentes, il s'agit des calculs des factorielles de N, P et N-P. Alors dans la solution suivante, on utilise une seule procédure pour les trois blocs similaires, on change à chaque fois les données seulement.

Dans la deuxième solution on calcule la valeur de la combinaison utilisant une procédure sans paramètres.

```
ALGORITHME Solution2
VARIABLE
N, P, Dif_NP, Fact_N, Fact_P, Fact_Dif_NP, Combm K, Fact_K : ENTIER

        PROCEDURE  Factorielle
        DEBUT
        Fact_K ← 1
                TANTQUE (K > 1)
                        Fact_K ← Fact_K * K
                        K ← K – 1
                 FINTQ
        FIN

DEBUT
        LIRE(N, P)

        K ← N
        Factorielle             // Calculer N !
        Fact_N ← Fact_K

        K ← P
        Factorielle             // Calculer P !
        Fact_P ← Fact_K

        K ← N-P
        Factorielle             // Calculer (N-P) !
        Fact_Dif_NP ← Fact_K
        Comb ← Fact_N / ( Fact_P * Fact_Dif_NP )

        ECRIRE ('la valeur de comnbinaison est =', comb)
FIN.
```

Fonctionnement de l'algorithme

La procédure Factorielle() prend une valeur à partir de la variable K ensuite calculer son factorielle et enfin stocker le résultat dans la variable Fact_K.

Pour que l'algorithme principale exploiter la procédure Factorielle() pour calculer la factorielle de N, il faut suivre les étapes suivantes:

- il doit affecter la valeur N à la variable de réception de la procédure K et
- appeler la procédure par la citation de son nom et
- recevoir le résultat à partir de la variable Fact_K.

La même chose est effectuer avec les valeurs N et (N-p).

3. Fonction

3.1. Définition

Une fonction est un sous-algorithme destiné à effectuer un traitement sous forme d'un bloc d'instructions, nommé et paramétré et réalisant une tâche. Une fonction est un cas particulier de procédures ; contrairement à la procédure, la fonction doit avoir un type, c'est-à-d que celle-ci doit retourner une valeur.

a. **Structure d'une Fonction :**

Une fonction a pour but de calculer une valeur, appelée résultat de la fonction. Une fonction a la même structure similaire qu'un algorithme principal

En-tête

FONCTION *Nom_de_fonction* ***(****param1****,*** *...****,*** *paramn*) ***:*** *Type_resultat* ***;***

Dans l'en-tête de la fonction on doit identifier la fonction par un identificateur et définir éventuellement **les paramètres** de la fonction et en fin le type de la valeur calculée par la fonction "Type_Fonction".

Exemple

FONCTION *Exemple (x :* **ENTIER** *; y,z :* **REEL***) :* **ENTIER** *;*

Les paramètres

Les paramètres de la fonction sont un moyen de communication entre la fonction et l'algorithme principal ou les autres sous algorithmes.

Type de la valeur de retour

La valeur de retour d'une fonction peut être soit d'un type prédéfinie (caractère, entier, réel, booléen), doit d'un nouveau type définie dans l'algorithme.

Partie des déclarations

Même que la procédure, les variables déclarées dans une fonction sont appelées les **variables locales** de la fonction.

Corps de la fonction

La syntaxe est identique à celle du corps de l'algorithme principal.
C'est dans le corps de la fonction que la valeur de la fonction est calculée et affectée au nom de la fonction par l'instruction suivante:

`Renvoyer` résultat_des_calculs

3.2. Déclaration d'une fonction

Même qu'une procédure, une fonction est un sous-algorithme qu'on déclare dans la partie des déclarations.

La définition d'une fonction se fait comme suit :

FONCTION <Nom_de_la_fonction> **[** (liste des paramètres : type)] : Type_fonction
VARIABLES : liste des variables
DEBUT {*corps de la fonction***}**
<LISTE DES ACTIONS> {*la liste ne doit pas être vide*}
`Renvoyer` résultat_des_calculs
FIN FONCTION

Remarque

Le type d'une fonction peut être l'un des types suivant : ENTIER, REEL, CARACTERE, CHAINE ou POINTEUR uniquement.

