1. INTRODUCTION

Le langage C a été mis au point au début des années 1970 et normalisé en 1988. C'est un langage de haut niveau qui génère un code très rapide grâce à un compilateur très performant.

2. STRUCTURE D'UN PROGRAMME C

Le langage **C** est un langage structuré, son programme de base est constitué de :

- Partie ent ête (Directive)
- Programme principal (Fonction principale)
- Partie traitement (Corps du programme = déclarations + instructions)

3. **EXEMPLE :** On essayera de faire la somme de deux nombres entiers lus.

Q1 : Ecriture l'algorithme correspondant ?

```
Algorithme somme
      Variables
          X, Y, Som : :entier ;
      DEBUT
        Ecrire ('entrer deux nombres entiers') ;
        Lire (x) ;
        Lire (y) ;
        Som← x+y ;
        Ecrire (Som) ;
       Fin.
```

Q2 : Ecriture son programme en C ?

```
      #include <stdio.h>  /* Partie entête ici C'est juste pour inclure des
bibliothèques */
       main()     /*Programma principale*/
      {  int X, Y, Som ;  /*Déclaration*/
      Printf(''entrer deux nombres entiers'') ; /* écriture d'un message */
      scanf("%i" , &X); /*Lecture de X*/
      scanf("%i" , &Y); /*Lecture de Y*/
      Som = X+Y ;  /*Calcul  la Somme*/
```

Remarque : La parenthèse ouvrante {joue le rôle de Début et la fermante} celui de Fin.

Explication de ce petit programme

- Ce programme principal `main()` utilise dans la partie traitement (corps du programme) deux différentes opérations `scanf()` et `printf()`, ce sont des opérations (fonctions) d'entrée/sortie (input/output),il utilise aussi une déclaration `int X,Y,Som` ; et une affectation `S = X+Y ;`
- Les opérations de déclaration et affectation sont reconnues par le compilateur par contre les opérations d'input/output ne le sont pas, donc il faut les inclure à l'aide de la directive `#include` qui précise au compilateur dans quel fichier se trouve la définition de ces fonctions.
- Que signifie `stdio.h` ?

  `Std : Standard`
  `io : input/output`
  `.h : header (entête)`
- L'inclusion qui est une opération appartenant aux instructions du pré processeur se fait comme suit : `#include <stdio.h>`

- Que signifie %i et & dans les opérations d'input/output `scanf()` et `printf()` ?
  `%i` : veut dire que c'est le format entier (i comme integer),
  `%i` est équivalent à `%d`
  `&` : cet opérateur doit précéder toute variable de type (`entier, réel, caractère …`).

REMARQUES

- Attention un mot écrit en minuscule est différent de celui qui est écrit en majuscule. Si vous écrivez `main()` et `Main()` ce n'est pas la même chose
- L'écriture du programme C doit être en minuscule !
- Le langage C permet la déclaration des variables à n'importe quel endroit dans le programme, il n'a pas une zone précise appelée zone déclarative. Il est permissif.
- Par contre la déclaration des constantes est dans l'entête (partie du pré compilateur), à l'aide de `#define`.
- En C, toute déclaration et/ou toute instruction se termine par un point- virgule ; sauf les instructions destinées au pré processeur comme par exemple `#include` …
- A chaque fois qu'on utilise une fonction standard, on doit inclure son fichier entête !

3. TYPES DE DONNEES

- Pour l'instant, on va étudier seulement trois (03) types de données (Variables) : les entiers, les réels et les caractères.
- **`int`** pour les entiers (`integer`)
- **`float`** pour les réels (flottant ou virgule flottante)
- **`char`** pour les caractères

4 .COMMENT DECLARER UNE VARIABLE ?

Algorithme	**Langage C**
Déclaration	Pas d'équivalent
Variables	Pas d'équivalent
A : entier ;	int A ;
C, D : réel ;	float C, D ;
K : caractère ;	char K ;

Aussi la déclaration d'une variable en C, se fait dans le corps du programme et non dans une zone déclarative comme dans un algorithme.

5. COMMENT DECLARER UNE CONSTANTE ?

Elles sont déclarées différemment des variables, on doit les définir dans la partie entête du programme C à l'aide de la directive #define et de la façon suivante :

`#define nom_de_la_constante valeur_de_la_constante`

Algorithme	Langage C
Constante	Pas d'équivalent
PI = 3.14 ;	#define PI 3.14
Déclaration	Pas d'équivalent

6. FICHIERS ENTETES

- On a déjà vu ce genre de déclaration quand on a écrit notre premier programme en langage C qui fait la somme de deux entiers X et Y dans Som, on a utilisé :

#include <stdio.h>

- D'une manière générale cela s'écrie comme suit :

#include <nom_du_fichier_entête>

- **Attention** : On n'inclut pas seulement les fichiers input /output mais il y en a d'autres qu'on verra par la suite !

7. LES OPERATIONS ELEMENTAIRES DU LANGAGE C

- **L'affectation :** c'est l'attribution d'une valeur ou d'un résultat d'une opération à une variable de même type.

Algorithme	Langage C
Début ……. ……. X ← 5 ; Y ← 18 ; SOM ← X+Y ; …. ….. FIN	X = 5 ; Y = 18 ; SOM = X+Y ; …… ……

- **La lecture (input)** : C'est l'attribution d'une valeur choisie par l'utilisateur à une variable d'entrée. La valeur choisie doit être de même type que la variable d'entrée. Cette opération se fait à l'aide de la fonction scanf() ; En générale cette fonction s'écrit sous la forme suivante :

scanf('' indicateur_de_type '', &nom_de_la_variable) ;

L'opérateur & précède toute variable de type scalaire (entier, réel ou caractère).

