Cours en algorithme

Le chemin vers la programmation

mai 26, 2021

absent : booléen

Les opérations sur des variables:

Type	Opérations	Symboles	Exemples
Entier	Addition	+	2 + 5 = 7
	Soustraction	-	8 – 5 = 3
	Multiplication	*	2 * 4 = 8
	Division	/	10 / 4 = 2,5
	Division entier	Div	10 Div 4 = 2
	Modulo (le reste de la division entier)	Mod	10 Mod 3 = 1
	Comparaison	≤ ≥ > < = ≠	7 > 2 vrai
Réel	Addition	+	2 + 5,4 = 7,4
	Soustraction	-	8,5 – 5 = 3,5
	Multiplication	*	2 * 4 = 8
	Division	/	10,6 / 4 = 2,65
	Comparaison	≤ ≥ > < = ≠	7 < 2 faux
Caractère	Comparaison	≤ ≥ > < = ≠	'B' < 'K ' faux
Chaine	Concaténation	& +	"ok " & " " & "by"
	Comparaison	≤ ≥ > < = ≠	'moh' < 'an ' vrai
Booléen	Logiques	et non ou	non(7>1) faux

3-3) Les constantes

Comme une variable, il existe une constante correspond un emplacement mémoire réservé auquel on accède par le nom qui lui a été attribué, mais dont la valeur stockée ne sera jamais modifiée au cours du programme.

Syntaxe :

4-2) L'instruction d'entrée :

L'instruction d'entrée ou de lecture donne la main à l'utilisateur pour saisir une donnée au clavier. La valeur saisie sera affectée à une variable

Syntaxe : Lire (identificateur)

Exemples : Lire (A) ou Lire (A, B, C)

L'instruction Lire(A) permet à l'utilisateur de saisir une valeur à clavier. Cette valeur sera affectée à la variable A.

4-3) L'instruction de sortie :

Avant de lire une variable, il est conseillé d'écrire un message à l'écran, afin de prévenir l'utilisateur de ce qu'il doit taper.
L'instruction de sortie (d'écriture) permet d'afficher des informations à l'écran.

Syntaxe : Ecrire (expression)

Exemple : Ecrire ("Donner votre âge : ")

Ecrire (A) cette instruction permet d'afficher à l'écran la valeur de variable A.

4-3) Les commentaires :

Lorsqu'un algorithme devint long, il est conseillé d'ajouter des lignes de commentaires dans l'algorithme, c'est-à-dire des lignes qui ont pour but de donner des indications sur les instructions

effectuées et d'expliquer le fonctionnement d'algorithme (programme) sans que le compilateur ne les prenne en compte.

On va voir deux types de commentaires

// Commentaire sur une ligne

/* Commentaire sur plusieurs lignes   */

Remarque :

Parfois on utilise les commentaires pour annuler l'action de quelques instructions dans un algorithme ou un programme au lieu de les effacer comme dans cet exemple :

Variable i: entier

// Variable j: réel

Chapitre 2 : Les structures alternatives et répétitives

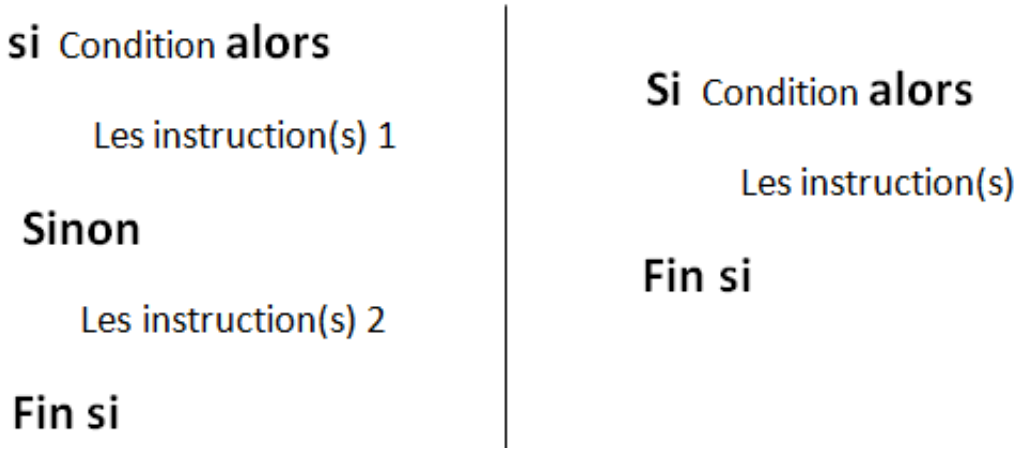
1) Les structures alternatives

1-1) Introduction :

