Les fonctions

1
```
Fonction  somme ( a: int, b: int): int
    début
          retourne(a+b)

     finFonction
```

2
```
 Fonction  produit ( a: réel, b: réel, c: réel): réel
    début
          retourne(a*b*c)

   finFonction
```
3
```
Fonction  Min_2 ( a: int, b: int): int
    début
         si (a<b) alors retourne(a)
         sinon retourne(b)
          finsi
     finFonction
```
4
```
Fonction  est_divi_3 ( a: int): int
    début
         si (n%3==0) alors retourne (1)
         sinon retourne(0)
          finsi

      finFonction
```
5
```
Fonction  somme_imp ( n: int): int
   variables i, s=0 : int
 début
     pour i : 1 à n faire
       si (n%2!=0) alors
             s<- s+i
       finsi
     finPour
    retourne(s)
finFonction
```

Les procédures

- Dans le cas où une tâche se répète dans plusieurs endroits du programme et elle ne calcule pas de résultats ou qu'elle calcule plusieurs résultats à la fois alors on utilise une **procédure** au lieu d'une fonction.

- Une **procédure** est un sous-programme semblable à une fonction mais qui **ne retourne rien.**

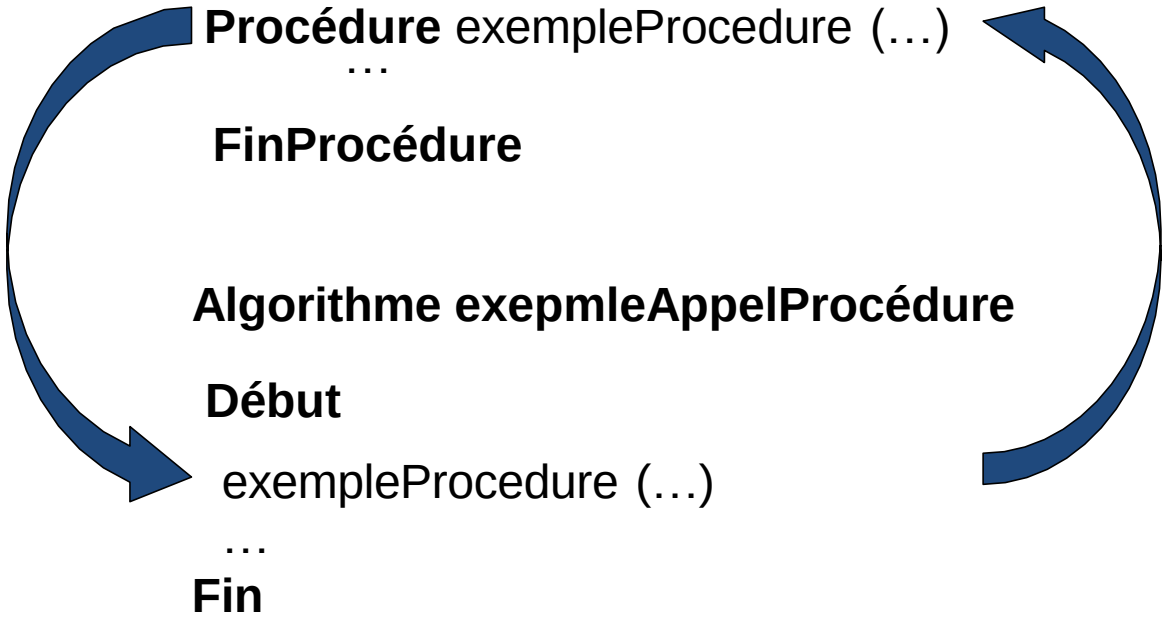
- Une procédure s'écrit en dehors du programme principal sous la forme :

```
Procédure nom_procédure (paramètres et leurs types)
Variables //locales
Début
   Instructions constituant le corps de la procédure
FinProcédure
```

- Remarque : une procédure peut ne pas avoir de paramètres

Les procédures

- Pour appeler une procédure dans un programme principale ou dans une autre procédure, il suffit d'écrire une instruction indiquant le nom de la procédure :



Remarque :

- contrairement à l'appel d'une fonction, on ne peut pas affecter la procédure appelée ou l'utiliser dans une expression.

- L'appel d'une procédure est une instruction autonome.