Algorithme	Langage C
Lire(X) ;	scanf('' %d '', &X) ;

NB : la variable x ici est de type entier c pour ça %d

- **L'écriture (output)** : C'est l'affichage (l'impression) du contenu d'une variable, d'une constante ou d'un simple message. Cette opération se fait à l'aide de la fonction `printf()` ; En générale cette fonction s'écrit sous la forme suivante : `pintf(''` indicateur_de_type `'',` nom_de_la_variable) ; ou bien `printf('' message '');`
-

Algorithme
Ecrire ('Bonjour') ; Ecrire(S) ;

Langage C
printf('' Bonjour '') ; printf('' %i '', Som);

`%i` est un indicateur de format (i comme **`integer`** c'est-à-dire entier). Vous aurez la liste de tous les indicateurs de format par la suite. Vous avez remarqué que le message doit être mis entre deux (02) double côtes `''message''`.

- **`L'incrémentation/la décrémentation`** : C'est une affectation un peu particulière, car on trouve le nom de la variable de part et d'autre du symbole d'affectation (← en algorithme et = en langage C).
  - L'incrémentation permet d'augmenter d'une valeur une variable par contre la décrémentation permet de diminuer d'une valeur une variable. Cette valeur est appelée le pas et le pas peut être positif ou négatif.

Algorithme	**Langage C**
Début	
X ← X + 1 ;	X = X + 1 ;
Y ← Y − 1 ;	Y = Y − 1 ;
A ← A + 2 ;	A = A + 2 ;
B ← B − 5 ;	B = B − 5 ;
Fin	

Attention : Il y a une écriture en langage C que je vous conseille d'éviter quoiqu'elle soit permise. `X = X + 1 ;` peut s'écrire `X + + ;` `Y = Y − 1` ; peut s'écrire `Y − − ;`

8. LES OPERATIONS ARITHMETIQUES

- L'addition +
- La soustraction –
- La multiplication *
- La division /
- Le modulo % c'est le reste de la division Euclidienne.

Algorithme	**Langage C**
Début	
Delta ← b² - 4ac ;	Delta = b*b − 4*a*c ;
Q ← X/2;	Q = X / 2;
R ← X − 2Q;	R = X − 2*Q;
Fin	

9. LES OPERATEURS LOGIQUES

Une condition peut être elle-même composée de plusieurs conditions reliées entre elles par des opérateurs logiques tels que et, ou, non. En langage C :

- Le et est représenté par &&
- Le ou est représenté par ||
- Le non est représenté par !

Algorithme	Langage C
A et B	A && B
X OU Y	X || Y
Non Z	! Z

10. LES OPERATEURS DE COMPARAISON :

On les appelle aussi signes de test :

==	teste si les deux informations sont égales.
<	teste si l'information à gauche est inférieure à celle de droite.
>	teste si l'information à gauche est supérieure à celle de droite.
<=	teste si l'information à gauche est inférieure ou égale à celle de droite.
>=	teste si l'information à gauche est supérieure ou égale à celle de droite.
!=	teste si les deux informations sont différentes.

Exemple : On va écrire deux instructions presque identiques pour ordonner la machine d'afficher le contenu d'une variable S de type entier.

- `printf(''%i'', S) ;`
- `printf(''%i\n'', S) ;`

La première instruction ordonne la machine à écrire la variable S et de rester sur la même ligne.

La seconde instruction ordonne la machine à écrire la variable S et d'aller à la ligne suivante, c'est ce qu'on appelle un retour chariot et cela grâce à \n.

NB : l'intérêt de cet exemple à cet endroit est de comprendre l'option \n

11. LES COMMENTAIRES

Pour rendre notre programme plus lisible, on doit le commenter c'est-à-dire écrire des commentaires pour expliquer ce qu'on est en train de faire. Ces commentaires seront ignorés par le compilateur C.

- Tout commentaire est mis entre /* et */.
- Exemple : /* ceci est un commentaire* /

12. LES STRUCTURES DE CONTROLE

On distingue deux types de structures de contrôle:

- **les conditions** :elles permettent d'écrire dans le programme des règles comme « Si ceci arrive... alors fais cela ...» ;
- **Les boucles**: elles permettent de répéter plusieurs fois une série d'instructions.
- On les appelle aussi les primitives, elles servent à contrôler le déroulement d'un traitement.

Les instructions : peuvent s'exécuter de différentes manières :

12.a Le traitement séquentiel : C'est une suite d'instructions qui s'exécutent l'une à la suite de l'autre sans aucune contrainte.

Algorithme

```
Début
   instruction1 ;
   instruction2 ;
  …
  Instruction n ;
Fin
```

Langage C

```
Instruction1 ;
Instruction2 ;
…
 instruction n;
```

12.b Le traitement conditionnel : Dans ce cas l'instruction est exécutée que si la condition est vérifiée. On dit que l'instruction est sous condition.

```
Algorithme

Début

   Si condition(s) alors

      Action ;

Finsi

fin
```

```
Langage C

 if condition(s)

   action ;
```

NB : on remarque que le mot alors en algorithme n'a pas d'équivalent en langage C de même pour le vocable finsi.

12.c Le traitement alternatif (si …sinon…..): Comme son nom l'indique, si ce n'est pas l'une ça sera l'autre ; c'est-à-dire si la condition est vérifiée l'action 1 est exécutée si ce n'est pas le cas l'action 2 sera exécutée et en aucun cas les actions 1 et 2 ne seront exécutées en même temps !

Algorithme

```
Début
…
 Si condition(s) alors
     Inst1 ;
     Inst2 ;
     Inst3 :
     ….
     .. ;.
     Instn :

Sinon
     Inst1 ;
     Inst2 ;
     Inst3 :
     ….
     .. ;.
     Instn :
finsi
     …
fin
```

Langage C

```
       …
 if condition(s)
    Inst1 ;
    Inst2 ;
    Inst3 :
    ….
    .. ;.
    Instn
else
    Inst1 ;
    Inst2 ;
    Inst3 :
    ….
    .. ;.
    Instn :
```

NB : si on a plusieurs actions que ce soit dans le alors ou/et dans le sinon, on utilisera des blocs. Début et fin en algorithme et les parenthèses ouvrantes et fermantes pour le langage C.

12 .d L'instruction selon le cas : Elle indique le choix multiple, et au lieu d'utiliser une cascade de si alors sinon, on préfère cette primitive pour nous faciliter l'écriture de l'algorithme ou le programme et le rendre plus lisible.(Voir cours sur algorithme).

