

**Université M'Hamed Bougara – Boumerdès (UMBB)**

**Faculté des sciences - Département informatique**

## **Introduction à l'algorithmique**

# **Les principes de l'Algorithmique et de la Programmation**

Décembre 2015

## Table des matières

1. Définition de l'informatique .....	3
2. Définition d'un ordinateur.....	3
3. Méthode de résolution d'un problème .....	4
4. Définition d'un algorithme .....	5
4.1. Problème.....	5
4.2. Analyse .....	5
4.3. Algorithme.....	5
4.3.1. Déclaration.....	6
4.3.2. Traitement.....	7
5. Instructions de base .....	8
5.1. Instruction d'affectation .....	8
5.2. Instruction d'affichage .....	9
5.3. Instruction de lecture .....	9
5.4. Instruction de condition simple .....	11
5.5. Instruction de décision alternée .....	14
5.6. Instruction de décision multiple.....	15
6. Structures répétitives.....	16
6.1. Instruction de boucle « Pour » .....	16
6.2. Instruction de boucle « TantQue ».....	19
6.3. Instruction de boucle « Répéter » .....	20
6.4. Différences entre les boucles « Pour », « TantQue » et « Répéter ».....	22
7. Structures de données .....	22
7.1. Les structures de données séquentielles. ....	23
7.1.1. Les tableaux à une dimension (vecteur).....	23
7.1.1.1. Comment remplir un vecteur .....	25
7.1.1.2. Manipulation de plusieurs vecteurs .....	28
7.1.1.3. Recherche du maximum.....	29
7.1.1.4. Rotation d'un vecteur.....	31
7.1.1.5. Tri d'un vecteur.....	32
7.1.2. Les tableaux à deux dimensions (matrice).....	34
7.1.2.1. Comment remplir une matrice .....	35
7.1.2.2. Trace d'une matrice .....	36
7.1.2.3. Matrice identité .....	38
7.1.2.4. Transposé d'une matrice .....	39

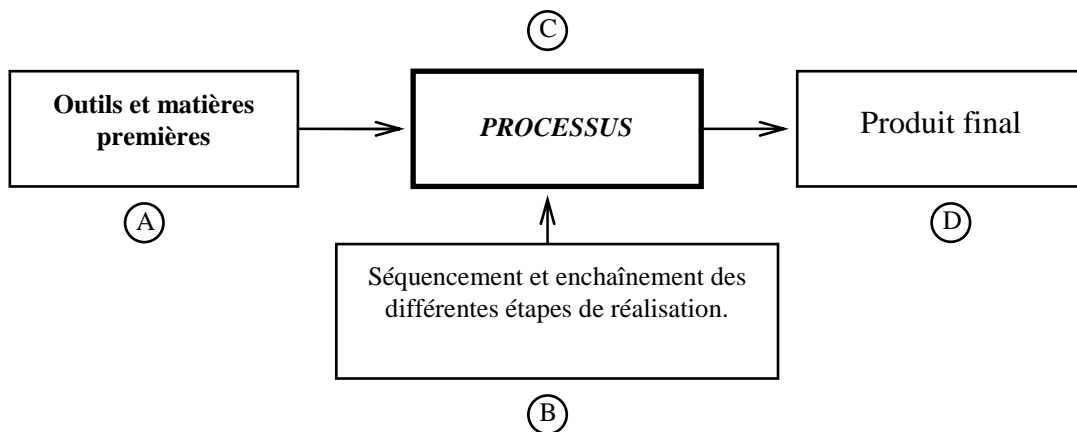
## 1. Définition de l'informatique

L'informatique (INFORmation autoMATIQUE), comme son nom l'indique, permet d'automatiser l'information que nous manipulons. Cette informatisation permettra de réaliser un gain considérable en temps et en effort.

Pour faire, il faut avoir une connaissance de la méthodologie à observer pour bien mener l'automatisation de l'information. Dans ce document, nous allons essayer de montrer, pédagogiquement, les différentes étapes à suivre pour résoudre un problème informatique.

## 2. Définition d'un ordinateur

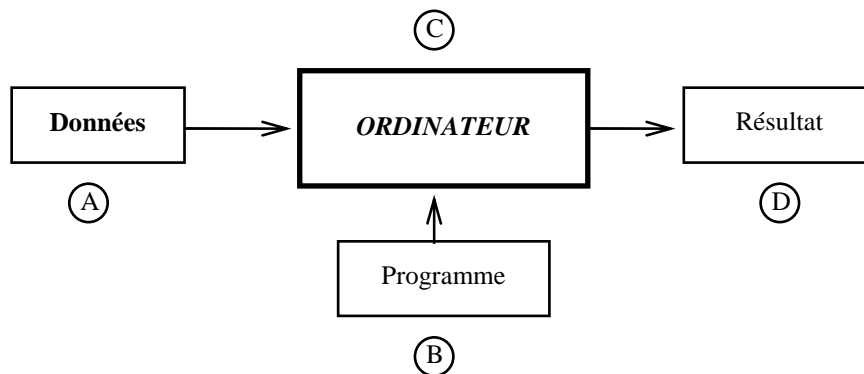
Pour mieux expliquer l'utilité d'un ordinateur, prenant comme exemple le processus de réalisation d'une pièce mécanique (le schéma ci-dessous):



Pour réaliser cette pièce, il faut:

1. Rassembler tous les outils, les machines et la matière première nécessaire.
2. Disposer les outils et les machines de telle manière qu'ils puissent décrire la succession des opérations à réaliser.
3. Le processus est exécuté progressivement et manuellement.
4. Le produit final constitue le résultat du traitement (la pièce)

Par analogie, si on veut automatiser le processus au moyen d'un ordinateur, nous aurons le schéma suivant:



Ce schéma montre que le processus ou traitement sera pris en charge par l'ordinateur pour automatiser le fonctionnement. Un traitement informatique nécessite en général des informations en entrées (données) et livre une sortie (résultat).

L'ordinateur est une super "machine à calculer". Il obéit au doigt et à l'oeil aux ordres que l'homme lui soumet. Supposant que l'on souhaite qu'il calcule les racines d'une équation du second degré en lui soumettant les ordres suivants:

1. Lire les 3 coefficients A, B et C.
2. Calculer la valeur de  $\Delta$  telle que  $\Delta = B^2 - 4AC$
3. En supposant que  $\Delta$  est positive, calculer les racines  $X_1$  et  $X_2$  tel que:

$$X_1 = \frac{-B - \sqrt{\Delta}}{2A} \quad X_2 = \frac{-B + \sqrt{\Delta}}{2A}$$

Dans ce cas, posons-nous les questions suivantes:

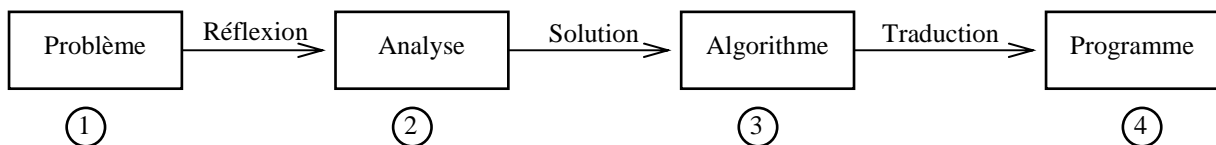
- Est ce que l'ordinateur possède toutes les informations nécessaires pour le calcul des deux racines? **OUI**.
- Est ce que l'ordinateur va effectuer normalement ces formules jusqu'à calcul de  $X_1$  et  $X_2$ ? **OUI**.
- Est ce que l'ordinateur donnera les valeurs exactes de  $X_1$  et  $X_2$ ? **NON, NON** et **NON**. Ceci est dû par le fait que la formule qui calcule  $\Delta$  est erroné puisque  $\Delta = B^2 - 4AC$ .

C'est-à-dire que l'ordinateur ne pourra jamais détecter que cette formule est fausse.

En conclusion, l'ordinateur est une machine qui permet d'automatiser le traitement de l'information, suivant les commandes et les instructions que lui soumet l'homme. Autrement dit, si une commande est fausse, le résultat n'est pas garanti.

### 3. Méthode de résolution d'un problème

La résolution d'un problème, en vue de son informatisation, nécessite une préparation d'un plan d'action que doit exécuter votre ordinateur. Les étapes à suivre pour l'élaboration de ce plan sont:



1. **Identification du problème**: cette étape consiste à rassembler les différentes informations et hypothèses (données) et connaître exactement ce qui est attendu comme résultat.
2. **Analyse et réflexion**: l'analyse va déterminer les différentes solutions et actions qui nous mèneront aux résultats attendus.  
En évaluant leurs avantages, leurs inconvénients ainsi que les moyens nécessaires à mettre en œuvre on pourra alors déterminer la meilleure solution.
3. **Plan d'action**: ayant décidé de la meilleure solution à adopter, il va falloir écrire, dans un langage conventionnel, séquentiellement les actions élémentaires que devrait exécuter l'ordinateur. Ceci constitue l'**ALGORITHME**.
4. **Programmation**: l'algorithme n'étant pas compréhensible par l'ordinateur, cette dernière étape consistera à traduire l'algorithme en un langage de programmation évolué, tel que le langage C. Le résultat de cette traduction s'appelle un programme.

## 4. Définition d'un algorithme

Dans cette section nous allons essayer d'expliquer ce que c'est qu'un algorithme.

### 4.1. Problème

Soit à calculer la moyenne des étudiants, où chaque étudiant possède 3 notes d'examen (A, B et C).

### 4.2. Analyse

Pour résoudre ce problème, décomposons-le comme suit:

1. Collecter les trois notes d'un étudiant. Par exemple:

$$A = 12,5 \quad B = 10 \quad C = 11$$

2. Calculer la moyenne de cet étudiant en utilisant la formule suivante:

$$moy = \frac{A + B + C}{3}$$

3. Afficher la moyenne.

### 4.3. Algorithme

C'est l'étape la plus importante pour la résolution d'un problème. Elle permet de détailler le processus que doit entreprendre un ordinateur pour arriver au résultat. Un algorithme représente la méthode de résolution d'un problème. Il est constitué d'un ensemble d'instructions ou de commandes, placées dans un ordre d'exécution particulier, nécessaire à la bonne marche du processus de calcul.

En général la forme d'un algorithme se présente comme suit:

```

Algorithme Nom ;
var
  Déclaration ;
Début
  Traitement ;
Fin.