3.3. Appel et retour d'une fonction

L'appel d'une fonction n'est pas une instruction mais fait partie d'une expression. Il consiste à nommer la fonction et à donner la liste des paramètres.

Lors de l'appel d'une fonction, la valeur retournée doit être exploitée dans une instruction (affectation, expression arithmétique, logique ou affichage).

Remarque

La fonction est considérée comme une variable.

Exemple

Prenant toujours l'exemple de calcul de combinaison

Solution3

Dans cette solution on proposer d'utiliser une fonction sans paramètres au lieu d'une procédure pour calculer la factorielle.

```
ALGORITHME Solution3
VARIABLE
N, P, Dif_NP, Fact_N, Fact_P, Fact_Dif_NP, Combm K, Fact_K : ENTIER
        FONCTION  Factorielle
        DEBUT
         Fact_K ← 1

            TANTQUE (K > 1)
                Fact_K ← Fact_K * K
                K ← K – 1
            FIN TANTQUE
        Renvoyer Fact_K
        FIN
DEBUT
        LIRE (N, P)

        K ← N
        Fact_N ← factorielle      // Calculer N !

        K ← P
        Fact_P ← Factorielle      // Calculer P !

        K ← N-P
        Fact_Dif_NP ← Factorielle      // Calculer (N-P) !

        Comb ← Fact_N / ( Fact_P * Fact_Dif_NP )

        ECRIRE ('la valeur de comnbinaison est =', comb)
FIN.
```

L'affectation du résultat au nom de la fonction

4. Les paramètres

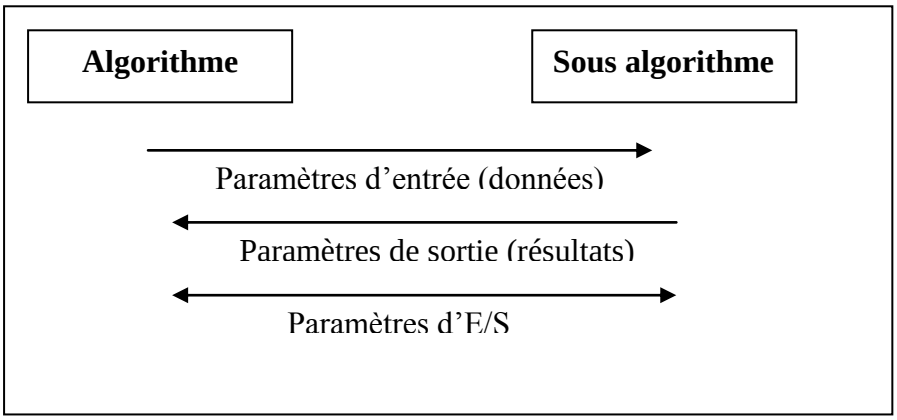
Les sous algorithmes et l'algorithme principal communiquent entre eux par des paramètres, c-à-d les paramètres permettent d'échanger les données entre les sous algorithmes et l'algorithme principal et pour cela, on distingue deux types de paramètres:

Paramètres d'entrée

Les paramètres d'entrée d'un sous algorithme (fonction ou procédure) sont les arguments du sous algorithme qui sont nécessaires pour effectuer le traitement associé à la fonction, les paramètres d'entrée servent à porter des données de l'appelant de la fonction vers la fonction.

Paramètres de sortie

Les paramètres de sortie d'un sous algorithme sont utilisés pour retourner les résultats calculés après avoir effectué le traitement associé au sous algorithme.



Remarque

La notion de paramètre est identique pour les procédure et fonctions

4.1. Types des paramètres

- Le type d'un paramètre ne peut être anonyme c-à-d il faut toujours spécifier le type du paramètre. **Exemple** Fonction Fact(K) :**ENTIER** → Erreur !!!
- Le type d'un paramètre est l'un des types suivant : **ENTIER** , **REEL**, **BOOLEEN**, **CARACTERE**, ou tout autre type défini explicitement par le programmeur.