Algorithme	**Langage C**
Début … … **Selon le cas** variable Cas1 : action1(s) ; Cas2 : action2(s) ; …….. Casn : actionn(s) ; Finselon ….. FIN	**switch** variable_de_choix case choix1 :action ;breack ; case Choix2 :action2 ; ;breack ; … case choix n :action n ;breack ;

Il existe un autre format d'écriture qui utilise le mot « autre » en algorithme qui sera traduit par « default » en langage C.

Algorithme	**Langage C**
Début Selon le cas variable_de choix Choix1 : action1 ; Choix2 :action2 ; choix n :action n ; … sinon : action n+1 ; Finselon … Fin	case switch variable_de_choix case choix1 :action1 ;breack ; case choix2 :action2 ;breack ; … Case Choix n :action n ; breack ; default : action n+1 ; breack ;

NB : En langage C la primitive switch utilise un bloc { } qui englobe tous les cas possibles et à la fin de chaque action il y a l'instruction breack pour casser la séquence. La finselon de l'algorithme n'a pas d'équivalent en C. **Exemple sur le SWITCH :**

```
#include <stdio.h>
main(s)
{   int M ;
printf(''Entrer une valeur entière'') ;
  scanf(''%d'', &M) ;
 switch (M)
{        case 1 : printf('' c'est Dimanche '') ;breack ;
             case 2 : printf(''c'est lundi'') ;breack ;
             case 3 : printf(''c'est Mardi'') ;breack ;
             case 4 : printf(''c'est mercredi'') ;breack ;
             case 5 : printf(''c'est jeudi'') ;breack ;
             case 6 : printf(''c'est vendredi'') ;breack ;
             case 7 : printf(''c'est Samedi'') ;breack ;
  default : printf(''Votre nombre ne correspond à aucun jour'');
 }  }
```

12.d Le traitement répetitif ,itératif ou boucles : On a trois primitives (instructions) pour contrôler un traitement répétitif.

A. Pour i allant de vi à vf (pas =p) faire
B. tant que (condition(s)) faire
C. répéter….jusqu'à (condition(s))

A. La boucle `Tant que faire` : Son principe est très simple :

- tant que la condition ou les conditions est/sont vérifiée(s), on exécute l'instruction ou les instructions qui est/sont à l'intérieur de la boucle.
- Une fois la condition ou les conditions n'est/sont plus vérifiée(s) on sort de la boucle. Si on a plus d'une action à l'intérieur de la boucle, on utilisera un bloc.

Algorithme

```
Début

Tantque   (condition(s)) Faire

    Action1 ;
    Action2 ;
    …….
    Action
Fintantque
 Fin
```

Langage C

```
while condition(s)
  action1 ;
  action2 ;
   …….
   Action ;
```

- On remarque que le mot clé Faire n'a pas d'équivalent en C, du même pour la fintantque.

Exemple : Afficher les dix (10) premiers nombres entiers positifs et chaque nombre est écrit sur une ligne.

```
#include <stdio.h>
main()
{  Int co ; /*déclaration du compteur*/
 co = 1 ;  /*initialisation du compteur*/
 while (co <= 10)   /*condition d'arrêt*/
    {   printf(''%d\n'', co) ;  /*affichage du compteur*/
        co = co+1 ;   /*incrémentation du compteur*/
}
```

B. La boucle `Pour` : Contrairement à la boucle tant que où devant le `while` on ne trouve que la/les condition(s) ; devant le `pour` (`for` en C) on trouve l'initialisation d'une variable, la condition d'arrêt et le pas (incrémentation ou décrémentation).

ALGORITHME

```
Pour V allant de Vi à Vf (pas = P) faire
    instruction(s) ;
Finpour
```

Langage C

```
for (initialisation ; condition ; pas ;)
action(s) ;
```

- Où V est le nom d'une variable déclarée.
- Vi est la valeur initiale (valeur de départ).
- Vf est la valeur finale (valeur d'arrive).

- P est le pas (incrémentation on dit que le pas est positif ou décrémentation on dit que le pas est négatif).
- Le même principe que la boucle tant que, si on a plus d'une action, un bloc est nécessaire.

Reprenons l'exemple traité avec la boucle while et écrivons le en utilisant la boucle for.

```
#include <stdio.h>
main()
 {
 int co ;
 for (co = 1 ; co <= 10 ; co = co+1 ;)
        printf(''%d\n'', co);
}
```

Faisons un parallèle entre les deux écritures **(while et for).**

```
       Boucle while
#include <stdio.h>
main()
{     int co;
Co = 1;
while (co <= 10)
{
   printf(''%d\n'', co);
  co=co+1;
    }  }
```

```
           Boucle for
#include <stdio.h>
main()
{ int co;
for (co=1; co<=10; co=co+1;)
printf(''%d\n'', co);

}
```

- On Remarque qu'avec la boucle for on a gagné deux (02) lignes de code: l'initialisation (`co=1;`) et l'incrémentation (`co=co+1;`) et même un bloc { }.
- Par conséquent si on a la possibilité d'utiliser la boucle for n'hésiter pas un seul instant ! à condition qu'on ait une et une seule variable V, une valeur initiale Vi, une valeur de fin Vf et le pas P (positif : incrémentation ou négatif : décrémentation).