```

- Un algorithme commence toujours par le mot '**Algorithme**' suivi de son nom. Ce dernier indique, en général, ce que fait cet algorithme.
- la partie déclaration est très importante, car elle indique les différentes données, variables et structures qui seront traitées, ainsi que leur type. Nous développerons cette partie plus loin dans ce chapitre.
- Le mot '**Début**' indique le début du traitement.
- La partie traitement est constituée d'un ensemble d'actions, nécessaire à la résolution du problème. Chaque action représente une opération.
- Le mot '**Fin**' suivi d'un **point '.'** indique la fin du traitement (ou de l'algorithme).

Pour mieux illustrer ceci, écrivons l'algorithme de calcul de la moyenne d'un étudiant. Ceci nécessite l'utilisation de quatre (4) variables; A, B, C et Moy.

### 4.3.1. Déclaration

Pour représenter une donnée (non constante) il faut utiliser un nom de variable pour le stockage de la valeur. Le nom de la variable représente l'adresse mémoire où se trouve la donnée correspondante. Elle doit toujours commencer par une lettre alphabétique.

EX:     **a** = 5  
           **delta** = alpha + b - 5  
           **y** =  $x_1 + x_2$

$2x = y + w$

Dans ce cas ' **2x** ' n'est pas une variable puisqu'elle commence par un chiffre

$x + y = w$

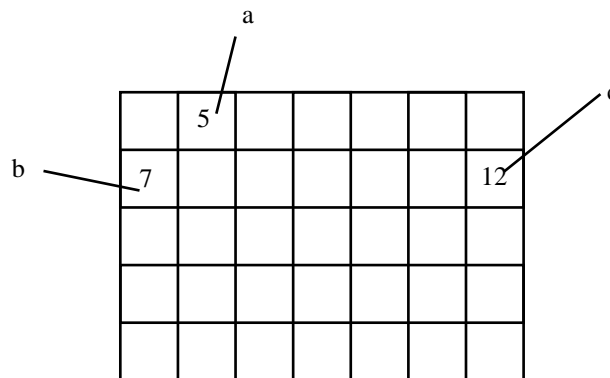
Cette écriture est fautive. La variable, où stocker la valeur, se trouve toujours à gauche du signe égal.

Dans notre cas, à gauche, il y a une expression. Il faut plutôt écrire :

**w** =  $x + y$

Schématiquement, la mémoire de travail (RAM) de l'ordinateur ressemble à une armoire contenant plusieurs cases, où chaque case (mémoire) représente une variable.

Exemple: additionner la variable  $a=5$  et la variable  $b=7$ . Le résultat sera rangé dans la variable  $c = a + b$ , soit 12.



Toute variable (identificateur) utilisée dans un algorithme doit être déclarée avant le traitement. Ceci consiste à indiquer son nom et son type pour que l'ordinateur lui réserve une case mémoire.

La syntaxe pour déclarer une variable est la suivante:

Syntaxe:

Identité

identificateur : Type ;

Si plusieurs variables possèdent le même type, on peut les déclarer en même temps, en les séparant par une virgule:

Syntaxe:

Identité

Ident1, Ident2, Ident3, ... : Type ;

Remarque:

Vous remarquerez qu'à la fin de la déclaration, il y a un point-virgule « ; ». Ce dernier indique la fin d'une déclaration.

Dans ce qui suit, le point virgule représentera la fin d'une action.

Identité : représente l'identité de l'identificateur:

- Var: pour les variables
- Const: pour les constantes
- Tableau : pour les tableaux
- etc...

Type : représente le type des données:

- Réel.
- Entier.
- Booléen.
- Caractère.
- etc...

Nous définirons ces différents types au fur et à mesure de leurs utilisations.

#### 4.3.2. Traitement

Le traitement est une suite d'actions qui manipulent généralement des expressions. Ces dernières peuvent être une constante, une variable ou la combinaison des deux à l'aide d'opérateurs: arithmétique, relationnel, logique, chaîne. Les 3 derniers types seront exposés au fur et à mesure de leur utilisation.

Les opérateurs arithmétiques: On distingue en général 5 opérateurs:

- L'addition (+).
- La soustraction (-).
- La multiplication (\*).
- La division (/).
- La puissance (^).

Les expressions seront écrites suivant les conventions de l'algèbre courante sur une ligne. Les parenthèses internes sont évaluées en premier en obéissant à la règle des priorités des opérations suivantes:

1. La puissance.
2. La multiplication et la division en commençant par la plus à gauche.
3. L'addition et la soustraction en commençant par la plus à gauche.

Soit l'exemple suivant:

$$y = \frac{3x + 1}{2p^2 - \frac{7}{8z + x}}$$

La traduction intégrale de cette équation, en informatique, donnera:

$$y = 3 * x + 1 / 2 * p^2 - 7 / 8 * z + x$$

Si on applique la règle des priorités informatiques à cette équation, elle sera évaluée comme suit:

$$y = 3x + \frac{p^2}{2} - \frac{7z}{8} + x$$

Ceci montre que l'équation reconstruite est totalement différente de l'équation originale. Pour éviter ce genre d'erreur, on fait appel aux parenthèses comme suit:

$$y = (3 * x + 1) / (2 * p^2) - 7 / (8 * z + x)$$

Dans la partie traitement, on sera appelé à utiliser différentes actions élémentaires.

## 5. Instructions de base

### 5.1. Instruction d'affectation

Dans un algorithme, on est appelé à sauvegarder les données et les résultats intermédiaires dans des variables. L'attribution d'une valeur à une variable peut se faire par affectation ou par lecture.

Syntaxe 1 :

Variable := expression ;

Syntaxe 2 :

Variable ← expression ;

Le « := » et « la flèche inverse » représentent le symbole d'affectation. La valeur issue de l'expression qui se trouve à droite est affectée à la variable qui se trouve à gauche.

La variable, qui reçoit le résultat de l'expression, doit être toujours à droite du signe d'affectation.

Avec l'instruction d'affectation, on peut maintenant écrire l'algorithme qui calcule la moyenne d'un étudiant.

Algorithme calculMoyenne ;	1
var	2
A, B, c, moy : Réel ;	3
Début	4
A ← 12,5 ;	5
B ← 10 ;	6
c ← 11 ;	7
moy ← ( A + B + c ) / 3 ;	8
Fin.	9

1. Le nom de l'algorithme est « CalculMoyenne »
2. Le mot « **Var** » Indique qu'on va déclarer des variables.
3. Les variables utilisées dans le traitement sont déclarées comme étant de type réel.
4. « **Début** » indique la fin des déclarations et le début du traitement.
5. La valeur « **12,5** » est affectée à la variable « **A** ». C'est-à-dire que la case mémoire, réservée pour cette variable, reçoit la valeur 12,5.
6. La valeur « **10** » est affectée à la variable « **B** ».



7. La valeur « 11 » est affectée à la variable « c ».
8. Le résultat, issu du calcul de la formule, est affecté à la variable « moy ».
9. « Fin. » indique la fin du traitement et de l'algorithme.

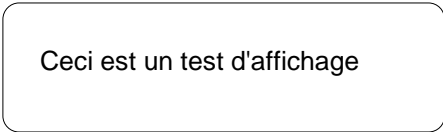
## 5.2. Instruction d'affichage

Dans l'algorithme précédent, la moyenne ne sera pas visualisée sur l'écran. Pour faire, il faut utiliser l'instruction d'affichage:

Syntaxe:

ECRIRE ( Expressions ) ;

où « Expressions » peut être un commentaire, une constante, une variable, une expression ou une combinaison de ces derniers. Le commentaire est une chaîne de caractères délimitée par deux apostrophes ou deux guillemets. Pour afficher, ce qui suit, sur écran:



Ceci est un test d'affichage

L'instruction devra s'écrire comme suit:

Exemple:

Ecrire ( " Ceci est un test d'affichage " ) ;

L'algorithme suivant affichera sur l'écran un message suivi d'une valeur

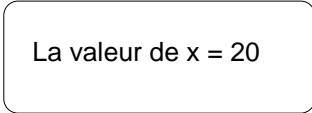
Exemple:

---

```

Algorithme Affichage ;
var
  x : Entier ;
Début
  x ← 20 ;
  Ecrire ( ' La valeur de x = ' , x ) ;
Fin.
```

---



La valeur de x = 20

Pour afficher la moyenne d'un étudiant dans l'algorithme « CalculMoyenne », il faut écrire l'instruction « Ecrire » comme suit:

---

```

Algorithme calculMoyenne ;
Var
  A, B, c, moy : Réel ;
Début
  A ← 12,5 ;
  B ← 10 ;
  c ← 11 ;
  moy ← ( A + B + c ) / 3 ;
  Ecrire ( ' La moyenne = ' , moy ) ;
Fin.
```

---

## 5.3. Instruction de lecture

L'algorithme précédent calcul la moyenne d'un seul étudiant. Dans ce cas, pour calculer la moyenne d'un autre étudiant, il faut modifier l'algorithme. C'est-à-dire, supprimer les notes de l'étudiant précédent pour les remplacer par les notes de l'étudiant suivant.

Ceci montre que cet algorithme ne présente pas la meilleure solution:

- Il serait fastidieux de porter les modifications pour chaque étudiant dans le cas où l'on doit calculer la moyenne d'un nombre important d'étudiants.
- Si un novice en informatique devra utiliser ce programme, il ne saura pas où porter les modifications.
- l'affichage est tellement rapide que l'on n'a pas le temps de visualiser le résultat.

Pour pallier ces inconvénients, nous utiliserons l'instruction de lecture « Lire ».

Syntaxe:

LIRE ( Liste de variables ) ;

Où, liste de variables est une suite de variables séparées par une virgule.

Cette instruction met en attente de lecture l'ordinateur. Pour lire les données, l'utilisateur doit saisir ou introduire les données, via le clavier par exemple. L'algorithme suivant affichera sur l'écran la somme de deux valeurs introduites au clavier :

Exemple:

Algorithme Lecture ;	
var	
x, y : Réel ;	
Début	
Ecrire ( ' Donnez deux valeurs : ' ) ;	1
Lire ( x , y ) ;	2
Ecrire ( ' La somme est : ' , x + y ) ;	3
Fin.	