Contrairement au traitement séquentiel, la structure alternative ou conditionnelle permet d'exécuter ou non une série

d'instructions selon la valeur d'une condition.

1-2) La structure Si alors ..... sinon ..... fin si ou Si alors ..... fin si



Une condition est une expression logique ou une variable logique évaluée à Vrai ou faux. La condition est évaluée. Si elle est vraie, la série d'instruction(s)1 est exécutée et
l'ensemble d'instruction(s) 2 est ignoré, la machine sautera directement à la première instruction située après **Fin si**.

De même, au cas où la condition était fausse la machine saute directement à la première ligne située après le **Sinon** et exécute l'ensemble d'instruction2.

Exercice d'application 1

Écrire un algorithme qui affiche si un nombre entier saisi au

clavier est pair ou impair

```
Algorithme  Parité
Variables n :entier
Debut
  Ecrire("Entrer un entier:")
  Lire(n)
  Si(n mod 2 = 0) alors
      Ecrire(n,"est pair")
     SiNon
      Ecrire(n,"est impair")
  FinSi
Fin

Résultat ==> Entrer un entier : 9
             9 est impair
```

Remarque : il existe aussi un autre type de condition c'est la condition composées.

Certains problèmes exigent de formuler des conditions qui ne peuvent être exprimées sous la forme simple, par exemple la condition de note de devoir doit être inclus dans l'intervalle [0, 20], cette condition est composée de deux conditions simples qui sont note ≥ 0 et note ≤ 20

Exercice d'application 2

Ecrire un algorithme qui teste une note saisi au clavier est comprise entre 0 et 20.

```
Algorithme
Variables note :réel
Debut
    Ecrire(" Entrer la note : ")
    Lire (note)

    Si(note < 0  ou  note > 20) alors
      Ecrire("Erreur !!" )
    FinSi
    Si( note >= 0  et note < 10) alors
      Ecrire(" non Validé" )
    FinSi
    Si( note >= 10  et note <= 20) alors
      Ecrire("Validé" )
Fin

Résultat ==> Entrer la note : 12.5
             Validé
```

Exercice 1:

Ecrire un algorithme qui demande deux nombres m et n à l'utilisateur et l'informe ensuite si le produit de ces deux nombres est positif ou négatif. On inclut dans l'algorithme le cas où le produit peut être nul.

```
Algorithme
Variables m,n:réels
Debut
   Ecrire(" Entrer un nombre:") Lire(m)
   Ecrire(" Entrer un nombre:") Lire(n)
  Si( n == 0  ou  m == 0) alors
     Ecrire("Le produit est nul!!")
  FinSi

  Si( m*n < 0) alors
     Ecrire("Le produit est négatif")
  FinSi

  Si( m*n > 0) alors
     Ecrire("Le produit  est positif")
  FinSi
Fin

Résultat ==> Entrer un nombre:  -7.4
             Entrer un nombre:  2
             Le produit est négatif
```

Exercice 2:

Une boutique propose à ces clients, une réduction de 15% pour les montants d'achat supérieurs à 200 dh. Ecrire un algorithme permettant de saisir le prix total HT et de calculer le montant TTC en prenant en compte la réduction et la TVA=20% .

```
Algorithme Calcul_TTC
Variables Prix_HT , Prix_TTC: reél
Constante TVA = 0.2
Debut
Ecrire(" Entrer le montant HT:")
Lire (Prix_HT)
Prix_TTC ← Prix_HT + Prix_HT*0.2
  Si ( Prix_HT > 200 ) alors
    Prix_TTC ← Prix_TTC - Prix_TTC*0.15
     Ecrire("le montant TTC est:", Prix_TTC)
  Sinon
    Ecrire("le montant TTC est:", Prix_TTC)
  Fin si
fin
```

1-3) Structure à choix multiples

Cette structure conditionnelle permet de choisir le traitement à effectuer en fonction de la valeur ou de l'intervalle de valeurs d'une variable o d'une expression.