4.2. Les paramètres formels et effectifs

Définitions

Paramètre formel

Un paramètre formel est un paramètre d'entrée d'une fonction utilisé à l'intérieur de la fonction appelée.

Paramètre effectif

Un paramètre effectif est un paramètre d'appel d'une fonction utilisé à l'extérieur de la fonction appelée.

Exemple

Soit l'écriture suivante :

Procédure X(I : ENTIER ; J : REEL; K, L : BOOLEEN)

On dit que I, J, K et L sont des paramètres formels de la procédure X.
Ces paramètres n'auront de valeurs qu'au moment où X est activée par un appel de procédure.

Soi l'algorithme suivant :

ALGORITHME Y
VARIABLES
……
PROCEDURE X(I : **ENTIER** ; J :**REEL** ; K, L : **BOOLEEN**)
……
DEBUT
……
FIN

DEBUT
U← 10.5
D← Vrai
S← Faux
X(1, U, D, S)
……
FIN.

Au moment de l'appel, les paramètres effectifs : 1, U, D, S sont associées dans l'ordre de leur apparition aux paramètres formels : I, J, K et L.

Remarques

- Le nombre de paramètres effectifs doit correspondre en **nombre**, en **ordre** et en **type** aux paramètres formels.
  L'appelle X(2.5, U, D, S) n'est pas correcte.
  L'appelle X(D, S) n'est pas correcte aussi.
- Un paramètre effectif n'a pas nécessairement le même nom que le paramètre formel.

4.3. Modes de passage de paramètres

Lors de l'échange de données entre sous algorithmes, on peut soit autoriser, soit interdire la modification des valeurs des paramètres. Il existe deux modes de passage de paramètres :

4.3.1. **Passage par valeur** (les données sont en entrée seulement) :

Les valeurs des paramètres sont utilisées sans qu'il soit possible de les modifier. On ne peut modifier les valeurs des paramètres effectifs lors de l'exécution du sous algorithme.

Lors de l'appel d'une procédure, il y a une copie des paramètres effectifs dans les paramètres formels: les valeurs des paramètres effectifs passés en argument sont affectées aux paramètres formels d'entrée correspondants. Alors, ne sont pas les paramètres effectifs qui sont manipulés par la fonction mais des copies de ces paramètres.

Remarque

Les noms des paramètres effectifs n'ont pas forcément les mêmes noms que les paramètres formels correspondants. Ils ne représentent pas la même variable en mémoire même s'ils ont le même nom.

Syntaxe

PROCEDURE Nom_procédure ***(p_1 : $type_1$; …; p_n : $type_n$;);***

Exemple

Soit l'algorithme suivant :

```
ALGORITHME Exemple1
VARIABLE
I : ENTIER
        PROCEDURE Add1(x : ENTIER)
        DEBUT

          X ← x + 1

        FIN PROCEDURE
DEBUT
    I ← 0
    Add1(I)
    ECRIRE(I)
FIN
```

Quelle valeur sera affichée ?
La valeur affichée ici est : 0.
Donc la valeur du paramètre effectif n'a pas été modifiée par la procédure Add1(x).

Explication

Au moment de l'appelle de la procédure Add1(x : **ENTIER**), une variable implicite est automatiquement crée (dans la procédure Add1(x)) ; la valeur du paramètre effectif I (ici 0) est attribuée (copiée ou affectée) à cette nouvelle variable.

L'incrémentation x← x+1 ce fait dans cette nouvelle variable et non pas dans le paramètre effectif I.

Nous disons ici que la transmission du paramètre effectif I ce fait par valeur (Appel par valeur)

Remarque

Lorsqu'on fait un appel par valeur, le paramètre effectif peut être :

- Un nom de variable.
  Exemple: VARIABLE y; Add1(y).
- Une constante.
  Exemple: CONSTANTE c; Add1(c).
- Une expression.
  Exemple: Add1(y+c-2)
- Une valeur.
  Exemple Add1(5).