Donnez deux valeurs :  
12.5 13  
La somme est : 25.5

1. L'instruction d'affichage « Ecrire » affiche le message (ou le commentaire) « Donnez deux valeurs : »
2. L'instruction de lecture « Lire » suspend l'exécution pour donner le temps à l'utilisateur de saisir autant de valeurs qu'il y a de variables entre parenthèses.
3. Après introduction des deux valeurs, il sera affiché le commentaire « La somme est : » suivi de la valeur de l'expression « x + y »

En introduisant cette instruction dans 'CalculMoyenne', l'algorithme montre que pour calculer la moyenne d'un étudiant, il suffit d'introduire les données au clavier sans avoir à le modifier.

Algorithme calculMoyenne ;	
Var	
A, B, c, moy : Réel ;	
Début	
Ecrire ( ' Donnez les trois notes : ' ) ;	
Lire ( A , B , c ) ;	
moy ← ( A + B + c ) / 3 ;	
Ecrire ( ' La moyenne = ' , moy ) ;	
Fin.	

Donnez les trois notes :  
10, 12.5, 10.5  
La moyenne = 11

#### 5.4. Instruction de condition simple

Elle permet d'exécuter une ou plusieurs actions, suivant une condition. Cette dernière est une expression logique qui utilise les opérateurs relationnels suivants:

<	Inférieur,
<=	Inférieur ou égal,
< >	Différent,
=	Egal,
>	Supérieur,
>=	Supérieur ou égale.

Syntaxe de l'instruction de condition simple :

- SI ( condition vraie ) ALORS action;

Ou bien

- SI ( condition vraie ) ALORS  
action;

L'action n'est exécutée que si la condition est vraie.

Exemple:

Algorithme Branchement ;	
var	
x,y : Réel ;	
Début	
Ecrire ( ' Donnez deux valeurs :' ) ;	1
Lire ( x , y ) ;	2
Si ( x > y ) Alors	
Ecrire ( ' x est plus grand que y ' ) ;	3
Fin.	

Donnez deux valeurs :  
100, 99  
x est plus grand que y

1. « Ecrire » affiche le commentaire « Donnez deuxvaleurs : »
2. « Lire » suspend l'exécution pour donner le temps à l'utilisateur de saisir deux valeurs (x et y).
3. Après introduction des deux valeurs, l'instruction « SI » sollicite le processeur pour vérifier si la condition est vraie, c'est-à-dire si x est supérieur à y.
  - SI la condition est vraie, le commentaire « x est plus grand que y » sera affiché.
  - Dans le cas contraire, le commentaire ne sera pas affiché.

Il se peut que la condition « Si » concerne plus d'une action. Dans ce cas, sa syntaxe est la suivante :

Syntaxe:

SI ( condition vraie ) ALORS Début    action 1 ; action 2 ; .... Fin;

Ou bien

```
SI ( condition vraie ) ALORS
  Début
    action 1 ;
    action 2 ;
    .....
  Fin;
```

Pour indiquer à l'ordinateur que la condition « Si » concerne plus d'une action, il faut mettre toutes les actions entre « Début.....Fin ; ». Les actions se trouvant entre Début et Fin ne sont exécutées que si la condition est vraie.

Exemple :

```
SI Moy >= 10 ALORS
  Début
    Ecrire ( ' La moyenne = ' , moy ) ;
    Ecrire ( ' Cet étudiant est admis ' ) ;
  Fin;
Ecrire ( ' Fin du test ' ) ;
```

- SI la moyenne est supérieure ou égale à 10, l'ordinateur affichera

La moyenne = 12.5 :  
Cet étudiant est admis  
Fin du test

- Si la moyenne est inférieure à 10, l'ordinateur n'exécutera pas les deux actions qui se trouvent entre « Début » et « Fin ; ». Il affichera donc :

Fin du test

Dans cet exemple, l'instruction « SI » se termine au point-virgule de la balise « Fin ; ».

Dans l'exemple suivant, nous avons délibérément supprimé les balises « Début » et « Fin ; ».

```
SI Moy >= 10 ALORS
  Ecrire ( ' La moyenne = ' , moy ) ;
  Ecrire ( ' Cet étudiant est admis ' ) ;
  Ecrire ( ' Fin du test ' ) ;
```

- SI la moyenne est supérieure ou égale à 10, l'ordinateur affichera

La moyenne = 12.5 :  
Cet étudiant est admis  
Fin du test

- Si la moyenne est inférieure à 10, l'ordinateur n'exécutera pas la première action. Il affichera donc :

Cet étudiant est admis  
Fin du test

Dans cet exemple, l'instruction « SI » se termine à au point-virgule de la première action « Ecrire ( ' La moyenne = ' , moy ) ; ».

Pour mieux illustrer cette instruction, prenant l'exemple de la résolution d'une équation du second degré  $Ax^2 + Bx + c = 0$ , en supposant que A est NON nulle:

Analyse :

1. le programme devra afficher sur l'écran un message pour nous inviter à introduire les valeurs de A, B et C:
2. Ensuite, l'ordinateur devra se mettre en attente pour lire les données introduites via le clavier:
3. Après avoir collecté les données, l'ordinateur va calculer  $\Delta$ :
4. Finalement, une décision sera prise par l'ordinateur suivant le signe de  $\Delta$ .

Ayant analysé et énuméré les différentes étapes que doit suivre l'ordinateur, nous allons écrire maintenant l'algorithme:

---

```

Algorithme Racines ;
  Var
    a , b , c , delta , x , x1 , x2 : Réel ;
Début
  Ecrire ( ' Donnez a , b et c ' ) ;
  Lire ( a , b , c ) ;
  delta ← b * b - 4 * a * c ;

  Si ( delta < 0 ) Alors
    Ecrire ( ' Pas de solution ' ) ;

  Si ( delta = 0 ) Alors
    Début
      x ← - b / ( 2 * a ) ;
      Ecrire ( ' Une solution double x = ' , x ) ;
    Fin ;

  Si ( delta > 0 ) Alors
    Début
      x1 ← ( - b - racine ( delta ) ) / ( 2 * a ) ;
      x2 ← ( - b + racine ( delta ) ) / ( 2 * a ) ;
      Ecrire ( ' Il y a deux solution : x1 = ' , x1 , ' x2 = ' , x2 ) ;
    Fin ;
Fin.

```

---

Cet algorithme permet de calculer les racines d'une équation du second degré, mais ne constitue pas la meilleure solution. C'est-à-dire, si delta est négatif, l'ordinateur affichera qu'il n'y a pas de solution. Mais, malgré que le premier test soit vrai, l'ordinateur va quand effectuer les 2 tests « si delta = 0 » et « si delta > 0 », alors qu'il aurait dû terminer l'exécution. Ceci prouve la stupidité de l'ordinateur.

### 5.5. Instruction de décision alternée

En introduisant l'instruction de décision alternée, on peut solutionner cet inconvénient. C'est une instruction qui offre une alternative dans le choix. C'est-à-dire, si la condition est vraie, elle exécute une ou plusieurs actions sinon elle exécute d'autres actions.

Syntaxe:

```
SI ( condition vraie ) ALORS
  Début
    action 1 ;
    action 2 ;
  Fin
SINON
  Début
    action 3 ;
    action 4 ;
    action 5 ;
  Fin ;
```

Les actions 1 et 2, ne sont exécutées que si la condition est vraie, sinon ce sont les actions 3, 4 et 5 qui seront exécutées.

Remarque:

Avant le mot-clé « sinon », il ne doit jamais y avoir de point-virgule. Par exemple, le premier mot-clé « Fin » ne possède pas de point-virgule.

---

```
Algorithme Condition ;
var
  x,y : Réel ;
Début
  Ecrire ( ' Donnez deux valeurs: ' );
  Lire ( x , y );
  Si ( x > y ) Alors
    Ecrire ( ' x est plus grand que y ' )
  Sinon
    Ecrire ( ' x est plus petit ou égal à y ' );
Fin.
```

---

Donnez deux valeurs:  
99 100  
x est plus petit ou égal à y

Remarque:

L'instruction « Ecrire » qui se situe avant le « sinon » ne se termine pas par le point-virgule « ; ».

L'instruction alternative peut-être utilisée dans l'algorithme « Racine » pour éviter à l'ordinateur de faire des tests inutiles

1. Si  $\Delta$  est négatif, l'ordinateur affichera qu'il n'y a pas de solution, puis termine l'exécution,
2. Sinon, si  $\Delta$  est nul, l'ordinateur calcule la racine double, affiche le résultat puis termine l'exécution,
3. Sinon, à ce niveau  $\Delta$  n'est ni négatif, ni nul, pas besoin d'ajouter un troisième test puisque  $\Delta$  est automatiquement positif. L'ordinateur calcule directement (sans test) les deux solutions, affiche le résultat puis termine l'exécution.

---

```

Algorithme Racines ;
Var
  a , b , c , delta , x , x1 , x2 : Réel ;
Début
  Ecrire ( ' Donnez a , b et c ' ) ;
  Lire ( a , b , c ) ;
  delta ← b * b - 4 * a * c ;
  Si ( delta < 0 ) Alors
    Ecrire ( ' Pas de solution ' )
  Sinon
    Si ( delta = 0 ) Alors
      Début
        x ← - b / ( 2 * a ) ;
        Ecrire ( ' Une solution double x = ' , x ) ;
      Fin
    Sinon
      Début
        x1 ← ( - b - racine ( delta ) ) / ( 2 * a ) ;
        x2 ← ( - b + racine ( delta ) ) / ( 2 * a ) ;
        Ecrire ( ' Il y a deux solution : x1 = ' , x1 , ' x2 = ' , x2 ) ;
      Fin ;
  Fin.

```

---

### 5.6. Instruction de décision multiple

L'instruction « Cas » est une généralisation de l'instruction « Si ». Elle permet d'exécuter une instruction parmi plusieurs, en s'orientant selon, les différentes valeurs que peut prendre une expression.

Syntaxe:

```

CAS valeur de l'expression PARMi
  cas 1 : action 1 ;
  cas 2 : action 2 ;
  cas 3 : action 3 ;
SINON
  action 4 ;
FIN ;

```

La clause « Sinon » est facultative. C'est-à-dire, si aucun cas n'est vérifié, alors l'ordinateur exécutera action 4.

Pour cette instruction, il n'y a pas lieu de supprimer le point-virgule avant le « Sinon ».