Syntaxe :

```
Selon sélecteur faire
        Valeur 1 : action(s)1
        Valeur 2 : action(s)2
        Valeur 3 : action(s)3
        ……..
        Valeur n : action(s) n
        Sinon
                Action (s)
FinSelon
```

Lorsque l'ordinateur rencontre cette instruction, il vérifie la valeur de la variable de sélection (sélecteur) et il la compare aux différentes valeurs.

Les valeurs sont évaluées dans l'ordre, les unes après les autres, et une fois la valeur de sélecteur est vérifiée l'action associée est exécutée. On peut utiliser une instruction Sinon (facultative), dont l'action sera exécutée si aucune des valeurs évaluées n'a pas été remplie

Exercice :

Ecrire un algorithme permettant d'afficher le mois en lettre selon le numéro saisi au clavier.

```
Algorithme Mois
Variables N: entier
Debut
Ecrire(" Entrer le numéro du mois:")
Lire (N)
selon N faire
    1: Ecrire (" janvier ")
    2: Ecrire (" février ")
    3: Ecrire (" mars ")
    .........
    12: Ecrire ("décembre ")
 sinon
      .......
FinSelon
fin
```

2) Les Structures répétitives

2-1) Introduction

Prenons d'une saisie au clavier, par exemple, on pose une question à laquelle doit répondre par « oui » ou « non ».
L'utilisateur risque de taper autre chose (une autre lettre), le programme peut soit planter par une erreur d'exécution soit dérouler normalement jusqu'au bout, mais en produisant des résultats fantaisistes.
Pour éviter ce problème, on peut mettre en place un contrôle de saisie pour vérifier que les données entrées au clavier correspondent bien à celles attendues par l'algorithme.

```
Algorithme Controle_de_saisie
Variable Reponse: caractère
Debut
   Ecrire("Voulez vous la suite de
           ce cours? (O/N)")
   Lire (Reponse)
 Si ( Reponse ≠ 'O' et Reponse ≠ 'N'  ) alors
     Ecrire("Erreur de saisie. Rocommencez")
     Lire (Reponse)

 Fin si
fin
```

!!! L'algorithme ci-dessus résout le problème si on se trompe qu'une seule fois, et on fait entrer une valeur correcte à la deuxième demande. Sinon en cas de deuxième erreur, il faudrait rajouter un « SI ». Et ainsi de suite, on peut rajouter des centaines de « SI »
==> La solution à ce problème consiste à utiliser une structure répétitive.

Une structure répétitive, encore appelée **boucle**, est utilisée quand une instruction ou une liste d'instruction, doit être répétée plusieurs fois. La répétition est soumise à une condition

2-2) La boucle Tant Que .... Faire

La boucle Tant que permet de répéter un traitement tant que la condition est vraie.

Syntaxe :

Tant que Condition **faire**

Instruction(s)

Fin TantQue

==> L'exécution de la boule dépend de la valeur de la condition. Si est vrai, l'algorithme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne FinTantque .Il retourne ensuite sur la ligne du Tantque, procède au même examen, et ainsi de suite.
==> La boucle ne s'arrête que lorsque prend la valeur fausse, et dans ce cas le programme poursuit son exécution après FinTantQue.

Exemple:

```
Algorithme Controle_de_saisie
Variable Reponse: caractère
Debut
   Ecrire("Voulez vous la suite de
          ce cours? (O/N)")
   Lire (Reponse)
 TantQue ( Reponse ≠ 'O'  et  Reponse ≠ 'N'  ) alors
     Ecrire("Erreur de saisie. Rocommencez")
     Lire (Reponse)
 FinTantQue
fin
```

Remarque :

Si la structure TantQue contient la condition ne devient jamais fausse. Le programme tourne dans une boucle infinie et n'en sort plus.