4.3.2. Passage par adresse (par variable): (les données sont en entrée/sortie):

Les sous algorithmes appelés peuvent modifier les valeurs des paramètres transmis. Les modifications des valeurs effectuées dans l'algorithme appelé seront effectives au retour dans l'algorithme appelant.

Le passage des paramètres par adresse consiste à copier la référence (l'adresse) du paramètre effectif dans le paramètre formel correspondant.

Syntaxe

PROCEDURE Nom_procédure **(Var p_1 : $type_1$; …; Var p_n : $type_n$; …);**

Exemple : Soit maintenant l'algorithme suivant :

```
ALGORITHME Exemple2
VARIABLE
I : ENTIER
        PROCEDURE Add1(VARIABLE x : ENTIER)
        DEBUT
          X ← x + 1
        FIN
DEBUT
    I ← 0
    Add1(I)
    ECRIRE(I)
FIN.
```

Quelle valeur sera affichée ? La valeur affichée ici est : 1.

Donc la valeur de *I* à été modifiée, c-à-d la procédure Add1 opère ces instructions directement sur la variable *I* de l'algorithme principal.

Nous disons cette fois que la transmission du paramètre effectif *I* ce faite par référence (par variable) c-à-d toute référence au paramètre formel x dans la procédure Add1 sera en fait une référence au paramètre effectif *I.*

Remarque

Dans le cas d'un passage de paramètre par adresse seules des variables peuvent apparaître comme paramètres effectifs. (Par ce qu'il s'agit de transmettre la case (la variable) et non pas leur valeur.

Exemple :

Prenant toujours l'exemple de calcul de combinaison:

Solution4: Dans ce cas on utilise une procédure avec paramètres pour calculer la factorielle.

```
ALGORITHME Solution4
VARIABLES
N, P, Dif_NP, Fact_N, Fact_P, Fact_Dif_NP, Combm: ENTIER
        PROCEDURE  Factorielle(K :ENTIER ; VAR F : ENTIER )
        DEBUT
          F ← 1
              TANTQUE (K > 1)
                    F  ← F * K
                    K ← K – 1
               FINTQ
        FIN
```

```
DEBUT
        LIRE (N, P)
        Factorielle(N, Fact_N)                  // Calculer N !
        Factorielle(K, Fact_K)                  // Calculer P !
        Factorielle(N-P, Fact_Dif_NP)           // Calculer (N-P) !
        Comb ← Fact_N / ( Fact_P * Fact_Dif_NP )
        ECRIRE(Comb)
FIN.
```

Solution5:

Dans ce cas on utilise une fonction avec paramètres pour calculer la factorielle.

```
ALGORITHME Solution5
VARIABLES
N, P, Comb: ENTIER
            FONCTION Fact(K : ENTIER) : ENTIER
            VARIABLES    F : ENTIER
            DEBUT
              F ← 1
                  TANTQUE (K > 1)
                        F  ←  F *  K
                        K  ← K – 1
                  FINTQ
              Fact ← F
            FIN
DEBUT
        LIRE (N, P)
        Comb ← Fact(N) / ( Fact(P) * Fact(N-P) )
        ECRIRE (Comb)
FIN.
```

5. Portée des variables

5.1. Variables globale

Si l'on veut qu'une variable soit accessible par tous les sousalgorithmes, il faut la définir comme une variable globale.

Une variable globale est une variable déclarée dans la partie des déclarations de l'algorithme principal. Elle est utilisable dans n'importe quel sousprogramme sans nécessité de redéfinition.