C .La boucle tant que condition faire: Cette boucle est un peu spéciale, on traite l'action puis on vérifie la condition. C'est pour cela que cette boucle n'est utilisée que si on est assuré qu'elle va s'exécuter au moins une fois. Bizarrement, elle ressemble à la boucle (répéter jusqu'à) qu'on a étudié en algorithmique. Sa syntaxe est la suivante :

```
     Algorithme
Faire
Action (s);
Tantque
condition(s);
```

```
     Langage C
do
action(s) ;
while (condition(s)) ;
```

Nb *: Traitons le même exemple du comptage avec la boucle (do while).*

```
#include <stdio.h>
main()
{   int co ;  co = 1 ;
   do
          {    printf(‘’ %d\n’’, co) ;
              co = co+1;          }
   while (co <= 10);    }
```

Remarque:

- la priorité est donnée à la boucle (`for`) puis la boucle (`while`) et en dernier la boucle (`do while`) car son problème est qu'il faut être assuré qu'elle va s'exécuter au moins une fois !
- La fonction printf est une fonction d'impression formatée, ce qui signifie que les données sont converties selon le format particulier choisi. Ci-dessous le tableau des formats d'impression pour la fonction printf

Format	Conversion	en Ecriture
%s	char*	chaine de caractères
%d	int	décimale signée
%u	unsigned int	décimale non signée
%o	unsigned int	octale non signée
%c	unsigned char	Caractère
%e	double décimale	notation exponentielle
%x	unsigned int	hexadécimale non signée

- Les opérateurs d'incrémentation ++ et de décrémentation -- s'utilisent aussi bien en suffixe (i++) qu'en préfixe (++i). Dans les deux cas la variable i sera incrémentée, toutefois dans la notation suffixe la valeur retournée sera l'ancienne valeur de i alors que dans la notation préfixe se sera la nouvelle
- Le langage C ne dispose que de la notation / pour désigner à la fois la division entière et la division entre flottants. Si les deux opérandes sont de type entier, l'opérateur / produira une division entière (quotient de la division). Par contre, il délivrera une valeur flottante dès que l'un des opérandes est un flottant.
- L'opérateur % calcule le reste de la division Euclidienne (modulo). Il ne s'applique qu'à des opérandes de type entier.
- Le tableau suivant classe les opérateurs par ordres de priorité décroissants, les opérateurs sur une même ligne ayant une priorité égale (on évalue alors de gauche à droite).

Priorité 1 -	( ) [ ]
Priorité 2 (unaire)	-- ++ !
Priorité 3	* / %
Priorité 4 (binaire)	+ -
Priorité 5	< <= > >=
Priorité 6	== !=
Priorité 7	&&
Priorité 8	||
Priorité 9	%= /= *= -= += =

CHAPITRE : LES VARIABLES INDICEES

Partie A : Les tableaux à une dimension (Vecteurs)

I. DEFINITION :

Les tableaux représentent une structure de données permettant de stocker des données de même nature. On les représente souvent par un ensemble de cases contenant chacune une valeur.

Les tableaux peuvent avoir plusieurs dimensions dont les plus répandus sont les tableaux à une dimension (vecteurs) et les tableaux à deux dimensions (matrices).

	1ère case	***2ème case***	***3ème case***		***100ème case***
Moyenne	15 .33	13.75	9.00	…	12 .50

Un vecteur peut être vu comme une liste d'éléments arrangés dans des cases. Chaque case est référencée par un indice permettant d'identifier sa valeur.

Exemple : vecteur *Moyenne*

- Un tableau possède un nom (ici *Moyenne*) et un nombre d'éléments (de cases) qui représente sa taille (ici 100).
- Tous les éléments d'un tableau ont le même type (ici les éléments sont des réels).
- Pour désigner un élément, on indique le nom du vecteur suivi par son indice (son numéro) entre crochets : Moyenne [2] représente le 2ème élément du tableau Moyenne et vaut 13.75.

II. DECLARATION DES TABLEAUX A UNE DIMENSION

La syntaxe de la déclaration d'une variable tableau est la suivante:

<Identificateur> [indiceMin.. indiceMax] : tableau de <type>

Exemple :
Moyenne [1..100] : tableau de réels
Tab [1..20] : tableau d'entiers
ElemT [1..200] : tableau de booléens

III. MANIPULATION DES VECTEURS

Les tableaux se manipulent à travers leurs éléments représentés par le nom du tableau suivi d'un indice entre crochets.

L'**indice** d'un élément peut être:

- directement une valeur ex: Moyenne [10]
- une variable ex: Moyenne [i]
- une expression entière ex: Moyenne [2*i+k] avec i et k de type entier

Quelque soit sa forme, **la valeur de l'indice** doit être :

- entière
- comprise entre les valeurs minimales et maximales déterminées à la déclaration du tableau.

Par exemple, avec le tableau Tab [1..50], il est impossible d'écrire Tab [0] et Tab [51]. Ces expressions font référence à des éléments qui n'existent pas.

IV. LECTURE D'UN VECTEUR

Cette opération de lecture consiste en le remplissage des différentes cases qui constituent le tableau.
Exemple :

Ecrire un algorithme qui permet de lire un tableau de 20 éléments entiers.

```
Algorithme LectureTableau
Variables :
   Tab [1..20] : tableau d'entiers
   i : entier
Début
   Pour i allant de 1 à 20 Faire
      Lire (Tab[i])
   Fin pour
Fin
```

V. EDITION D'UN VECTEUR

L'édition d'un tableau consiste en l'affichage de ses éléments c'est-à-dire le contenu de ses différentes cases.

Exemple :Ecrire un algorithme qui permet d'afficher un tableau de 20 éléments entiers.

```
Algorithme LectureTableau
Variables :
   Tab [1..20] : tableau d'entiers
   i : entier
Début
   Pour i allant de 1 à 20 Faire
      Ecrire (Tab[i])
   Fin pour
Fin
```

Exercices :

Exercice1 : Ecrire un algorithme qui calcule la moyenne des valeurs d'un tableau de 100 éléments entiers.

Exercice2 : Ecrire un algorithme qui calcule le nombre d'entiers positifs et le nombre d'entiers négatifs d'un tableau de 200 éléments.

Exercice3 : Ecrire un algorithme qui permet de trouver la valeur maximale d'un tableau de 20 éléments réels.

Exercice4 : Ecrire un algorithme qui permet de trouver si une valeur « X » appartient à un tableau de 100 éléments entiers.