Exemple :

```
Algorithme Boucle_infinie
Variable i: entier
Debut
  i ← 1
   TantQue ( i <= 10  ) faire
     Ecrire("Bonsoir")
   FinTantQue
fin
```

Dans cet exemple nous avons une boucle infinie. L'ordinateur ne s'arrêtera jamais d'afficher le message ***Bonsoir*** car la variable i qui est testée dans la condition n'est jamais incrémenter. Alors pour résoudre le problème de boucle infinie dans ce cas-là, on incrémente la variable i à chaque tour de boucle pour afficher ***Bonsoir*** 10 fois exactement.

```
Algorithme Bonsoir_10_fois
Variable i: entier
Debut
  i ← 1
   TantQue ( i <= 10  ) faire
     Ecrire("Bonsoir")
      i ← i+1  // i++
   FinTantQue
fin
```

```
Résultat : ==>
       Bonsoir
       Bonsoir
       Bonsoir
       Bonsoir
       Bonsoir
       Bonsoir
       Bonsoir
       Bonsoir
       Bonsoir
       Bonsoir
```

Exercice1

Ecrire un algorithme qui calcule la somme S = 1+2+3+4+.....+ 10

```
Algorithme Somme_de_1_jusqu'au_10
Variables i,S: entier
Debut
  i  ← 1
  S  ← 0
   TantQue ( i <= 10  ) faire
     S  ← S + i
     i  ← i+1
   FinTantQue
   Ecrire("La somme de 1 à 10 est:",S)
fin

Résultat : ==>
      La somme de 1 à 10 est: 55
```

Exercice2

Ecrire un algorithme qui calcule la somme S= 1+2+3+4+.....+ N, où N saisi par l'utilisateur.

```
Algorithme Somme_de_1_jusqu'au_N
Variables i,S,N: entier
Debut
  i  ← 1
  S  ← 0
  Ecrire("Donner un entier:")
   Lire (N)
   TantQue ( i <= N  ) faire
      S ← S + i
      i ← i+1
   FinTantQue
   Ecrire("La somme de 1 à N est:",S)
fin
```

```
Résultat : ==> Donner un entier : 6
               La somme de 1 à 6 est: 21
```

2-3) La boucle Pour .... Faire

La boucle *pour .... Faire* permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe :

```
Pour compteur ← valeur_initiale Jusqu'à valeur_finale faire
   Instruction(s)
Fin pour
```

==> La variable compteur est de type entier. Elle est initialisée par la valeur initiale, le compteur augmente sa valeur de 1 automatiquement à chaque tour de boucle jusqu'à la valeur finale.

==> Lorsque la variable compteur vaut la valeur finale, le traitement est exécuté une seule fois puis le programme sort de la boucle.

Exemple:

Ecrire un algorithme qui affiche **Bonjour** 10 fois.

```
Algorithme Bonjour_10_ fois
Variable i: entier
Debut
  pour i  ← 1 jusqu'à  10 faire
  Ecrire("Bonjour")
  FinPour
fin
```

```
 Résultat : ==>
            Bonjour
            Bonjour
            Bonjour
            Bonjour
            Bonjour
            Bonjour
            Bonjour
            Bonjour
            Bonjour
            Bonjour
```

Exercice1

Ecrire un algorithme qui calcule S= 1+2+3+4+.....+ 10. Utilisant la boucle pour.

```
Algorithme Somme_de_1_jusqu'au_10
Variables i,S: entier
Debut
  S ← 0
   pour i ← 1 jusqu'à  10 faire
     S ← S  + i
   FinPour
   Ecrire("La somme de 1 à 10 est:" S)
fin

Résultat : ==>
          La somme de 1 à 10 est: 55
```

Exercice2

Ecrire un algorithme qui calcule S= 1+2+3+4+.....+ N. Utilisant la boucle pour.

```
Algorithme Somme_de_1_jusqu'au_N
Variables i,S,N: entiers
Debut
  S ← 0
   Ecrire("Donner un entier")
   Lire (N)
   pour i ← 1 jusqu'à  N faire
     S ← S  + i
   FinPour
   Ecrire("La somme est:" S)
fin

Résultat : ==>
        Donner un entier: 17
        La somme de 1 à 17 est: 153
```

