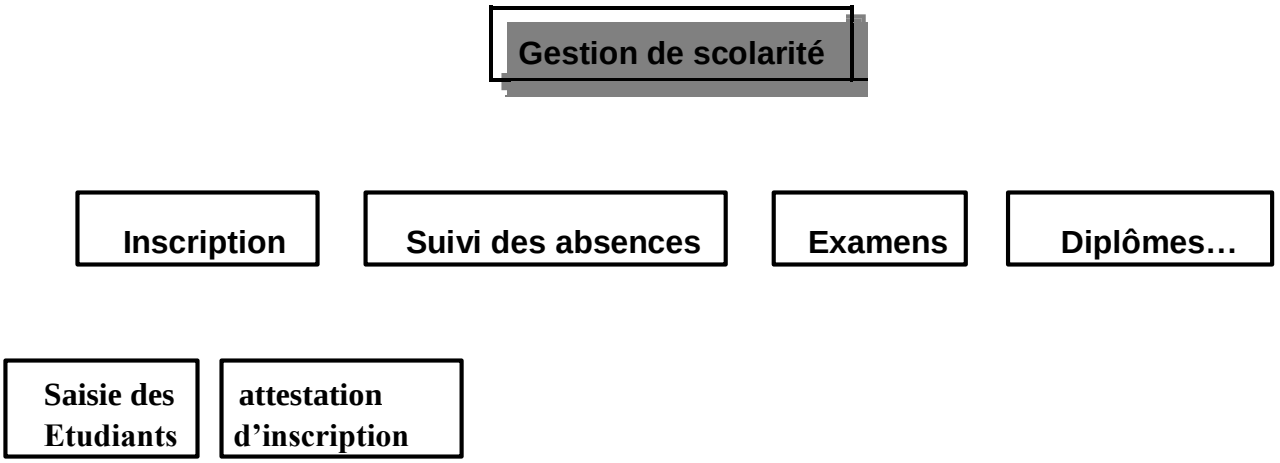
Les procédures et les fonctions

1-Introduction

Dans la résolution d'un problème, on peut constater qu'une suite d'actions revient plusieurs fois. Dans ce cas il serait judicieux de l'écrire une seule fois, et de l'utiliser autant de fois que c'est nécessaire, en effectuant des calculs avec des données différentes.

Cette suite d'actions sera définie dans un sous-programme, qui peut prendre soit la forme d'une procédure, soit la forme d'une fonction. D'autre part, on peut observer que certains groupes d'actions se rapportent à des traitements précis et différents. Il est souhaitable alors de représenter chacun d'eux dans un sous-programme. On perçoit alors un programme comme un ensemble de procédures/fonctions. La structuration d'un programme par morceaux (modules) est la base de la programmation structurée et modulaire.

A titre d'exemple, un programme de gestion de scolarité peut être découper en plusieurs modules : inscription, suivi des absences, examens, diplômes, etc.



2-Avantages d'utilisation des procédures/fonctions

- D'améliorer la lisibilité des programmes
- Optimiser le nombre d'instructions dans un programme
- De faciliter la mise au point et la correction des erreurs.

3-Les procédures

3-1- Définition

Une procédure est un ensemble d'instructions décrivant une action simple ou composée, à laquelle on donne un nom, qui devient lui-même en quelque sorte un sous-programme.

```
Procédure Nom_de_la_procédure (Liste d'arguments : type)
        Les déclarations
DÉBUT
        Instructions de la procédure
FIN
```

La première ligne s'appelle l'en-tête de la procédure. La liste d'arguments est une suite de données à échanger avec d'autres programmes.

3-2- Appel de la procédure

Nom_de_la_procédure (Liste d'arguments)

3-3- Procédure simple (sans paramètres)

Exemple

```
ALGORITHME PROC
CONST
        titre = "Algorithmique"

Procédure TRAIT( )
VAR
        i : entier
DÉBUT
        Pour i de 1 à 13 faire
                Écrire ( " - " )
        FinPour
FIN

DÉBUT(Programme principal)
        Ecrire (titre)
        TRAIT( )        /* Appel de la fonction TRAIT*/
FIN
```

Remarques

1-La structure d'une procédure est analogue à celle d'un algorithme. Elle possède une entête, une partie déclarative et un corps.

2- Pour appeler une procédure simple, il suffit d'écrire le nom de cette procédure.

3- Une variable locale (privée) est déclarée dans une procédure et ne peut être utilisée qu'à l'intérieur de celle-ci.

Exemple : **«** i » est une variable locale à la procédure « TRAIT ».

4- Une variable globale (publique) est déclarée au début de l'algorithme. Elle peut être utilisée dans le corps principal de l'algorithme ou par les différentes procédures.

Exemple : « titre » est une constante globale.