SOLUTION

```
Algorithme  exercice1
Variables :
   Tab [1..100] : tableau d'entiers
   Som, i : entier
   Moy : réel
Début
        Pour i allant de 1 à 100 Faire
        |       Lire (Tab[i])
        Fin pour
        Som ← 0
        Pour i allant de 1 à 100 Faire
        |       Som ← Som+ Tab[i]
        Fin pour
        Moy ← Som/100
        Ecrire ('Moyenne=', Moy)
  Fin
```

```
Algorithme exercice2
Variables :
   Tab [1..200] : tableau d'entiers
   Pos, Neg, i : entier
  Début
        Pour i allant de 1 à 200 Faire
        |       Lire (Tab[i])
        Fin pour
         Pos ← 0
         Neg ← 0
         Pour i allant de 1 à 200 Faire
        |       Si (Tab[i] < 0) Alors
        |       |       Neg ← Neg+1
        |       |     Sinon
        |       |       Pos ← Pos+1
        |          Fin si
        Fin pour
        Ecrire ('Le nombre de valeurs positives=', Pos)
        Ecrire ('Le nombre de valeurs négatives=', Neg)
  Fin
```

```
Algorithme exercice3
Variables :
   Tab [1..20] : tableau de réels
   i : entier
   max : réel
   Début
        Pour i allant de 1 à 20 Faire
        |       Lire (Tab[i])
        Fin pour
          max ← Tab [1]
         Pour i allant de 2 à 20 Faire
        |       Si (Tab[i] > max) Alors
        |       |       max ← Tab [i]
        |          Fin si
        Fin pour
        Ecrire ('Valeur max=', max)
   Fin
```

```
Algorithme exercice4
Constantes :
    N = 100
Variables :
    Tab [1..N] : tableau d'entiers
    X, i : entier
    Trouv : booléen
Début
    1  Lire (X)
       Pour i allant de 1 à 200 Faire
           Lire (Tab[i])
       Fin pour
    2  i ← 1
       Trouv ← faux
       Tant que ((i <= N) et (non Trouv)) Faire
           Si (Tab[i] = X) Alors
        3      Trouv ← vrai
           Fin si
        4  i ← i+1
       Fin TQ
       Si (Trouv) Alors
        5      Ecrire (X, 'appartient au tableau')
       Sinon
        6      Ecrire (X, 'n-appartient pas au tableau')
       Fin si
Fin
```

- *Tracé d'exécution (N=5):*

Etape	N	X	I	Trouv	Tab[1]	Tab[2]	Tab[3]	Tab[4]	Tab[5]	Ecran
1	5	9	/	/	2	7	9	12	3	/
2	5	9	1	Faux	2	7	9	12	3	/
4	5	9	2	Faux	2	7	9	12	3	/
4	5	9	3	Faux	2	7	9	12	3	/
3	5	9	3	Vrai	2	7	9	12	3	/
4	5	9	4	Vrai	2	7	9	12	3	/
5	5	9	4	Vrai	2	7	9	12	3	9 appartient au tableau

Etape	N	X	I	Trouv	Ta [1]	Tab[2]	Tab[3]	Tab[4]	Tab[5]	Ecran
1	5	8	/	/	2	7	12	9	3	/
2	5	8	1	faux	2	7	12	9	3	/
4	5	8	2	faux	2	7	12	9	3	/
4	5	8	3	faux	2	7	12	9	3	/
4	5	8	4	faux	2	7	12	9	3	/
4	5	8	5	faux	2	7	12	9	3	/
4	5	8	6	faux	2	7	12	9	3	/
6	5	8	6	faux	2	7	12	9	3	8 n-appartient pas au tableau

Partie B : Les tableaux à deux dimensions (Matrices)

I. DEFINITION :

Les matrices sont des tableaux à deux dimensions, c'est-à-dire une structure de données permettant de stocker des informations, de même nature, référencées par leurs numéros de ligne et de colonne.

II. CARACTERISTIQUES DES MATRICES

Les matrices présentent les mêmes caractéristiques que les vecteurs. Les éléments d'une matrice sont arrangés dans des cases référencées par deux indices servant d'identifier sa valeur.

Exemple : matrice *MAT*

		1ère colonne	***2ème colonne***	***3ème colonne***	***…***	***100ème colonne***
MAT	***1ère ligne***	5	2	0	…	7
	2ème ligne	8	23	40	…	12
	…	…	…	…	…	…
	200éme ligne	3	15	32	…	4

- Une matrice possède un nom (ici *MAT*) et un nombre d'éléments qui représente sa taille (ici 100*200).
- Tous les éléments d'une matrice ont le même type (ici les éléments sont des entiers).
- Pour désigner un élément, on indique le nom de la matrice suivi par ses indices (indice ligne et indice colonne) entre crochets : MAT [2,3] représente la case située à l'intersection de la ligne « 2 » et la colonne « 3 » et vaut 23.

III. DECLARATION DES MATRICES

La syntaxe de la déclaration d'une variable matrice est la suivante:

<Identificateur> [LiMin.. LiMax, ColMin..ColMax] : Matrice de <type>

Exemple :
MAT [1..100,1..200] : matrice d'entiers
Somme [1..20,1..30] : matrice de réels

IV. MANIPULATION DES MATRICES

Les éléments d'une matrice se manipulent en spécifiant le nom de la matrice suivi des deux indices ligne et colonne entre crochets.

Similairement aux vecteurs, Les deux indices ligne et colonne peuvent avoir plusieurs formes : valeur, variable ou expression. Quelque que soit cette forme, la valeur des deux indices doivent être de type entier et être dans l'intervalle des indices permis.

V. LECTURE D'UNE MATRICE

Cette opération de lecture consiste en le remplissage des différentes cases qui constituent la matrice.

Exemple **:** Ecrire un algorithme qui permet de lire une matrice de 20 lignes et 30 colonnes d'éléments entiers

```
Algorithme LectureMatrice
Variables :
   Mat [1..20, 1..30] : matrice d'entiers
   i, j : entier
Début
        Pour i allant de 1 à 20 Faire
            Pour j allant de 1 à 30 Faire
                Lire (Mat [i, j])
            Fin pour
        Fin pour
  Fin
```

.