Exercice 3

Ecrire un algorithme qui affiche la table de multiplication de 5. Utilisant la boucle pour.

```
Algorithme Table_Multiplication_de_5
Variables i:entier
Debut
   pour i  ← 0 jusqu'à  10 faire
      Ecrire("5*",i,"=",i*5)
   FinPour
fin

Résultat :
  5*0=0
  5*1=5
  5*2=10
  .....
  5*10=50
```

Exercice 4

Ecrire un algorithme qui affiche la table de multiplication d'un entier saisie par l'utilisateur, Utilisant la boucle pour.

```
Algorithme Table_Multiplication_de_N
Variables i,N :entiers
Debut
      Ecrire("Donner un entier")
      Lire (N)
   pour i ← 0 jusqu'à N faire
      Ecrire("N*",i,"=",i*N)
   FinPour
fin

Résultat :
  N*0=..
  N*1=..
  N*2=..
  .....
  N*10=..
```

2-4) La boucle Répéter...jusqu'à

Cette boucle permet de répéter une instruction jusqu'à ce qu'une soit vrai.

Remarque : Cette boucle ne s'utilise en général que pour des menus, elle est dangereuse car il n'y a pas de vérification de la condition avant d'y entrer.

Syntaxe :

==> La liste d'instructions est exécutée, puis la condition est évaluée. Si elle est fausse, le corps de la boucle est exécuté à nouveau puis la condition est réévaluée et si elle est vraie, le programme sort de la boucle et exécute l'instruction qui suit **Jusqu'à**.

Exemple:

En utilisant la boucle Répéter.....jusqu'à, on écrit un algorithme qui affiche ***Bonjour*** 10 fois.

```
Algorithme Bonsoir_10_fois
Variables i: entier
Debut
      i ← 1
   Répéter
      Ecrire("Bonsoir")
       i ← i+1
   Jusqu'à  i>10
Fin

Résultat : ==>  Bonjour
                Bonjour
                Bonjour
                Bonjour
                Bonjour
                Bonjour
                Bonjour
                Bonjour
                Bonjour
                Bonjour
```

Exercice1

Ecrire un algorithme qui calcule la somme S= 1+2+3+...+ 10.
Utilisant la boucle Répéter jusqu'à.

```
Algorithme Somme_de_1_jusqu'au_10
Variables i,S: entiers
Debut
  S <== 0
  i<== 1

  Répéter
      S <== S  + i
      i<== i+1
   jusqu'à  i>10
   Ecrire("La somme de 1 0 10 est:" S)
fin


 Résultat ==>
            La somme de 1 0 10 est: 55
```

Exercice 2

Ecrire un algorithme qui affiche la table de multiplication de 8. Utilisant la boucle Répéter jusqu'à.

hhhhh

```
Algorithme Table_Multiplication_de_8
Variables i:entier
Debut
    i ← 0
   Répéter
      Ecrire("8*",i,"=",i*8)
      i ← i+1
   jusqu'à  i>10
fin

Résultat :
  8*0=0
  8*1=8
  8*2=16
  .....
  8*10=80
```

Remarques :

==> Les boucles « Répéter » et « «TantQue » sont utilisées lorsqu'on ne sait pas au départ combien de fois il faudra exécuter ces boucles.

==> A la différence de la boucle « TantQue » par rapport à la boucle « Répéter » cette
dernière est exécutée au moins une seule fois.

==> La condition d'arrêt de la boucle « répéter » est la négation de la condition de poursuite de la boucle «TantQue »

==> On utilise la boucle « Pour » quand l'on connait le nombre d'itérations à l'avance.

** Chapitre 3: Les Tableaux **

1) Introduction

Imaginons que dans un algorithme, nous avons besoin d'un grand nombre de variables, il devient difficile de donner un nom pour chaque variable.