Il est fortement recommandée d'utiliser autant que possible des variables locales pour rendre les modules plus autonomes et par conséquent utilisables dans d'autres programmes.

3-4- Procédure paramétrée (avec paramètres)

Une procédure paramétrée est un module qui utilise des paramètres pour faire passer des informations entre la procédure appelée et le programme appelant.

3-4-1- Les différent types de paramètres

Les paramètres formels (fictifs): qui figurent dans l'entête de la déclaration de la procédure sont utilisés dans les instructions de la procédure et la seulement. Ils correspondent à des variables locales.

Procédure Nom_de_la_procédure (pf1 : type 1 ; pf2 : type 2 ; ……..)

Les paramètres effectifs (réels) : qui figurent dans l'instruction d'appel de la procédure et sont substitués aux paramètres formels au moment de l'appel de la procédure.

Nom_de_la_procédure (pe1 , pe2 , ……..)

3-4-2- Passage par valeur

Exemple

ALGORITHME PASSAGE_VALEUR
VAR
x : entier

```
Procédure ESSAI ( i : entier )
DÉBUT
i ← 3 * i
Écrire ( "Le paramètre valeur i =" , i )
FIN
```

DÉBUT (P.P)
Ecrire ("Donner la valeur de X : ")
Lire (x)
Ecrire ("Avant appel x = " , x)
ESSAI (x) */* Appel de la fonction ESSAI*/*
Ecrire ("Après appel x = " , x)
FIN

Interprétation :

Le passage de paramètres par valeur permet au programme appelant de transmettre une valeur à la procédure appelée. Le paramètre formel **« i »** est une variable simple. Le paramètre effectif correspondant **« x »** est une expression de même type que le paramètre formel.

Avant d'exécuter les instructions de la procédure la valeur du paramètre effectif est affectée au paramètre formel. Dans ce cas, le paramètre formel est considéré comme une variable locale qui est initialisée lors de l'appel.
Compléter le tableau suivant :

x = 10	P.P	Procédure
Appel de « ESSAI »	x = 10	i = 10
Exécution de « ESSAI »	x = 10	i = 30
Après « ESSAI »	x = 10	i = 30

- **Avant** appel x =10
- **Après** appel x =10

Remarque

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

Conclusion :

1- Le transfert d'information est effectué dans un seul sens, du programme principal vers la procédure.
Sens de transfert **:**

P.P. ⟶ **Procédure**

2- Toute modification du paramètre formel est sans conséquent sur le paramètre effectif.

3-4-3- Passage par variable

Exemple

ALGORITHME PASSAGE_VARIABLE

VAR

x : entier

Procédure ESSAI (VAR i : entier)
DÉBUT
i ← 3 * i
Écrire ("Le paramètre variable i =" , i)
FIN

DÉBUT(P.P)

Ecrire ("Donner la valeur de X : ")
Lire (x)
ESSAI (x) */* Appel de la fonction ESSAI*/*
Ecrire ("Après appel x = " , x)

FIN

Interprétation :

Le passage de paramètres par variable permet au programme appelant de transmettre une valeur à la procédure appelée et vice-versa. Le paramètre formel « i » est une variable. Le paramètre effectif correspondant « x » doit également être une variable du même type que le paramètre formel.

Pendant l'exécution des instructions de la procédure, le paramètre formel fait référence au même contenant variable que celui désigné par le paramètre effectif.
Compléter le tableau suivant :

x = 10	P.P	Procédure
Appel de « ESSAI »	x = 10	i = 10
Exécution de « ESSAI »	x = 30	i = 30
Après « ESSAI »	x = 30	i = 30

- **Avant** appel x =10
- **Après** appel x =30

Conclusion :

1-Le transfert d'information est effectué dans les deux sens, du programme principal vers la procédure et vice-versa.

Sens de transfert **:**

P.P. ⟷ **Procédure**

2- Toute modification du paramètre formel entraîne automatiquement la modification de la valeur du paramètre effectif.

Exercice N°1

Faire la trace d'exécution de l'algorithme suivant :