Les procédures

Paramètres d'une procédure

- Les paramètres servent à échanger des données entre le programme principale (ou la procédure appelante) et la procédure appelée.

- Les paramètres placés dans la déclaration d'une procédure sont appelés **paramètres formels**. Ils sont des variables locales à la procédure.

- Les paramètres placés dans l'appel d'une procédure sont appelés **paramètres effectifs**. ils contiennent les valeurs pour effectuer le traitement.

- Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels.
- L'ordre et le type des paramètres doivent correspondre.
- Il existe deux modes de transmission de paramètres dans les langages de programmation :

  La transmission par valeur et la transmission par adresse.

Les procédures

Transmission des paramètres

- **La transmission par valeur** : les valeurs des paramètres effectifs sont affectées aux paramètres formels correspondants au moment de l'appel de la procédure. Dans ce mode le paramètre effectif ne subit aucune modification.

- **La transmission par adresse (ou par référence)** : les adresses des paramètres effectifs sont transmises à la procédure appelante. Dans ce mode, le paramètre effectif subit les mêmes modifications que le paramètre formel lors de l'exécution de la procédure.

  - **Remarque :** le paramètre effectif doit être une variable (et non une valeur) lorsqu'il s'agit d'une transmission par adresse.

Les procédures

Transmission des paramètres: exemples

Procédure incrementer1 **(x** : entier **par valeur, y** : entier **par adresse)**
    x ← x+1;
    y ← y+1;
FinProcédure

Algorithme Test_incrementer1
variables n, m : entier
Début
  n ← 3;
  m ← 3;
  incrementer1(n, m) ;
  écrire (" n= ", n, " et m= ", m) ;
Fin

résultat : n=3 et m=4

Remarque : l'instruction x ← x+1; n'a pas de sens avec un passage par valeur

Les procédures

Transmission des paramètres: exemples

Procédure qui calcule la somme et le produit de deux entiers :

Procédure SommeProduit (x, y: entier **par valeur,** som, prod : entier **par adresse**)
    som ← x+y; prod ← x*y;
FinProcédure

Procédure qui échange le contenu de deux variables :

Procédure Echange (**x** : réel **par adresse, y** : réel **par adresse**)

Variables z : réel
    z ← x;
    x ← y;
    y ← z;
FinProcédure

SOUS ALGORITHMES

- **Passage par valeur**

Fonction échnger(x: réel, y : réel) : vide
z : réel;
début
z := x;
x := y;
y := z;
fin
Fonction appelante() a, b : réel
début
a := 2; b:= 7;
échanger(a,b);
écrire(a = , a);
écrire(b = , b);
fin

Les résultas affichés par la fonction appelante :
a = 2 **b = 7**

- **Passage par référence**

Fonction echnger(ref x: réel, ref y : réel) : vide
z : réel;
début
z := x;
x := y;
y := z;
fin
Fonction appelante() a, b : réel
début
a := 2; b:= 7;
échanger(a,b);
écrire(a = , a);
écrire(b = , b);
fin
Les résultas affichés par la fonction appelante :
a = 7 **b = 2**

Les procédures

Variables locales et globales

- On peut manipuler 2 types de variables dans un module (procédure ou fonction) : des **variables locales** et des **variables globales**. Elles se distinguent par ce qu'on appelle leur **portée** (leur "champ de définition", leur "durée de vie").

- Une **variable locale** n'est connue qu'à l'intérieur du module où elle a été définie. Elle est créée à l'appel du module et détruite à la fin de son exécution.

- Une **variable globale** est connue par l'ensemble des modules et le programme principale. Elle est définie durant toute l'application et peut être utilisée et modifiée par les différents modules du programme.

- La manière de distinguer la déclaration des variables locales et globales diffère selon le langage
  - En général, les variables déclarées à l'intérieur d'une fonction ou procédure sont considérées comme variables locales.

- En pseudo-code, on va adopter cette règle pour les variables locales et on déclarera les variables globales dans le programme principale.

Les procédures

Variables locales et globales

- Quelles sont les valeurs de a et b après l'exécution du programme suivant:

Fonction f (x : entier): entier
a ← 2;
b ← 7;
Retourne(a*x+b)
FinFonction

Algorithme Test_f
variables a, b : entier
Début
a ← 0;
b ← 2;
a++
b--
b ← f(a)

Fin