Exemple :

Ecrire un algorithme permettant de saisir cinq notes et de les afficher après avoir multiplié toutes les notes par trois.

```
Algorithme Notes
Variables N1,N2,N3,N4,N5 : reéls
Debut
Ecrire("Donner  la note 1:")
Lire (N1)
Ecrire("Donner  la note 2:")
Lire (N2)
Ecrire("Donner  la note 3:")
Lire (N3)
Ecrire("Donner  la note 4:")
Lire (N4)
Ecrire("Donner  la note 5:")
Lire (N5)

Ecrire("la  1er note est :",N1*3)
Ecrire("la  2eme note est :",N2*3)
Ecrire("la  3eme note est :",N3*3)
Ecrire("la  4eme note est :",N4*3)
Ecrire("la  5eme note est :",N5*3)
fin
```

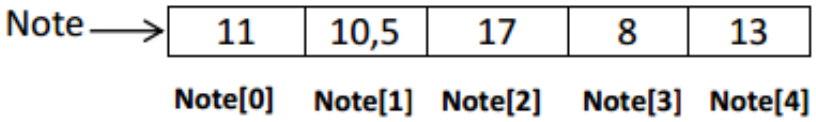
==> La même instruction répète cinq fois. Imaginons que si l'on voudrait réaliser cet algorithme avec 100 notes, cela devient très difficile.

==> Pour résoudre ce problème, il existe un type de données qui

permet de définir plusieurs. variables de même type.

2) Définition

Un tableau est une suite d'éléments de même type. Il utilise plusieurs cases mémoire à l'aide d'un seul nom. Comme toutes les cases portent le même nom, elles se différencient par un numéro ou un indice.
Nous pouvons représenter schématiquement un tableau nommé Note composé de cinq cases, dans la mémoire comme suit :



3) Tableau à une dimension

3-1) Déclaration

La déclaration d'un tableau permet d'associer à n nom d'une zone mémoire composée d'un certain nombre de cases mémoires de même type.
Syntaxe : ***Variable identificateur : tableau [taille_max] de type .....***

Exemple : Variable Note : Tableau[40] de réels
Remarques :
==> Le premier élément d'un tableau porte l'indice 1 ou 0 selon les langages.
==> La valeur d'un indice doit être un nombre entier.
==> La valeur d'un indice doit être inférieure ou égale au nombre

d'éléments du tableau. Par exemple, avec le tableau tab [20], il est impossible d'écrire tab[21] ou tab[26], ces expressions font référence à des éléments qui n'existe pas.

==> L'utilisation de ces éléments se fait en suite, via le nom du tableau et son indice. Ce dernier peut être soit une valeur (tab[3]) , soit une variable (tab [i]) ou encore une expression (tab[i+1]).

==> Pour Faire un parcours complet sur un tableau, on utilise une boucle.

Exemple 1

Ecrire un algorithme permettant de saisir 20 notes et de les stocker dans un tableau nommé Etudiant, puis les afficher.

```
Algorithme Notes
Variable Etudiant:Tableau[20]de reéls
         i : entier
Debut
      pour i ← 1 jusqu'à  20 faire
      Ecrire (" donner la note d'étudiant:",i)
      lire(Etudiant[i])
      FinPour

  pour i ← 1 jusqu'à  20 faire
  Ecrire ("la note d'étudiant",i,"est:",Etudiant[i])
  FinPour
fin
```

Exemple 2 :

Ecrire un algorithme permettant de saisir 20 notes et de les afficher après avoir multiplié toutes ces notes par un coefficient fourni par l'utilisateur.

```
Algorithme Notes
Variable Notes : Tableau[20]de reéls
         i,Coef : entier
Debut
        Ecrire (" donner le coefficient:)
        lire(Coef)
       pour i ← 1 jusqu'à  20 faire
       Ecrire (" donner la note d'étudiant:",i)
       Lire (Notes[i])
       FinPour

  pour i ← 1 jusqu'à  20 faire
  Ecrire (Etudiant[i]*Coef)
  FinPour
fin
```