Les deux algorithmes suivants explicitent son utilisation. Ils permettent de donner le nombre de jours dans un mois.

---

Algorithme1:

```

Algorithme JourMois ;
Var
  n : Entier ;
Début
  Ecrire ( ' Donnez le mois en chiffre: ' ) ;
  Lire ( n ) ;
  Cas n Parmi ;
    4, 6, 9, 11      : Ecrire ( ' Le nombre de jours : ' , 30 ) ;
    2                : Ecrire ( ' Le nombre de jours : ' , 28 , ' ou ' , 29 ) ;
    1, 3, 5, 7, 8, 10, 12 : Ecrire ( ' Le nombre de jours : ' , 31 ) ;
  Fin ;
Fin.

```

---

---

Algorithme2:

```

Algorithme JourMois ;
Var
  n : Entier ;
Début
  Ecrire ( ' Donnez le mois en chiffre: ' ) ;
  Lire ( n ) ;
  Cas n Parmi ;
    4, 6, 9, 11      : Ecrire ( ' Le nombre de jours : ' , 30 ) ;
    2                : Ecrire ( ' Le nombre de jours : ' , 28 , ' ou ' , 29 ) ;
    1, 3, 5, 7, 8, 10, 12 : Ecrire ( ' Le nombre de jours : ' , 31 ) ;
  Sinon
    Ecrire ( ' Le mois doit être entre 1 et 12 ' ) ;
  Fin ;
Fin.

```

---

## 6. Structures répétitives

Les instructions de boucles sont utilisées pour des traitements répétitifs. C'est-à-dire qu'ils permettent d'exécuter un traitement plusieurs fois.

Comme tout traitement, une boucle doit s'exécuter un certain nombre de fois. Il faut donc, après chaque exécution, prendre une décision de fin de boucle. Ceci se fait suivant une condition de sortie de la boucle.

### 6.1. Instruction de boucle « Pour »

La boucle « Pour » permet d'exécuter plusieurs fois un même traitement avec différentes données. Sa syntaxe se présente comme suit :

Syntaxe :

```

POUR VarCompteur := ValeurInitiale A ValeurFinal FAIRE
  Action;

```

où:

- VarCompteur est une variable numérique utilisée comme compteur de boucle. C'est-à-dire, elle compte le nombre de fois d'exécution de l'action.
- ValeurInitiale (ou VI) et ValeurFinal (ou VF) sont des valeurs ou expressions représentant, respectivement, la valeur initiale du compteur de boucle et sa valeur finale.

Pour mieux comprendre le fonctionnement de la boucle « Pour », voyant l'exemple suivant. Soit un algorithme qui affiche le carré des N premiers entiers naturels.

---

```

Algorithme BouclePour ;
Var
  N , i , X : Entier ;
Début
  Ecrire ( ' Donnez la valeur de N ' ) ;
  Lire ( N ) ;
  Pour i ← 1 à N faire
    Début
      X ← i * i ;
      Ecrire ( ' Le carré de ' , i , ' = ' , i * i ) ;
    Fin ;
  Ecrire ( ' Merci ' ) ;
Fin.

```

---



Nous allons définir l'exécution de cet algorithme, étape par étape en s'aidant de la trace. Cette dernière se présente sous forme d'un tableau qui retrace toutes les étapes d'exécution. Ce tableau est composé de plusieurs colonnes :

- Une colonne pour chaque variable : Ces colonnes retracent l'évolution du contenu des variables, qui se trouvent dans « la RAM », tout le long de l'exécution.
- Une colonne pour l'affichage : Cette colonne retrace l'évolution de l'affichage sur « l'écran », tout le long de l'exécution.

RAM			Ecran
N	i	X	Affichage
			Donnez la valeur de N
4	1	1	Le carré de 1 = 1
	2	4	Le carré de 2 = 4
	3	9	Le carré de 3 = 9
	4	16	Le carré de 4 = 16
	5		Merci

1. Au départ et après déclaration, l'algorithme (ou le programme) affiche le message « Donnez la valeur de N », (N premiers entiers naturels), pour inviter l'utilisateur à introduire une valeur..
2. Ensuite, l'utilisateur introduit une valeur. Par exemple « 4 ». Donc la valeur « 4 » sera affectée à la variable « N ».
3. A la rencontre de la boucle « Pour », la valeur initiale « 1 » sera affectée à la variable compteur « i ». Donc la valeur « 1 » sera affectée à la variable « i ». Autrement dit, la variable compteur sera initialisée à la valeur initiale « 1 ».
4. Automatiquement, le programme va vérifier si la variable compteur est inférieure ou égale ( $\leq$ ) à la valeur finale.
  - 4.1. Si c'est vrai, le programme exécutera la boucle. C'est-à-dire toutes les actions qui se trouvent entre « Début » et « Fin ; » :
    - Il calcule le carré de  $i$  ( $i * i$ ) puis l'affecte à la variable « X ».
    - Ensuite il affiche le message (résultat) : « Le carré de 1 = 1 »
    - A la fin de la boucle « Fin ; » :
      - La variable « i » sera automatiquement incrémentée de « 1 » ( $i := i + 1$ )
      - Ensuite, le programme retourne automatiquement à l'étape « 4. » pour vérifier s'il doit poursuivre l'exécution de la boucle. Etc...
  - 4.2. Si c'est non ( $i = 5$ ), le programme sort de la boucle pour continuer l'exécution du programme, tout de suite après la fin de la boucle.

En conclusion :

- Si le compteur « i » est inférieur ou égal à la valeur finale, le programme exécute les actions de la boucle.
- A la fin de chaque exécution, des actions, de la boucle, le programme incrémente automatiquement le compteur et ainsi de suite.
- Lorsque le compteur devient supérieur à la valeur finale, le programme quitte la boucle puis continue l'exécution à partir de la première action après la boucle.
- Si la  $V_i > V_f$ , la boucle ne sera jamais exécutée, comme l'illustre l'exemple suivant : l'action « Ecrire ( i , ' Boucle ' ) ; » ne sera jamais exécutée.

```

Algorithme PasDeBoucle ;
Var
  i : Entier ;
Début
  Ecrire ( ' Début de la boucle : ' ) ;
  Pour i ← 5 à 1 Faire
    Ecrire ( i , ' Boucle ' ) ;
  Ecrire ( ' Fin de la boucle ' ) ;
Fin.

```

Sur écran nous aurons

Début de la boucle  
Fin de la boucle

- Si la  $VI \leq VF$ , la boucle sera exécutée :

$VF - VI + 1$  fois

Comme le montre l'exemple suivant.

<pre> Algorithmme Boucle ; Var   i : Entier ; Début   Ecrire ( ' Début de la boucle : ' ) ;   Pour i ← 1 à 5 Faire     Ecrire ( ' Boucle ', i ) ;   Ecrire ( ' Fin de la boucle ' ) ; Fin. </pre>	<p style="font-size: small; margin: 0;">Sur écran nous aurons</p> <div style="border: 1px solid black; border-radius: 15px; padding: 10px; margin: 10px auto; width: 80%;"> <pre> Début de la boucle Boucle 1 Boucle 2 Boucle 3 Boucle 4 Boucle 5 Fin de la boucle </pre> </div>
---	--

- Si l'on veut exécuter la boucle telle que  $VI \geq VF$ , il faut indiquer à l'ordinateur qu'il faut décrémente le compteur à chaque exécution de la boucle. C'est-à-dire, l'ordinateur exécute la boucle puis, décrémente à chaque fois le compteur jusqu'à ce que ce dernier devienne inférieur à  $VF$ , comme le montre l'exemple suivant :

<pre> Algorithmme Boucle ; Var   i : Entier ; Début   Ecrire ( ' Début de la boucle : ' ) ;   Pour i ← 5 à 1 Vers le bas Faire     Ecrire ( ' Boucle ', i ) ;   Ecrire ( ' Fin de la boucle ' ) ; Fin. </pre>	<p style="font-size: small; margin: 0;">Sur écran nous aurons</p> <div style="border: 1px solid black; border-radius: 15px; padding: 10px; margin: 10px auto; width: 80%;"> <pre> Début de la boucle Boucle 5 Boucle 4 Boucle 3 Boucle 2 Boucle 1 Fin de la boucle </pre> </div>
---	--

Pour calculer la moyenne de  $N$  étudiants, en utilisant l'algorithmme 'CalculMoyenne' (voir ci-dessus), il faut ordonner l'exécution du programme  $N$  fois.

Par contre, en utilisant la boucle « Pour » on peut calculer la moyenne de plusieurs étudiants en lançant demandant l'exécution du programme, une seule fois, comme le montre l'algorithmme suivant :

<pre> Algorithmme calculMoyenne ; Var   A, B, c, moy : Réel ;   n, i : Entier ; Début   Ecrire ( ' Donnez le nombre d'étudiants : ' ) ;   Lire ( n ) ;   Pour i ← 1 à n Faire     Début       Ecrire ( ' Donnez les trois notes ' ) ;       Lire ( A , B , c ) ;       moy ← ( A + B + c ) / 3 ;       Ecrire ( ' La moyenne = ', moy ) ;     Fin;   Fin. </pre>	
--	--

Tout de suite après avoir affiché la première moyenne, le programme vous demande d'introduire les 3 notes du prochain étudiant, sans avoir à redemander l'exécution du programme et ainsi de suite jusqu'au Nième étudiant.

*La boucle « Pour » est souvent sollicitée lorsque le nombre de fois d'exécution d'un traitement (le nombre de boucles) est connu. Dans le cas contraire on fait appel à la boucle « TantQue » ou « Répéter ».*

## 6.2. Instruction de boucle « TantQue »

Cette instruction est utilisée dans le cas où le nombre de boucle à exécuter n'est pas connu. Elle utilise une condition pour décider s'il faut, encore, exécuter un traitement (une boucle).

Syntaxe:

```
TANTQUE (Condition vraie) FAIRE
    Action ;
```

A la rencontre de l'instruction « TantQue » :

1. « TantQue » évalue la condition :
  - 1.1. Si elle est vraie :
    - 1.1.1. Le programme exécute le ou les actions qui se trouvent dans la boucle,
    - 1.1.2. A la fin du traitement (ou de la boucle), il retourne, **automatiquement**, à l'étape 1, pour réévaluer la condition.
  - 1.2. Si elle est fautive (non vraie), le programme sort de la boucle puis continue l'exécution à la première action qui vient tout de suite après la boucle.