VI. EDITION D'UNE MATRICE

L'édition d'une matrice consiste en l'affichage de ses éléments c'est-à-dire le contenu de ses différentes cases. ***Exemple*** **:** Ecrire un algorithme qui permet d'afficher une matrice de 20 lignes et 30 colonnes d'éléments entiers.

```
Algorithme éditionMatrice
Variables :
   Mat [1..20, 1..30] : matrice d'entiers
   i, j : entier
Début
        Pour i allant de 1 à 20 Faire
            Pour j allant de 1 à 30 Faire
                écrire (Mat [i, j])
            Fin pour
        Fin pour
  Fin
```

Exercices :

Exercice1 : Ecrire un algorithme qui lit deux matrices a et b et calcule leur somme.

Exercice2 : Ecrire un algorithme qui permet de trouver la valeur maximale pour chaque ligne d'une matrice de 20 lignes et 20 colonnes d'éléments réels.

Exercice3 Ecrire un algorithme permettant de rechercher le plus grand élément d'une matrice et de retourner ses numéros de ligne et de colonne.

1. INTRODUCTION

Supposons qu'on ait une dizaine de variables de même type (entier par exemple) qu'on appellera A_1, A_2, A_3, A_4, … , A_{10}. Ces dix variables simples occupent chacune un emplacement physique en mémoire, après leur déclaration. Il y a une autre possibilité : Les rassembler toutes dans une seule structure appelée « tableau » dans dix emplacements contigus mais avec un seul nom (tableau). Chaque élément de ce tableau est repéré par un indice (son rang dans le tableau).

2. DECLARATION

En algorithmique, la déclaration d'un tableau se fait à la clause variable par la syntaxe suivante :

identificateur-tableau : tableau [1..N] de type ;

Comme vous pouvez le remarquer, pour définir un tableau on doit donner son nom, son type et sa taille (nombre de cases). ***En langage C***, La définition d'un tableau à une dimension (vecteur) suit la syntaxe suivante :

Syntaxe :

Type *Identificateur* **[***Taille constante***] ;**

- La ***Taille*** du tableau est le ***nombre de ses éléments***. Elle ne peut être une variable. ***Elle doit être une constante*** définie avant ou au moment de la déclaration.

C'est-à-dire Type-case nom-tableau [nombre-case] ;

REMARQUE :

- Un tableau est un ensemble de données qui sont toutes de même type.
- Le tableau a un nom.
- Les données possèdent un identificateur unique : le nom du tableau.
- La taille d'un tableau correspond au nombre de cases.
- Les données se différentient par leur numéro d'indice.
- le nombre de cases est entre crochets et non entre parenthèses.

Exemples :

Déclaration de tableaux

```
#include <stdio.h>
#define taille1  5      /*taille1: constante de valeur 5*/
#define taille2  3      /*taille2: constante de valeur 3*/
main( )
{
int a [taille1];        /*a: tableau de 5 entiers*/
a[0]=15;                /*la première case du tableau a reçoit 15*/
char b [taille2];       /*b: tableau de 3 caractères*/
b[0]='x';               /*la première case du tableau b reçoit la lettre x*/
b[1]='y';               /*la deuxième case du tableau b reçoit la lettre y*/
float c [10];           /*c: tableau de 10 réels*/
scanf("%f", &c[0]);     /*lit un réel et l'affecte à la première case de c*/
}
```

ATTENTION:

- En C, Les indices d'un tableau sont des entiers commençant **à 0.**
- L'affectation élément par élément d'un tableau B à un autre tableau A (A[i]=B[i]) réalise une copie de B dans A. Une affectation "brutale" de B à A (A=B) n'est pas possible .

3. INITIALISATION

Syntaxe :
Type *Identificateur* **[***Taille constante***] = {***Valeur1***,** *Valeur2*,…,*Valeurn***};**

Initialisation d'un tableau à plusieurs dimensions

```
#define taille1 3  /*taille1: constante de valeur 3*/
#define taille2 2  /*taille2: constante de valeur 2*/
main( )
{
int M [taille1][taille2]={{0,1},{2,3},{4,5}};          /*initialisation ligne par ligne*/
}                                                      /*M[0]={0,1}, M[1]={2,3} et M[2]={4,5}*/
```

4. LECTURE ET AFFICHAGE

Lecture et affichage de tableaux

```
#include <stdio.h>
#define taille 20                  /*taille: constante de valeur 20*/
main( )
{
int i, t [taille];                 /*t: tableau de 20 entiers*/
for(i=0;i<taille;i++)              /*lit les 20 entiers élément par élément*/
      scanf ("%d",&t[i]);
for(i=0;i<taille;i++)              /*affiche les 20 entiers élément par élément*/
      printf ("%d\n",t[i]);
}
```

5. L'AFFECTATION

L'affectation de valeurs aux éléments d'un tableau se fait également individuellement (comme pour la lecture et l'affichage).

Affectation de valeurs à un tableau

```
#include <stdio.h>
#define taille 20              /*taille: constante de valeur 20*/
main( )
{
int i, t [taille];             /*t: tableau de 20 entiers*/
for(i=0;i<taille;i++)          /*affecte i à chaque élément d'indice i*/
      t[i]=i;
}
```

6. DECLARATION D'UN TABLEAU A DEUX DIMENSIONS

Syntaxe :

Type *Identificateur* **[***Taille1***]** **[***Taille2***]** … **[***Taille n***] ;**

- Taillei est la taille de la dimension i. Elle doit être une constante définie avant ou au moment de la déclaration.
- ***Un élément*** d'un tableau t à n dimensions est repéré par ses indices, sous forme ***t[i1][i2]…[in]***.

A deux dimensions :

<NomTableau>[N (nbr de lignes),M (nbr de colonnes)] : type

Déclaration et lecture d'un tableau à deux dimensions

```
/*Ce programme lit le nombre de buts marqués par chacun des 11 joueurs de 8 équipes*/

      #include <stdio.h>
      #define   taille1   8          /*taille1: constante de valeur 8*/
      #define   taille2   11         /*taille2: constante de valeur 11*/

      main( )

      {
      int t [taille1][taille2];      /*t : matrice de 8 lignes, 11 colonnes*/
      int i, j;

      for(i=0;i<taille1;i++)         /*lit les éléments de t ligne par ligne*/
            for (j=0; j<taille2;j++)
                  scanf ("%d",&t[i][j]);
      }
```

EXERCICES

I.OBJECTIF :

Le but principal de ces exercices est :

- Maitriser l'utilisation des structures de contrôles répétitives (Pour, Tant que et répéter).
- Se familiariser avec les tableaux et les matrices.