5.2. Variables locale

Un sous algorithme peut avoir des variables internes (variables locales) qui ne sont pas visibles ni par les autres sous algorithmes ni par l'algorithme principal. Elles ne sont accessibles que par le sous algorithme où elles sont définies. Par conséquent, différents sous algorithme peuvent avoir des variables locales portant le même nom. Celles-ci n'auront pas la même signification pour chacun d'eux ; La portée d'une variable locale est le sous algorithme où elle est définie.

Remarques

- Si dans un sous algorithme il existe une variable qui porte le même nom que la variable globale, alors c'est cette variable locale qui sera considérée à l'intérieur du sous bloc.

- Cette notion de portée s'applique à tous les objets {constantes, types, variables, fonctions, procédures} définis ou déclarés à l'intérieur d'un sous algorithme {fonction ou procédure}.

Deux sous algorithmes différents peuvent utiliser des variables locales de même nom ou être définies avec des paramètres de même nom sans qu'il y ait ambiguïté ou conflit.

Exemple

Soit l'algorithme suivant:

```
ALGORITHME Exemple1
 VARIABLES x : ENTIER
        PROCEDURE modifie
        VARIABLES x : ENTIER
        DEBUT {début modifie}
            x := 1
        FIN {Fin modifie}
DEBUT {Début algorithme principal}
    x ← 0;
    modifie;
    ECRIRE(x);
FIN. {Fin algorithme principal}
```

Quelle valeur sera affichée ?

Cet Algorithme a deux variables appelées x, l'une est une variable globale et l'autre déclarée dans la procédure "modifie" est une variable locale à cette procédure. La valeur de x initialisée dans l'algorithme principal n'est pas "modifiée" par la procédure "modifie" car la variable locale empêche l'accès a la variable globale de même nom. Alors, l'algorithme affiche "0".

Remarque

On peut déclarer un sous algorithme dans un sous algorithme.

Exercice:

Exécuter l'algorithme suivant**:**

```
ALGORITHME Exemple 2
VARIABLES i, j, k : ENTIER
      PROCEDURE P(VARIABLE i  : ENTIER)
      DEBUT
        i ← i+1;
        ECRIRE(i,j,k)
      FIN;
              PROCEDURE Q(h : ENTIER; VARIABLE j : ENTIER)
              VARIABLES i : ENTIER
                      PROCEDURE R  {debut R}
                      DEBUT
                        i := i+1
                      FIN PROCEDURE;  {Fin R}
             DEBUT       {début Q}
                      i ←j ;
                      SI h = 0 ALORS     P(j)
                      SINON
```

```
                    SI h = 1 ALORS  P(i)
                    SINON  R
                    FINSI
                FINSI
                 ECRIRE (i,j,k)
            FIN PROCEDURE      {fin Q}
DEBUT       {Algorithme principal}
    i ← 0; j ← 1; k ← 2;  Q(0,k); Q(1,i); Q(2,j)
FIN. {Fin Algorithme principal}
```

6. Etapes d'écriture d'un sous algorithme

Pour écrire un sous algorithme dans un algorithme, on suivra la démarche suivante :

1. Déterminer le type du sous algorithme à utilisé; si notre sous problème retourne un et un seul résultat on utilise une fonction, sinon on utilise une procédure
2. Donner un nom au sous algorithme visée: Le nom du sous algorithme est un identificateur arbitraire (comme le nom d'un algorithme), de préférence assez courts et exprimer clairement ce que la fonction est censée faire
3. Définir les paramètres d'entrée du sous algorithme qui correspond aux données passées au sous algorithme.
4. Si le sous algorithme est une procédure, définir les éventuels paramètres de sortie de la procédure avec ses types, les paramètres de sortie correspond aux résultats trouvés par la procédure. S'il s'agit d'une fonction déterminé le type de la fonction qui correspond le résultat retourné par la fonction.
5. Ecrire le corps du sous algorithme: L'ensemble des instructions qui assurent la tâche du sous algorithme.
6. Utiliser le sous algorithme dans l'algorithme principal ou dan un autre sous algorithme: Appeler le sous algorithme dans un ou plusieurs endroits dans l'algorithme appelant.