L'exemple suivant illustre le fonctionnement de la boucle « TantQue ».

---

```
Algorithme Tantque ;
var
  x : Entier ;
Début
  Ecrire ( ' Donnez la valeur de x: ' ) ;
  Lire ( x ) ;
  Ecrire ( ' début de la boucle ' ) ;
  TantQue x > 0 Faire
    Début
      Ecrire ( ' x = ' , x ) ;
      x ← x - 1 ;
    Fin ;
  Ecrire ( ' Fin de la boucle ' ) ;
Fin.
```

---

Sur écran nous aurons

```
Donnez la valeur de x
3
Début de la boucle
x = 3
x = 2
x = 1
Fin de la boucle
```

L'exemple suivant est un algorithme qui permet de calculer le carré des N premiers nombres entiers. On remarquera dans cet exemple, que la boucle « TantQue » peut remplacer la boucle « Pour », avec les automatismes en moins :

- La variable compteur « i » est initialisée avant la boucle.
- La condition de la boucle est visible au début de la boucle.
- L'incréméntation du compteur doit apparaître dans l'algorithme (non automatique).

---

```

Algorithme Carré ;
var
  i , N : Entier ;
Début
  Ecrire ( ' Donnez la valeur de N: ' ) ;
  Lire ( N ) ;
  i ← 1 ;
  Ecrire ( ' début de la boucle ' ) ;
  TantQue i <= N Faire
    Début
      Ecrire ( i , ' au carré = ' , i * i ) ;
      i ← i + 1 ;
    Fin ;
  Ecrire ( ' Fin de la boucle ' ) ;
Fin.

```

---

Sur écran nous aurons

```

Donnez la valeur de N
5
Début de la boucle
1 au carré = 1
2 au carré = 4
3 au carré = 9
4 au carré = 16
5 au carré = 25
Fin de la boucle

```

*La boucle « TantQue » est sollicitée lorsque le nombre de boucles est inconnu. Toutefois, il est possible de l'utiliser lorsque le nombre de boucles est connu.*

Soit à écrire un algorithme qui permet de calculer la somme d'une suite de valeurs non nuls. L'algorithme s'arrête lorsque l'on saisie la valeur zéro.

---

```

Algorithme Suite ;
var
  x , s : Réel ;
Début
  S ← 0 ;
  x ← 1 ;
  TantQue x <> 0 Faire
    Début
      Ecrire ( ' Donnez une valeur : ' ) ;
      Lire ( x ) ;
      s ← s + x ;
    Fin ;
  Ecrire ( ' La somme = ' , s ) ;
Fin.

```

---

Sur écran nous aurons

```

Donnez une valeur :
5
Donnez une valeur :
12
Donnez une valeur :
-3
Donnez une valeur :
2
Donnez une valeur :
0
La somme = 16

```

On remarquera dans cet exemple qu'il est impératif d'initialiser la valeur de x à une valeur différente de zéro ( 1 dans notre exemple) pour forcer le programme à exécuter au moins une fois la boucle « TantQue ».

Dans ce cas de figure, il est souvent fait appel à l'instruction de boucle « Répéter ».

### 6.3. Instruction de boucle « Répéter »

C'est une instruction de boucle au même titre que « TantQue », sauf que la condition est évaluée à la fin de la boucle.

Syntaxe:

```

Répéter
  Action1;
  Action2;
  .....
Jusqu'à ( Condition vraie ) ;

```

1. L'ordinateur exécute d'abord, sans aucune condition, la boucle (Action1, Action2, ...),
2. A la fin de la boucle, il évalue la condition,
  - 2.1. Si elle est fausse :
    - 2.1.1. l'ordinateur exécute la boucle,
    - 2.1.2. Retourne à l'étape 1 pour réévaluer la condition,
  - 2.2. Si elle est vraie :
    - 2.2.1. l'ordinateur sort de la boucle puis, continue l'exécution du programme à l'instruction qui vient tout de suite après « Jusqu'à »

Dans l'exemple suivant, qui reprend le même exercice que précédemment, on remarque que la boucle « Répéter » est exécutée au moins une fois sans avoir à initialiser la variable « x » à une valeur différente de zéro.

De plus, on n'a pas besoin de mettre les « actions » entre « Début » et « Fin ; »

	Sur écran nous aurons
<pre> Algorithmme Suite ; var   x , s : Réel ; Début   S ← 0 ;   Répéter     Ecrire ( ' Donnez une valeur : ' ) ;     Lire ( x ) ;     s ← s + x ;   Jusqu'à x = 0 ;   Ecrire ( ' La somme = ' , s ) ; Fin. </pre>	<pre> Donnez une valeur : 5 Donnez une valeur : 12 Donnez une valeur : -3 Donnez une valeur : 2 Donnez une valeur : 0 La somme = 16 </pre>

L'exemple suivant, illustre la particularité de la boucle « répéter ».

	Sur écran nous aurons
<pre> Algorithmme Répéter ; var   x : Réel ; Début   Ecrire ( ' Donnez la valeur de x : ' ) ;   Lire ( x )   Ecrire ( ' début de la boucle ' ) ;   Répéter     Ecrire ( ' x = ' , x ) ;     x ← x - 1 ;   Jusqu'à x &lt;= 0 ;   Ecrire ( ' Fin de la boucle ' ) ; Fin. </pre>	<pre> Donnez la valeur de x -1 Début de la boucle x = -1 Fin de la boucle </pre>

Dans cet exemple, on remarque que la boucle à été exécutée une fois malgré que x = -1.

L'instruction « Répéter » peut être utilisée dans l'algorithmme 'Carré' comme suit:

---

```

Algorithme Carré ;
var
  i, N : Entier ;
Début
  Ecrire ( ' Donnez la valeur de N: ' ) ;
  Lire ( N ) ;
  i ← 1 ;
  Ecrire ( ' début de la boucle ' ) ;
  Répéter
    Début
      Ecrire ( i , ' au carré = ' , i * i ) ;
      i ← i + 1 ;
    Fin ;
  Jusqu'à i > N ;
  Ecrire ( ' Fin de la boucle ' ) ;
Fin.

```

---

#### 6.4. Différences entre les boucles « Pour », « TantQue » et « Répéter »

Le choix de la boucle à utiliser dans un problème se fait de la manière suivante :

1. Si on connaît le nombre de boucle à effectuer, on utilisera une boucle « Pour ».
2. Si le nombre de boucles dépend d'une condition, on utilisera « TantQue » ou « Répéter » :
  - 2.1. « TantQue » vérifie la condition avant de commencer l'exécution de la boucle. Donc la boucle peut ne pas être exécutée.
  - 2.2. « Répéter » vérifie la condition après avoir exécuté la boucle. Donc la boucle est forcément exécutée une fois.
  - 2.3. « TantQue » est exécutée si la condition est « VRAIE ».
  - 2.4. « Répéter » est exécutée si la condition est « FAUSSE ».

### 7. Structures de données

Toutes les données qu'on a manipulées jusqu'à maintenant, ont été stockées dans une structure simple qu'est la variable. Leurs manipulations s'avèrent limitées lorsqu'il s'agit de stocker plusieurs données de même nature.

Par exemple, dans le programme, du calcul de la moyenne des étudiants, si l'on veut enregistrer la moyenne de 1000 étudiants, il nous faudra faire appel à 1000 variables. Donc :

- Il va falloir donner, à chacune des variables, un nom propre.
- De plus, il faut les mentionner une à une dans la partie des déclarations, ce qui est fastidieux à réaliser.

Pour pallier ces problèmes, le programmeur fait appel aux structures de données. Ces dernières permettent d'organiser les données, afin de simplifier leurs utilisations et de les gérer d'une manière efficace. Leur utilité principale réside dans le fait que l'on puisse enregistrer ou stocker des informations accessibles, pas l'utilisateur, plus tard.

Il existe plusieurs types de structure de données où chacune d'elles a :

- Sa manière de stocker les données.
- Un ensemble d'opérations qu'on peut appliquer aux données.

Les programmeurs font souvent appel à ces méthodes pour organiser les données d'une manière efficace. Exemple de structure de données :

- Les structures de données séquentielles (tableaux) ;
- Les structures de données linéaires (listes chaînées) ;
- Les arbres ;
- Les graphes.

Dans ce chapitre nous nous intéresserons aux structures de données séquentielles, soit les tableaux.

### 7.1. Les structures de données séquentielles.

Revenons à notre précédent problème qui consiste à enregistrer les 1000 moyennes pour une utilisation ultérieure, tel que la trace de la courbe de Gauss, l'écart type, ...

L'algorithme suivant n'enregistre que la moyenne du dernier étudiant traité :

---

```

Algorithme calculMoyenne ;
  Var
    a, b, c, moy : Réel ;
    n, i : Entier ;
  Début
    Ecrire ( ' Donnez le nombre d'étudiants : ' ) ;
    Lire ( n ) ;
    Pour i ← 1 à n Faire
      Début
        Ecrire ( ' Donnez les trois notes ' ) ;
        Lire ( a , b , c ) ;
        moy ← ( a + b + c ) / 3 ;
        Ecrire ( ' La moyenne = ' , moy ) ;
      Fin;
  Fin.

```

---

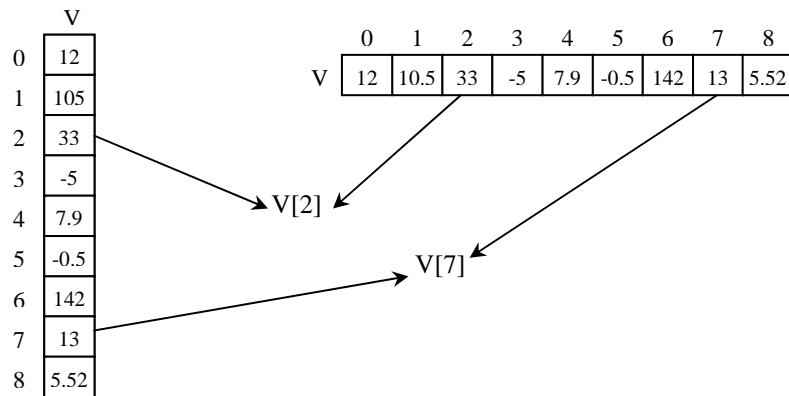
Pour pouvoir stocker toutes les moyennes, nous ferons appel à une structure de données, séquentielle, qu'est le tableau à une dimension.