Exercice 1

Ecrire un algorithme permettant de saisir 20 notes et qui affiche la moyenne de ces notes.

```
Algorithme Moyenne_Notes
Variables Notes : Tableau[20]de reéls
            S   :réel
            i   :entier
Debut
          S ← 0
        pour i ← 1 jusqu'à  20 faire
        Ecrire (" donner la note:",i)
        Lire (Notes[i])
        S ← S+Notes[i]
        FinPour
       Ecrire ("La moyenne est:",S/20)
fin
```

Exercice 2

Ecrire un algorithme permettant de saisir 20 notes et qui affiche le maximum de ces notes.

```
Algorithme Maximum_Notes
Variables Notes : Tableau[20]de reéls
          Max   :réel
          i   :entier
Debut
        Ecrire (" donner la note 1 ")
        Lire (Notes[1])
        Max ← Notes[1]
      pour i ← 2 jusqu'à  20 faire
      Ecrire (" donner la note",i)
      Lire (Notes[i])
      Si( Notes[i] > Max )
        Max ← Notes[i]
      FinSi
      FinPour
     Ecrire ("Le Maximum est:",Max)
fin
```

4) Tableau à deux dimensions

Reprenons l'exemple des notes en considérant cette fois qu'un étudiant a plusieurs notes (une note pour chaque matière). On peut simplifier des choses comme suite.

	Mohamed	*Meryem*	*Anas*	*Rachida*
Informatique	*12*	*17*	*20*	*18*
Mathématiques	*12.5*	*10*	*18*	*14*
Physique	*15*	*16.5*	*13*	*15.5*

==> Les tableaux à deux dimensions se représentent comme une matrice ayant un certain nombre de lignes (première dimension) et un certain nombre de colonne (seconde dimension).

4-1) Déclaration

Syntaxe : ***Variable identificateur : tableau [***nb_lignes , nb_colonnes***] de <type>***

Exemple : Variable Note : Tableau[3,4] de réels

Remarques:

==> L'utilisation d'une matrice se fait via son nom et ses indices. Ces derniers peuvent être soient des valeurs (tab[1,3]) , soient des variables (tab [i,j]) ou encore des expressions (tab[i+1,j]).

==> Pour Faire un parcours complet sur une matrice, on utilise deux boucles, l'une au sein de l'autre, c'est ce qu'on appelle les boucles imbriquées. La première boucle pour parcourir les lignes tandis que la deuxième est utilisée pour parcourir les éléments de la ligne précisée par la boucle principale (la première boucle).

Exemple 1

Ecrire un algorithme permettant de saisir les notes d'une classe de 30 étudiants en 5 matières.

```
Algorithme Matrice_Notes
Variables Notes : Tableau[30,5]de reéls
          i ,j   :entier
Debut
    pour i  ←  1 jusqu'à  30 faire
       pour j  ←  1 jusqu'à  5 faire
        Ecrire ("étudiant",i,"matière",j)
        Lire (Notes[i,j])
       FinPour
    FinPour
    pour i  ←  1 jusqu'à  30 faire
     pour j  ←  1 jusqu'à  5 faire
     Ecrire ("étudiant",i,"matière",j,"=",Notes[i,j])
     Lire (Notes[i,j])
     FinPour
    FinPour
fin
```

*** Chapitre 4 : Procédures et les 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 fois.

Lorsque un algorithme dépasse deux page ou plus devient difficile à comprendre et à gérer par les programmeurs. 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-algorithme : les procédures et les fonctions

2) Les procédures

Une procédure est une série d'instruction regroupées sous un nom, qui permet d'effectuer des actions par un simple appel de la procédure dans un algorithme ou dans un autres sous algorithme.

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

2-1) Déclaration d'une procédure

Syntaxe :

```
Procédure  Nom (liste de paramètres)
Variables  identificateurs : type

Début

Instruction(s)

Fin
```

Après le nom de la procédure, il faut donner la liste des paramètres (s'il en a) avec leur type respectif. Ces **paramètres formels.** Leur valeur n'est pas connue lors de la création de la procédure.