II. EXERCICES :

❖ **Exercice 01 :** Montrer le tracé d'exécution des algorithmes *Calcul_1 et Calcul_2.*

```
Algorithme Calcul_1
Variables :
A, B, C, i : entier
Début
1  A ← 1
2  B ← A
3  Pour i allant de 1 à 3 faire
4      A ← A + 2*B
5      B ← B + i
   Fin pour
6  C ← A – B
7  Écrire (A, B, C)
Fin
```

```
Algorithme Calcul_2
Variables :
A, B, C, D, E : entier
Début
1  A ← 5
2  B ← 2
   Tant que (B < 10) faire
3      D ← B + 2
       C ← A+B
       B ← B + 2
   Fin TQ
4  E ← C + D
5  Écrire (E)
Fin
```

- Montrer le tracé d'exécution de l'algorithme ***Calcul_3*** *pour les valeurs* ***N = 2*** *et* ***N=3****.*

```
Algorithme Calcul_3
Variables :
N, I, R, S, P: entier
Début
1  Lire (N)
   R ← N
   I ← 1
   Répéter
2      P ← 0
       S ← 1
       Tant que (S <= N) faire
3          P ← P + R
           S ← S + 1
       Fin TQ
4      R ← P
       I ← I + 1
   Jusqu'à (I > 2)
5  Écrire (R)
Fin
```

❖ Exercice 02 :

Écrire un algorithme qui permet de calculer ***la somme*** deux tableaux de *20* éléments *entiers*.

❖ Exercice 03 :

Écrire un Algorithme qui calcule ***le nombre*** de valeurs ***paires*** dans un tableau de ***10*** *éléments entiers*.

❖ Exercice 04 :

Écrire un **algorithme** qui affiche le nombre de répétition d'un nombre "X" dans un tableau de 20 éléments réels.

❖ Exercice 05 :

Soit un tableau de 20 entiers. Écrire un ***Algorithme*** qui détermine le plus grand élément de ce tableau.

❖ Exercice 06 :

Soit **Tab** un tableau de 10 éléments entiers. Écrire un **Algorithme** qui détermine les positions du ***plus petit*** et du ***plus grand*** élément de ce tableau.

❖ Exercice 07 :

Écrire un **algorithme** qui permet de ***trier*** un tableau de 20 éléments réels.

❖ Exercice 08 :

Écrire un **algorithme** qui permet de calculer ***la somme*** de deux matrices de 10 lignes et 15 colonnes d'éléments entiers.

❖ Exercice 09 :

Écrire un **algorithme** qui permet de calculer le produit ***de deux*** matrices A [5,8] et B [8,10] d'éléments réels.

❖ Exercice 10 :

Écrire un **programme** qui détermine la valeur maximale avec ses indices d'une matrice de 5 lignes et 5 colonnes d'éléments entiers.

❖ Exercice 11:

Écrire un **algorithme** qui permet de calculer pour chaque ligne, la somme des valeurs paires et pour chaque colonne, le nombre des éléments impairs d'une matrice ***de 4 lignes*** et ***8 Colonnes*** d'éléments ***entiers***.

Exercice 01 :

Algorithme Calcul-1 :

Etape	A	B	C	I	Ecran
1	1	/	/	/	/
2	1	1	/	/	/
3	1	1	/	1	/
4	3	1	/	1	/
5	3	2	/	1	/
3	3	2	/	2	/
4	7	2	/	2	/
5	7	4	/	3	/
3	7	4	/	3	/
4	15	4	/	3	/
5	15	7	/	3	/
6	15	7	8	3	/
7	15	7	8	3	15 – 7 - 8

Algorithme Calcul-2 :

Etape	A	B	C	D	E	Ecran
1	5	/	/	/	/	/
2	5	2	/	/	/	/
3	5	4	7	4	/	/
3	5	6	9	6	/	/
3	5	8	11	8	/	/
3	5	10	13	10	/	/
4	5	10	13	10	23	/
5	5	10	13	10	23	23

Algorithme Calcul-3 :

Pour N=2

Etape	N	I	R	S	P	Ecran
1	2	1	2	-	-	-
2	2	1	2	1	0	-
3	2	1	2	2	2	-
3	2	1	2	3	4	-
4	2	2	4	3	4	-
2	2	2	4	1	0	-
3	2	2	4	2	4	-
3	2	2	4	3	8	-
4	2	3	8	3	8	-
5	2	3	8	3	8	8

Pour **N=3**

Etape	N	I	R	S	P	Ecran
1	3	1	3	-	-	-
2	3	1	3	1	0	-
3	3	1	3	2	3	-
3	3	1	3	3	6	-
3	3	1	3	4	9	-
4	3	2	9	4	9	-
2	3	2	9	1	0	-
3	3	2	9	2	9	-
3	3	2	9	3	18	-
3	3	2	9	4	27	-
4	3	3	27	4	27	-
5	3	3	27	4	27	27

Exercice 02 :

```
Algorithme Somme ;
Constantes
N = 20 ;
Variables :
A, B, C : tableau [1..N] d'entiers ;
i : entier ;
Début
    Pour i allant de 1 à N faire
        Lire (A[i]) ;
    Fin pour
    Pour i allant de 1 à N faire
        Lire (B[i]) ;
    Fin pour
    Pour i allant de 1 à N faire
        C[i] ← A[i] + B[i] ;
    Fin pour
    Pour i allant de 1 à N faire
        Ecrire (C[i]) ;
    Fin pour
Fin
```