#### 7.1.1. Les tableaux à une dimension (vecteur)

Une variable est composée d'une seule case mémoire, alors que le vecteur (tableau à une dimension) est une structure de données composée d'un ensemble de cases mémoires, où :

- Le vecteur doit avoir un nom. Par exemple, le vecteur V.
- Une case mémoire représente un élément du vecteur.
- Tous les éléments du vecteur sont de même type.
- Le nom de chaque élément est composé du nom du vecteur suivi d'un indice. Ce dernier indique la position de l'élément dans le vecteur. Par exemple, l'élément V[3].
- L'indice est indiqué entre crochets.

Un vecteur peut être représenté verticalement ou horizontalement, comme le montre l'exemple suivant :



- La description d'un élément ou plutôt pour accéder à un élément du tableau, on utilise le nom du vecteur suivi de son indice entre crochets.
- L'indice est toujours positif.
- L'indice du premier élément du tableau est zéro « 0 ». Le 3<sup>ème</sup> élément du vecteur est V [2].
- L'indice du dernier élément du tableau est « le nombre d'éléments composant le vecteur » moins « un » (9-1=8 dans notre exemple).
- Un élément du tableau est manipulé exactement comme une variable. On peut :
  - o Lui affecter une valeur,
  - o L'utiliser dans une expression
  - o Faire des tests dessus....

L'algorithme suivant illustre l'utilisation d'un vecteur pour le stockage de toutes les moyennes des étudiants :

---

```

Algorithme Moyenne ;
Var
  a, b, c, M : Réel ;
  n, i : Entier ;
  V : Tableau[1..50] de réel ;
Début
  Répéter
    Ecrire ( ' Donnez le nombre des étudiants <= 50 ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n>=1 ) et ( n<=50 )

  Pour i ← 0 à n-1 Faire
    Début
      Ecrire ( ' Donnez les trois notes ' ) ;
      Lire ( a , b , c ) ;
      M ← ( a + b + c ) / 3 ;
      V[ i ] ← M ;
    Fin;

  Pour i ← 0 à n - 1 Faire
    Ecrire( ' V [ ' , i , ' ] = ' , V[ i ] ) ;
Fin.

```

---



A la place de la variable « moy », cet algorithme utilise plutôt, un vecteur « V » pour enregistrer toutes les moyennes des étudiants. L'indice « i » d'une boucle « Pour » a été utilisé pour accéder aux éléments du vecteur.

**On remarque :**

1. La déclaration de tous les éléments du tableau est réalisée avec le mot-clé « Tableau » en indiquant entre parenthèses le minimum et le maximum d'éléments composant le vecteur « [1..50] ».

Cette technique permet d'éviter de déclarer les éléments du vecteur de la manière suivante :

[1], V[2], V[3], V[4], ....., V[49], V[50] : Réel (Ouf !)

2. La boucle répéter est utilisée, dans cet algorithme, pour s'assurer que le nombre des étudiants « n » doit être compris entre 1 et 50 « [1..50] ».
3. La boucle « Pour » a pour indice « i ». Ce dernier est utilisé pour indiquer la position d'un élément du vecteur.

Comme le premier élément du vecteur se trouve à la position zéro, l'indice variera de « 0 » à « n - 1 ».

4. Chaque moyenne est enregistrée, distinctement, dans les éléments du vecteur. La position de l'élément est indiquée par l'indice « i » (V[ i ]).
5. Cette boucle permet d'afficher les moyennes stockées dans le vecteur V.

**Note :** Dans ce qui suit, nous utiliserons la notation « /\*/\* » pour indiquer que c'est un commentaire explicatif d'une partie d'un algorithme.

#### 7.1.1.1. Comment remplir un vecteur

**Exemple 1 :** Faire l'algorithme qui remplit les éléments du vecteur T avec le carré de leur propre indice :

	0	1	2	3	4	5	6	7	8
T	0	1	4	9	16	25	36	49	64

Analysons la situation :

```

Pour i = 0 on a V[ 0 ] = 0 * 0 = 0
      i = 1 // V[ 1 ] = 1 * 1 = 1
      i = 2 // V[ 2 ] = 2 * 2 = 4
      i = 3 // V[ 3 ] = 3 * 3 = 9
      .....
      .....
      i = 8 // V[ 8 ] = 8 * 8 = 64

```

Donc, la formule générale pour remplir ce vecteur est :

$$V[ i ] = i * i$$

---

```

Algorithme carré ;
  Var
    n, i : Entier ;
    V : Tableau[1..50] de réel ;
Début
  Répéter
    Ecrire ( ' Donnez le nombre d'éléments <= 50 : ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 )

  Pour i ← 0 à n - 1 Faire
    V[ i ] ← i * i ;

  Pour i ← 0 à n - 1 Faire
    Ecrire( ' V [ ' , i , ' ] = ' , V [ i ] ) ; /**/ pour i = 3, il affichera « V[3] = 9 »
Fin.

```

---

**Remarque :** Dans cet algorithme, il est possible de regrouper les deux dernières boucles en une seule.

```

Pour i ← 0 à n - 1 Faire
  Début
    V[ i ] ← i * i ;
    Ecrire ( ' V [ ' , i , ' ] = ' , V [ i ] ) ;
  Fin;

```

**Exemple 2 :** Ecrire un algorithme qui permet de remplir un vecteur avec des valeurs saisies par l'utilisateur.

Analysons la situation :

Pour	i = 0	on saisie une valeur dans x	V[ 0 ] = x
	i = 1	//	V[ 1 ] = x
	i = 2	//	V[ 2 ] = x
	i = 3	//	V[ 3 ] = x
	.....		
	.....		
	i = N	//	V[ N ] = x

Donc, le traitement général pour remplir ce vecteur est :

- 1- Saisir la valeur à introduire dans le vecteur : Lire(x)
- 2- Stocker la valeur dans le vecteur : V[ i ] = x

---

```

Algorithme valeur ;
Var
  x : Réel ;
  n, i : Entier ;
  V : Tableau[1..50] de réel ;
Début
  Répéter
    Ecrire ( ' Donnez le nombre d'éléments <= 50 ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 )

  Pour i ← 0 à n - 1 Faire
    Début
      Ecrire ( ' Donnez une valeur ' ) ;
      Lire ( x ) ;
      V [ i ] ← x ;  /**/ la valeur lu est ensuite stockée dans le vecteur
    Fin;

  Pour i ← 0 à n - 1 Faire
    Ecrire ( ' V [ ' , i , ' ] = ' , V [ i ] ) ;
Fin.

```

---

**Remarque :** Dans cet algorithme, il est possible d'éliminer la variable X en stockant la valeur saisie directement dans le vecteur.

```

Pour i ← 0 à n - 1 Faire
  Début
    Ecrire ( ' Donnez une valeur ' ) ;
    Lire ( V [ i ] ) ;  /**/ La valeur lu est automatiquement stockée dans le vecteur
  Fin;

```

**Exemple 3 :** Ecrire un algorithme qui remplit un vecteur avec des valeurs saisies par l'utilisateur, puis affiche à la fin la somme des valeurs.

Analyse est presque identique à celle de l'exemple 2 ci-dessus :

- 1- Saisir et stocker la valeur dans le vecteur V
- 2- Ajouter la valeur stocker dans la somme :  $S = S + V[i]$

---

```

Algorithme valeur ;
Var
  s : Réel ;
  n, i : Entier ;
  V : Tableau[1..50] de réel ;
Début
  Répéter
    Ecrire ( ' Donnez le nombre d'éléments <= 50 ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 )

  Pour i ← 0 à n - 1 Faire  /**/ Cette boucle est utilisée pour remplir le vecteur
    Début
      Ecrire ( ' Donnez une valeur ' ) ;
      Lire ( V [ i ] ) ;
    Fin;

  s ← 0
  Pour i ← 0 à n - 1 Faire  /**/ Cette boucle est utilisée pour calculer la somme
    s ← s + V [ i ] ;

  Ecrire ( ' Somme = ' , s ) ;
Fin.

```

---

**Remarque** : Il est possible de remplacer les deux boucles par une seule. C'est-à-dire, faire la somme au fur et à mesure de la lecture des données.

```

s ← 0
Pour i ← 0 à n - 1 Faire
  Début
  Ecrire ( ' Donnez une valeur ' );
  Lire ( V [ i ] );
  s ← s + V [ i ];
Fin;

```

### 7.1.1.2. Manipulation de plusieurs vecteurs

**Exemple** : Ecrire l'algorithme qui multiplie chaque élément d'un vecteur « V1 » par chaque élément d'un vecteur « V2 », et additionne le tout.

Par exemple, soit :

- Le vecteur V1

	0	1	2
V1	5	10	15

- Et le vecteur V2

	0	1	2	3
V2	3	2	12	15

La somme sera calculée comme suit

$S = (5*3 + 5*2 + 5*12 + 5*15) + (10*3 + 10*2 + 10*12 + 10*15) + (15*3 + 15*2 + 15*12 + 15*15)$
---

Analysons la situation :

- Le vecteur V1 sera parcouru une seule fois de 0 à 2. Donc on utilisera dans ce cas une boucle « Pour », avec l'indice « i » par exemple.
- Pour chaque élément du vecteur V1, le vecteur V2 sera parcouru de 0 à 3. Donc le vecteur V2 sera parcouru trois fois. Dans ce cas, on utilisera une autre boucle « Pour », avec l'indice « j », par exemple, qui sera imbriquée dans la première boucle « i ».

Après avoir rempli les deux vecteurs V1 et V2, on aura :

Pour i = 0

Pour j = 0	ajouter dans la somme	S = S + V1[0] * V2[0]
j = 1	//	S = S + V1[0] * V2[1]
j = 2	//	S = S + V1[0] * V2[2]
j = 3	//	S = S + V1[0] * V2[3]

Pour i = 1

Pour j = 0	ajouter dans la somme	S = S + V1[1] * V2[0]
j = 1	//	S = S + V1[1] * V2[1]
j = 2	//	S = S + V1[1] * V2[2]
j = 3	//	S = S + V1[1] * V2[3]

```

Pour i = 2
  Pour j = 0 ajouter dans la somme S = S + V1[2] * V2[0]
    j = 1 // S = S + V1[2] * V2[1]
    j = 2 // S = S + V1[2] * V2[2]
    j = 3 // S = S + V1[2] * V2[3]

```

Ceci nous donnera comme formule générale  $S = S + V1[i] * V2[j]$  qu'il faut placer dans la boucle imbriquée (j).