Exemple 1 :

Ecrire une procédure qui affiche à l'écran une ligne de 15 étoiles

```
procédure étoiles()
Début
variable i : entier
pour i  ←  1 jusqu'à 15 faire
ecrire ("*")
finPour
Fin
```

Exemple 2 :

Ecrire une procédure qui affiche à l'écran une ligne de N étoiles, où N passé comme paramètre.

```
procédure étoiles(N: entier)
Début
variable i : entier
pour i ← 1 jusqu'à N faire
ecrire ("*")
finPour
Fin
```

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_procédure (.....)

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

Exemple 2 :

Ecrire un algorithme qui permet d'affiche à l'écran une ligne de N étoiles, où N passé comme paramètre dans une procédure.

```
Algorithme

  Procédure étoiles (n:entier)
   variable i : entier
      début
      pour i ← 1 jusqu'à n faire
          ecrire("*")
      finPour
   finProcédure

Debut
 étoiles (15) // affiche 7 étoiles
fin
```

2-3) Passage de paramètres

Les échanges d'informations entre une procédure 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.

==> ***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 effectif ne sera jamais modifiée.

Exemple:

Soit l'algorithme suivant :

```
Algorithme Passage_par_valeurs
variable N: entier
  Procédure Proc(A:entier)
     début
          A ← A*2
          ecrire(A)
  finProcédure

Debut                                   Résultat
   N ← 5
   Proc(N)                              10
   ecrire(N)                            5
fin
```

Cet algorithme définit une procédure pour laquelle on utilise le passage de paramètres par valeurs. Lors de l'appel de la procédure, la valeur du paramètre effectif N est recopiée dans le paramètre 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.

==> ***Passage par adresse (ou par référence)***

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 par le mot clé **Var.**

Exemple:

Soit l'algorithme suivant :

```
Algorithme Passage_par_adresse
variable N: entier
  Procédure Proc(Var A:entier)
     début
          A ← A*2
          ecrire(A)                    Résultat
  finProcédure
                                       10
Debut                                  10
   N ← 5
   Proc(N)
   ecrire(N)
fin
```

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

Dans cet algorithme le paramètre passé correspond à la référence (l'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 algorithme admettant des paramètres (ou sans paramètres) et retournant 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 (liste de paramètres) : type
Variables identificateurs : 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.

Remarque :

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

Exemple:

Définir une fonction qui renvoie le plus grand de deux nombres passées par les paramètres.

```
Fonction Maximum(A:réel,B:réel):réel
    début
        si A<B  alors
            retourner B
          sinon
            retourner A
        FinSi
finFonction
```

3-2) L'appel d'une fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivi 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_fonction (.....)

Exemple :

Ecrire un algorithme qui permet d'appeler à la fonction Max de l'exemple précédent :

```
Algorithme Maximum
variable A,B,M: entier
Fonction Max(m:réel,n:réel):réel
    début
        si n<m  alors
            retourner m
          sinon
            retourner n
        FinSi
finFonction
Debut
    ecrire ("Entrer deux nombres:")
    lire (A,B)
    M ← Max(A,B)
    ecrire("le maximum est:",M)
fin
```

Exercice1

Ecrire une fonction puissance qui permet de calculer la puissance d'un nombre réel.

```
Fonction puissance(a:réel,n:entier):réel
   variable i : entier
            p: reél
            p ← 1
     début
     pour i ← 1 jusqu'à n faire
          p ← p*a
     finPour
     retourner p
   finFonction
```

Exercice2

Ecrire un algorithme qui demande à l'utilisateur deux entiers, et calcule la somme des nombres entiers compris entre ces deux entiers. Par exemple : si l'utilisateur tape 5 et 17, c'est-à-dire la somme est : S= 5+6+7+…+ 17.

```
Algorithme Somme
Variables M,N,S: entiers
Fonction som(a:entier,b:entier):réel
   variable i : entier
            c: entier
            c ← 0
      début
      pour i ← a jusqu'à  b faire
          c ← c+i
      finPour
      retourner c
   finFonction
Debut
    ecrire("Entrer deux entiers :")
    Lire (M,N)
    S ← som(M,N)
ecrire("la somme des entiers entre",M,"et",N,"est",S)
fin
```

```
Résultat
7       13

la somme des entiers entre 7 et 13 est 70
```