ALGORITHME PARAMÈTRE
VAR
i, j : entier

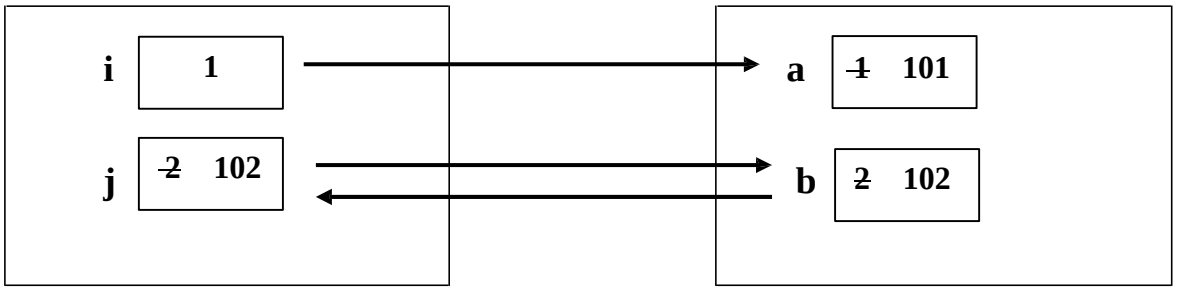
Procédure TRANSMISSION (a : entier ; VAR b : entier)
DÉBUT
a ← a + 100
b ← b + 100
Écrire (" a =" , a , " b =" , b)
FIN

DÉBUT(P.P)
i ← 1
j ← 2
TRANSMISSION (i , j)
Écrire (" i =" , i , " j =" , j)
FIN

i	1
j	2 102
a	1 101
b	2 102
Résultat	**a = 101 b = 102** **i = 1 j = 102**

Espace mémoire défini pour les variables du P.P.

Espace mémoire défini pour les variables de la procédure



Exercice N°2

ALGORITHME PERMUTATION
VAR
x, y : entier

Procédure ÉCHANGE (VAR a , b : entier)
VAR
vs : entier
DÉBUT
vs ← a
a ← b
b ← vs
Écrire (" Dans ECHANGE a =" , a , " b =" , b)
FIN

DÉBUT(P.P)

Écrire (" Donner x et y")
Lire (x,y)
Écrire (" Avant appel de ECHANGE x =" , x , " y =" , y)
ECHANGE (x , y)
Écrire (" Aprés appel de ECHANGE x =" , x , " y =" , y)

FIN

Exercice N°3

Ecrire une procédure PUISSANCE à 3 paramètres a, b et P qui calcule $P = a^b = a * a * a *……*a$ (b fois)

Ecrire le L'algorithme principal qui permet :

- De saisir deux entiers positifs (x, et y)
- D'afficher x^y

ALGORITHME CALCUL
VAR

x, y , R : entier

Procédure PUISSANCE (a , b : entier ; VAR P : entier)
VAR

i : entier

DÉBUT

P ← 1
Pour i de 1 à b faire
P ← P * a
Fin pour

FIN

DÉBUT(P.P)

Répéter
Ecrire (" Donner x et y")
Lire (x,y)
Jusqu'à (x>0 et y>0)
PUISSANCE(x, y, R)
Écrire (x , "^" , y , " =" , R)

FIN

4-Les fonctions

4-1- Définition

Une fonction est une procédure particulière qui admet un seul paramètre variable de type simple (entier, caractère, réel, etc.). Ainsi une fonction est tout simplement une procédure qui ne retourne qu'un seul résultat, une seule valeur, au programme appelant.

Il est obligatoire de préciser, dès le début le type de la fonction qui est en même temps le type du résultat à retourner.

4-2- Structure d'une fonction

Fonction Nom_de_la_fonction (Paramètres formels) : type
Les déclarations
DÉBUT
<Séquence d'instructions>
Nom_de_la_fonction ⟵ Résultat
Fin

4-3- Appel d'une fonction

Nom_de_la_fonction (<liste des paramètres effectifs>)

Exemple

Ecrire une fonction MIN qui retourne le minimum de 2 entiers a et b.

```
Fonction MIN (a , b : entier) : entier
VAR
    c : entier
DÉBUT
        Si (a<=b) alors
                c ← a
        si non
                c ← b
        Fin si
    MIN ← c
FIN
```

APPEL

1- z ← MIN(3,5)

2- Ecrire (" Donner x et y")
Lire (x,y)
z ← MIN(x,y)

3- t ← (7 + MIN(x,y)) / 2

4- Ecrire (" Le minimum = " , MIN(x,y))

5- si (MIN(x,y) > 0) alors …….

Exercice **:**

Comment utiliser la fonction MIN pour afficher le minimum de trois entiers x, y et z en une seule instruction et sans utiliser de variables supplémentaires.

Ecrire (" Le minimum = " , MIN (MIN(x,y) , z))

Remarques **:**

1-Le nom de la fonction joue un double rôle, c'est à la fois l'identifiant de la fonction et une variable locale.

2-Dans une fonction, le passage de tous les paramètres se fait par valeur.

Conclusion

Lors de l'utilisation d'une fonction, il faut :

1-Spécifier le type de la fonction,

2- Déclaré, si nécessaire, une variable locale de même type que la fonction (pour faire les calculs intermédiaires)

3-Affecter le résultat du calcul de la fonction au nom de la fonction, obligatoirement avant la fin du bloc (ranger le contenu de la variable locale dans la fonction).