---

```

Algorithme valeur ;
  Var
    s : Réel ;
    n , i : Entier ;
    V1 , V2 : Tableau[1..50] de réel ;
  Début
    Répéter
      Ecrire ( ' Dimension de V1 <= 50 ? ' ) ;
      Lire ( n ) ;
      Jusqu'à ( n >= 1 ) et ( n <= 50 )

      Répéter
        Ecrire ( ' Dimension de V2 <= 50 ? ' ) ;
        Lire ( m ) ;
        Jusqu'à ( m >= 1 ) et ( m <= 50 )

      Pour i ← 0 à n - 1 Faire /*/ Remplir le vecteur V1
        Début
          Ecrire ( ' Donnez la valeur de V1 [ ' , i , ' ] = ' ) ;
          Lire ( V1 [ i ] ) ;
        Fin;

      Pour i ← 0 à m - 1 Faire /*/ Remplir le vecteur V2
        Début
          Ecrire ( ' Donnez la valeur de V2 [ ' , i , ' ] = ' ) ;
          Lire ( V2 [ i ] ) ;
        Fin;

      s ← 0
      Pour i ← 0 à n - 1 Faire
        Pour i ← 0 à m - 1 Faire /*/ La boucle imbriquée « j » pour la somme
          S ← S + V [ i ] * V [ j ] ;

      Ecrire ( ' Somme = ' , s ) ;
  Fin.

```

---

### 7.1.1.3. Recherche du maximum

**Exemple** : Ecrire un algorithme qui remplit un vecteur avec N valeurs saisies par l'utilisateur, puis affiche à la fin la valeur maximale ainsi que sa position dans le vecteur.

Analysons la situation :

- Remplir le vecteur V
- Initialiser le maximum au premier élément du vecteur
- Parcourir le vecteur V à partir du deuxième élément. Pour chaque élément du vecteur on le vérifie sa valeur avec celle du maximum:
  - o Si l'élément du vecteur est supérieur au maximum, cet élément deviendra le nouveau maximum et on enregistre sa position dans une variable.
- A la fin on affiche le maximum et sa position

---

```

Algorithme valeur ;
  Var
    max : Réel ;
    n, i, posmax : Entier ;
    V : Tableau[1..50] de réel ;
  Début

  Répéter
    Ecrire ( ' Dimension de V <= 50 ? ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 )

  Pour i ← 0 à n - 1 Faire
    Début
      Ecrire ( ' Donnez la valeur de V [ ' , i , ' ] = ' ) ;
      Lire ( V [ i ] ) ;
    Fin;

  max ← V [ 0 ] ;
  posmax ← 0 ;
  Pour i ← 1 à n - 1 Faire
    Si V [ i ] > max Alors
      Début
        max ← V [ i ] ;
        posmax ← i ;
      Fin;
  Ecrire ( ' Max = ' , max ) ;
  Ecrire ( ' Positon du Max = ' , posmax ) ;
Fin.

```

---

**Remarque :** Il est possible de déterminer le maximum, sans avoir à initialiser ce dernier au premier élément du vecteur. Il suffit de n'utiliser que sa position pour indiquer le maximum :

```

posmax ← 0
Pour i ← 1 à n - 1 Faire
  Si V[ i ] > V [ posmax ] Alors
    posmax ← i
Ecrire ( ' Max = ' , V [ posmax ] ) ;
Ecrire ( ' Positon du Max = ' , posmax ) ;

```

**Remarque :** N'utiliser que la position du maximum, permet d'éliminer une boucle. Le maximum sera déterminé au fur et à mesure du remplissage du vecteur.

---

```

Algorithme valeur ;
  Var
    n, i, posmax : Entier ;
    V : Tableau[1..50] de réel ;
Début

  Répéter
    Ecrire ( ' Dimension de V <= 50 ? ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 )

  posmax ← 0
  Pour i ← 0 à n - 1 Faire
    Début
      Ecrire ( ' Donnez la valeur de V [ ' , i , ' ] = ' ) ;
      Lire ( V [ i ] ) ;
      Si V [ i ] > V [ posmax ] Alors
        posmax ← i
    Fin;

  Ecrire ( ' Max = ' , V [ posmax ] ) ;
  Ecrire ( ' Positon du Max = ' , posmax ) ;
Fin.

```

---

#### 7.1.1.4. Rotation d'un vecteur

La rotation des éléments d'un vecteur est réalisée en exécutant un décalage de ses éléments.

	0	1	2	3	4	5	6	7
V	103	12	45	7	23	21	3	11

Après une rotation :

12	45	7	23	21	3	11	103
----	----	---	----	----	---	----	-----

Analysons la situation :

$$\left. \begin{array}{l}
 \text{Pour } i = 0 \longrightarrow V[0] = V[1] \\
 \text{Pour } i = 1 \longrightarrow V[1] = V[2] \\
 \text{Pour } i = 2 \longrightarrow V[2] = V[3] \\
 \text{Pour } i = 3 \longrightarrow V[3] = V[4] \\
 \dots\dots\dots \\
 \text{Pour } i = n - 2 \longrightarrow V[n - 2] = V[n - 1]
 \end{array} \right\} V[i] = V[i + 1]$$

Et à la fin, le dernier élément,  $V[n - 1]$ , recevra, normalement, la valeur qui était dans  $V[0]$ . Malheureusement, cette valeur a été remplacée par celle de  $V[1]$ . Pour garder en mémoire la valeur de  $V[0]$ , il faut :

- Enregistrer la valeur de  $V[0]$  dans une variable « x », avant de la remplacer par  $v[1]$  ; avant de commencer le décalage.
- A la fin du décalage, l'élément  $v[n - 1]$  recevra la valeur de x.

---

```

Algorithme valeur ;
  Var
    x : Réel ;
    n, i : Entier ;
    V : Tableau[1..50] de réel ;
  Début

  Répéter
    Ecrire ( ' Dimension de V <= 50 ? ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 )

  x ← V [ 0 ] ;
  Pour i ← 0 à n - 2 Faire
    V [ i ] ← V [ i + 1 ] ;

  V [ n - 1 ] ← x

  Fin.

```

---

Si on veut exécuter plusieurs décalages, la boucle de décalage sera imbriquée dans une boucle qui indique le nombre de décalage à faire.

---

```

Algorithme valeur ;
  Var
    x : Réel ;
    n , m , i : Entier ;
    V : Tableau[1..50] de réel ;
  Début

  Répéter
    Ecrire ( ' Dimension de V <= 50 ? ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 )

  Ecrire ( ' Donner le nombre de décalages à faire : ' ) ;
  Lire ( m ) ;

  Pour d ← 1 à m Faire
    Debut
      x ← V [ 0 ] ;
      Pour i ← 0 à n - 2 Faire
        V [ i ] ← V [ i + 1 ] ;

      V [ n - 1 ] ← x
    Fin ;

  Fin.

```

---

#### 7.1.1.5. Tri d'un vecteur

Il existe plusieurs techniques de tri d'un vecteur. Dans ce qui suit on va s'intéresser à la technique du « tri par sélection ». Il s'agit de trier un vecteur quelconque dans un ordre croissant (du plus petit au plus grand).



Soit le vecteur suivant :

	0	1	2	3	4	5	6	7
V	103	12	45	7	23	21	3	11

Comment fonctionne le tri par sélection ?

Pour  $i = 0$

- On cherche le minimum, dans la suite du vecteur, à partir de la position suivante. C'est-à-dire, à partir de « 1 ».
- Puis, on le compare avec l'élément «  $i$  ». S'il est plus petit, on exécute une permutation.

	0	1	2	3	4	5	6	7
V	3	12	45	7	23	21	103	11

Pour  $i = 1$

- On cherche le minimum, dans la suite du vecteur, à partir de la position suivante. C'est-à-dire, à partir de « 2 », car le premier minimum est déjà placé à la première position « 0 ».
- Puis, on le compare avec l'élément «  $i$  ». S'il est plus petit, on exécute une permutation.

	0	1	2	3	4	5	6	7
V	3	7	45	12	23	21	103	11

Pour  $i = 2$

- On cherche le minimum, dans la suite du vecteur, à partir de la position suivante. C'est-à-dire, à partir de « 3 », car le deuxième minimum est déjà placé à la deuxième position.
- Puis, on le compare avec l'élément «  $i$  ». S'il est plus petit, on exécute une permutation.

	0	1	2	3	4	5	6	7
V	3	7	11	12	23	21	103	45

Et ainsi de suite, jusqu'à la position  $N-1$ . Ceci nous amène à dire, qu'on doit utiliser deux boucles ; l'une imbriquée dans l'autre :

- Une boucle «  $i$  » allant de 0 à  $N-1$
- Pour chaque «  $i$  », on utilise une autre boucle imbriquée «  $j$  », allant de  $i + 1$  à  $N$  pour chercher le minimum.
- La permutation est exécutée à la sortie de la boucle «  $j$  ». C'est-à-dire dans la boucle «  $i$  ».

---

```

Algorithme valeur ;
  Var
    x : Réel ;
    n, i, j, Posmin : Entier ;
    V : Tableau[1..50] de réel ;
  Début

  Répéter
    Ecrire ( ' Dimension de V <= 50 ? ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n>=1 ) et ( n<=50 )

  Pour i ← 1 à n Faire
    Début
      Ecrire ( ' Donnez la valeur de V [ ' , i , ' ] = ' ) ;
      Lire ( V [ i ] ) ;
    Fin ;

  Pour i ← 0 à n - 1 Faire
    Debut
      posmin ← i
      Pour j ← i + 1 à n Faire
        Si V[j] < V [ posmin ] Alors
          posmin ← j

      Si V [ posmin ] < V [ i ]
        Debut
          x ← V [ posmin ]
          V [ posmin ] ← V [ j ]
          V [ j ] ← x
        Fin ;
    Fin ;
  Fin.

```

---

### 7.1.2. Les tableaux à deux dimensions (matrice)

Au contraire du vecteur, un tableau à deux dimensions permet de donner une information, suivant deux coordonnées (deux indices). Le premier indiquant la ligne et le second, la colonne.

Une matrice (tableau à deux dimensions) est une structure de données composée d'un ensemble de cases mémoires, où :

- La matrice doit avoir un nom unique. Par exemple, la matrice T.
- Une case mémoire représente un élément de la matrice.
- Tous les éléments de la matrice sont de même type.
- Le nom de chaque élément est composé du nom de la matrice avec deux indices (la ligne et la colonne). Ces derniers indiquent la position de l'élément dans la matrice. Par exemple, l'élément T [ 3 , 2 ].
- Les indices sont indiqués entre crochets.

	0	1	2	3
0	12	9	-4	6.5
1	105	0	35	2
2	33	-1	-1	14.5
3	-5	128	18	6
4	7.9	4.3	8.5	17.9
5	-0.5	0	0	1

$T[3, 2] = 18$

Il existe deux manières pour remplir une matrice :

1. Ligne par ligne
2. Colonne par colonne

### 7.1.2.1. Comment remplir une matrice

Deux boucles « Pour » permettent de remplir une matrice, ligne par ligne.

- La première boucle parcourt les lignes de la matrice.
- Pour chaque ligne, la deuxième boucle parcourt les colonnes de la matrice.

	0	1	2	3
0	.....	.....	.....	.....
1	.....	.....	.....	.....
2	.....	.....	.....	.....

**Convention :** Pour éviter les ambiguïtés qui peuvent être induites par les variables-compteurs « i et j » des deux boucles « Pour », par convention ;

- L'indice « i » sera utilisé pour indiquer la ligne
- L'indice « j » sera utilisé pour indiquer la colonne

Pour i = 0	Tous les éléments de la ligne seront remplis un par un	Pour j = 0 Remplir l'élément T[ 0 , 0 ] Pour j = 1 Remplir l'élément T[ 0 , 1 ] Pour j = 2 Remplir l'élément T[ 0 , 2 ] Pour j = 3 Remplir l'élément T[ 0 , 3 ]	}	<i>Boucle « j »</i>	}	<i>Boucle « i »</i>
Pour i = 1	Tous les éléments de la ligne seront remplis un par un	Pour j = 0 Remplir l'élément T[ 1 , 0 ] Pour j = 1 Remplir l'élément T[ 1 , 1 ] Pour j = 2 Remplir l'élément T[ 1 , 2 ] Pour j = 3 Remplir l'élément T[ 1 , 3 ]	}	<i>Boucle « j »</i>	}	<i>Boucle « i »</i>
Pour i = 2	Tous les éléments de la ligne seront remplis un par un	Pour j = 0 Remplir l'élément T[ 2 , 0 ] Pour j = 1 Remplir l'élément T[ 2 , 1 ] Pour j = 2 Remplir l'élément T[ 2 , 2 ] Pour j = 3 Remplir l'élément T[ 2 , 3 ]	}	<i>Boucle « j »</i>	}	<i>Boucle « i »</i>

Donc, la partie, de l'algorithme, qui permet de remplir une matrice aura la forme suivante

```

Pour i
  Pour j  saisir  T[ i , j ]

```

---

```

Algorithme valeur ;
Var
  n, m , i , j : Entier ;
  T : Tableau [ 1..50 , 1..50 ] de réel ;
Début

  Répéter
    Ecrire ( ' Dimension de V1 <= 50 ? ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n>=1 ) et ( n<=50 )

  Répéter
    Ecrire ( ' Dimension de V2 <= 50 ? ' ) ;
    Lire ( m ) ;
  Jusqu'à ( m>=1 ) et ( m<=50 )

  Pour i ← 1 à n Faire
    Pour j ← 1 à m Faire
      Début
        Ecrire ( ' Donnez la valeur de T [ ' , i , ' , ' , j , ' ] = ' ) ;
        Lire ( T[ i , j ] ) ;
      Fin ;

  Pour i ← 1 à n Faire  /**/ Pour afficher une matrice, utilisez deux boucles.
    Pour j ← 1 à m Faire
      Ecrire ( ' T [ ' , i , ' , ' , j , ' ] = ' , T[ i , j ] ) ;

```

Fin.

### 7.1.2.2. Trace d'une matrice

La trace d'une matrice carrée (nombre de lignes = nombre de colonnes) est la somme des éléments de la diagonale.

	0	1	2	3
0	12	9	-4	6.5
1	105	0	35	2
2	33	-1	-1	14.5
3	-5	128	18	6

$$S = T[0, 0] + T[1, 1] + T[2, 2] + T[3, 3] = 12 + 0 + -1 + 6 = 17$$

Donc, la somme est donnée par l'équation suivante :

$$S = \sum_{i=1}^N T[i, i]$$

Analysons la situation :

- 1- Donner les dimensions de la matrice. Comme la matrice est carrée, une seule dimension est suffisante puisque  $M = N$ .
- 2- Remplir la matrice
- 3- Parcourir la matrice diagonalement, en n'utilisant qu'un seul indice, pour calculer la trace.
- 4- Afficher la trace.

Ceci nous donnera l'algorithme suivant

---

```

Algorithme valeur ;
  Var
    S : Réel ;
    n, i, j : Entier ;
    T : Tableau [ 1..50 , 1..50 ] de réel ;
  Début

    Répéter
      Ecrire ( ' Nombre de lignes de T <= 50 ? ' ) ;
      Lire ( n ) ;
    Jusqu'à ( n >= 1 ) et ( n <= 50 ) ;

    Pour i ← 1 à n Faire
      Pour j ← 1 à n Faire
        Début
          Ecrire ( ' Donnez la valeur de T [ ' , i , ' , ' , j , ' ] = ' ) ;
          Lire ( T [ i , j ] ) ;
        Fin ;

    /**/ Une seule boucle est suffisante pour parcourir la diagonale
    s ← 0
    Pour i ← 1 à n Faire
      s ← s + T [ i , i ] ;

    Ecrire ( ' La Trace = ' , s ) ;
  Fin.

```

---

**Remarque :** Il est possible d'éliminer la dernière boucle en demandant à l'algorithme de faire la somme après remplissage d'une ligne de la matrice.

---

```

Algorithme valeur ;
  Var
    S : Réel ;
    n, i, j : Entier ;
    T : Tableau [ 1..50 , 1..50 ] de réel ;
  Début

  Répéter
    Ecrire ( ' Nombre de lignes de T <= 50 ? ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 ) ;

  s ← 0
  Pour i ← 1 à n Faire
    Début
      Pour j ← 1 à n Faire
        Début
          Ecrire ( ' Donnez la valeur de T [ ' , i , ' , ' , j , ' ] = ' ) ;
          Lire ( T [ i , j ] ) ;
        Fin ;
      s ← s + T [ i , i ] ;
    Fin ;

  Ecrire ( ' La Trace = ' , s ) ;
  Fin.

```

---

### 7.1.2.3. Matrice identité

Une matrice identité est une matrice où la valeur, de tous les éléments de la diagonale, est égale à (1) et le reste, zéro (0) :

	0	1	2	3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

Analysons la situation :

- 1- Donner les dimensions de la matrice. Comme la matrice est carrée, une seule dimension est suffisante puisque  $M = N$ .
- 2- Remplir la matrice
- 3- Parcourir la matrice ligne par ligne pour affecter un (1) aux éléments de la diagonale et zéro (0) aux autres éléments
- 4- Afficher la matrice.

Ceci nous donnera l'algorithme suivant :

---

```

Algorithme valeur ;
  Var
    n , i , j : Entier ;
    T : Tableau [ 1..50 , 1..50 ] de réel ;
Début

  Répéter
    Ecrire ( ' Nombre de lignes de T <= 50 ? ' ) ;
    Lire ( n ) ;
    Jusqu'à ( n >= 1 ) et ( n <= 50 ) ;

  Pour i ← 1 à n Faire
    Début
      Pour j ← 1 à n Faire
        T[ i , j ] ← 0 ;
      T[ i , i ] ← 1 ;
    Fin ;

  Pour i ← 1 à n Faire
    Pour j ← 1 à n Faire
      Ecrire ( ' T [ ' , i , ' , ' , j , ' ] = ' , T[ i , j ] ) ;
Fin.

```

---

#### 7.1.2.4. Transposé d'une matrice

Pour réaliser une matrice transposée, il suffit d'inter-changer entre les lignes et les colonnes :

$$\mathbf{A} = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 12 & 9 & -4 & 6.5 \\ 1 & 105 & 0 & 35 & 2 \\ 2 & 33 & -1 & -1 & 14.5 \\ 3 & -5 & 128 & 18 & 6 \\ 4 & 7.9 & 4.3 & 8.5 & 17.9 \\ 5 & -0.5 & 0 & 0 & 1 \end{array}$$

$$\mathbf{B} = {}^t\mathbf{A} = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 12 & 105 & 33 & -5 & 7.9 & -0.5 \\ 1 & 9 & 0 & -1 & 128 & 4.3 & 0 \\ 2 & -4 & 35 & -1 & 18 & 8.5 & 0 \\ 3 & 6.5 & 2 & 14.5 & 6 & 17.9 & 1 \end{array}$$

Analysons la situation :

- La première ligne devient la première colonne, la deuxième ligne devient la deuxième colonne, et ainsi de suite.
- Donc, au fur et à mesure que la matrice A est remplie, on remplit la matrice B en inversant les indices.

$$B[j, i] = A[i, j]$$

---

```
Algorithme valeur ;
  Var
    n , m , i , j : Entier ;
    T : Tableau [ 1..50 , 1..50 ] de réel ;
Début

  Répéter
    Ecrire ( ' Nombre de lignes de T <= 50 ? ' ) ;
    Lire ( n ) ;
  Jusqu'à ( n >= 1 ) et ( n <= 50 ) ;
  Répéter
    Ecrire ( ' Nombre de colonnes de T <= 50 ? ' ) ;
    Lire ( m ) ;
  Jusqu'à ( m >= 1 ) et ( m <= 50 ) ;

  Pour i ← 1 à n Faire
    Pour j ← 1 à m Faire
      Début
        Ecrire ( ' Donnez la valeur de A [ ' , i , ' , ' , j , ' ] = ' ) ;
        Lire ( A [ i , j ] ) ;
        B [ j , i ] ← A [ i , j ] ;
      Fin ;

  Pour i ← 1 à n Faire
    Pour j ← 1 à m Faire
      Ecrire ( ' B [ ' , i , ' , ' , j , ' ] = ' , B [ i , j ] ) ;

Fin.
```

---