Exercice 03 :

```
Algorithme Valeurs-Paires ;
Constantes
N = 100 ;
Variables :
Tab : tableau [1..N] d'entiers ;
Co, i : entier ;
Début
    Pour i allant de 1 à N faire
        Lire (Tab[i]) ;
    Fin pour
    Co ← 0 ;

    Pour i allant de 1 à N faire
        Si (Tab[i] mod 2 = 0) alors
            Co ← Co + 1 ;
        Fin si
    Fin pour
    Ecrire (« Nombre de valeurs paires= », Co) ;
Fin
```

Exercice 04 :

```
Algorithme Occurrence
Constantes
N = 50
Variables :
Tab : tableau [1..N] de réels ;
Co, i : entier ;
X : réel ;
Début
    Lire (x) ;
   Pour i allant de 1 à N faire
       Lire (Tab[i]) ;
   Fin pour
   Co ⟵ 0
   Pour i allant de 1 à N faire
       Si (Tab[i] = x) alors
           Co ⟵ Co + 1 ;
       Fin si
   Fin pour
Ecrire (« Nombre de répétitionde x= », Co) ;
Fin
```

Exercice 05 :

```
Algorithme Valeur-Max ;
Constantes
N = 200 ;
Variables :
Tab : tableau [1..N] d'entier ;
Max, i : entier ;
X : réel
Début
   Pour i allant de 1 à N faire
       Lire (Tab[i]) ;
   Fin pour
       Max ⟵ Tab[i] ;
   Pour i allant de 2 à N faire
       Si (Tab[i] >Max) alors
          Max ⟵ Tab[i] ;
       Fin si
   Fin pour
   Ecrire (« Valeur Max= », Max) ;
Fin
```

Exercice 06 :

```
Algorithme PosMinMax
Constantes
N = 100
Variables :
Tab : tableau [1..N] d'entier ;
PosMax, PosMin, Min, Max, i : entier ;
Début
   Pour i allant de 1 à N faire
       Lire (Tab[i]) ;
   Fin pour
   Max ⟵ Tab[1] ;
   Min ⟵ Tab[1] ;
   PosMin ⟵ 1 ;
   PosMax ⟵ 1 ;
  Pour i allant de 2 à N faire
       Si (Tab[i] > Max) alors
          Max ⟵ Tab[i] ;
          PosMax ⟵ i ;
       Fin si
  Fin pour
  Ecrire (« Position du minimum », PosMin)
  Ecrire(« Position du maximum», PosMax)
Fin
```

Exercice 07 :

```
Algorithme TriCroissant
Constantes
N = 20
Variables :
Tab : tableau [1..N] de réels ;
T : réel ;  i,j :entier ;
Début
   Pour i allant de 1 à N faire
       Lire (Tab[i]) ;
   Fin pour
    Pour i allant de 1à N-1 faire
        Pourj allant de (i+1) à faire
          Si (Tab[i]>Tab[j]) alors
              T ⟵ Tab[i];
             Tab[i] ⟵ Tab[i];
              Tab[j] ⟵ T ;
          Fin si
        Fin pour
   Fin pour
    Pour i allant de 1àN faire
      Ecrire (Tab[i]) ;
  Fin pourFin
Fin
```

Exercice 08 :

Algorithme *Somme*
Constantes
N = 10
M = 15
Variables :
A, B, C : matrice [1..N, 1..M] d'entiers
i, j : entier
Début
Pour *i allant de 1 à N* ***Faire***
Pour *j allant de 1 à M****Faire***
Lire (A [i, j])
Fin pour
Fin pour
Pour *i allant de 1 à N* ***Faire***
Pour *j allant de 1 à M****Faire***
Lire (B [i, j])
Fin pour
Fin pour
Pour *i allant de 1 à N* ***Faire***
Pour *j allant de 1 à M****Faire***
C [i, j] ← A [i, j] + B [i, j]
Fin pour
Fin pour
Pour *i allant de 1 à N* ***Faire***
Pour *j allant de 1 à M****Faire***
Écrire (C [i, j])
Fin pour
Fin pour
Fin

Exercice 09 :

Algorithme *Produit*
Constantes
N = 5
P = 8
M = 10
Variables :
A : matrice [1..N, 1..P] de réel :
B : matrice [1..P, 1..M] de réels ;
C : matrice [1..N, 1..M] de réels :
i, j, k, Som : entier
Début
Pour *i allant de 1 à N* ***Faire***
Pour *j allant de 1 à P* ***Faire***
Lire (A [i, j])
Fin pour
Fin pour
Pour *i allant de 1 à P* ***Faire***
Pour *j allant de 1 à M****Faire***
Lire (B [i, j])
Fin pour
Fin pour
Pour *i allant de 1 à N* ***Faire***
Pour *j allant de 1 à M****Faire***
Som ← 0
Pour *k allant de 1 à P* ***Faire***
Som ← Som + A [i, k] B [k, j]*
Fin pour
C [i, j] ← Som
Fin pour
Fin pour
Pour *i allant de 1 à N* ***Faire***
Pour *j allant de 1 à M****Faire***
Écrire (C [i, j]) ;
Fin pour
Fin pour
Fin

Exercice 10 :

```
Algorithme MaxIndices
  Constantes
  N = 5
  M = 5
Variables :
Mat : matrice [1..N, 1..M] d'entiers
 i, j, Max, iMax, jMax : entier
Début
    Pour i allant de 1 à N Faire
      Pour j allant de 1 à MFaire
       Lire (Mat [i, j])
      Fin pour
    Fin pour
      Max ??Mat [1, 1]
        iMax??1
       jMax??1
     Pour i allant de 1 à N Faire
       Pour j allant de 1 à MFaire
         Si (Mat [i, j] > Max) Alors
           Max??Mat [i, j]
           iMax??i
          jMax??j
         Fin si
       Fin pour
     Fin pour
      Écrire ('Valeur max=', Max)
   Écrire ('Indices valeur max=',iMax, jMax)
    Fin pour
Fin
```

Exercice 11 :

```
Algorithme MaxIndices
Constantes
N = 5
M = 5
Variables :
Mat : matrice [1..N, 1..M] d'entiers
i, j, Max, iMax, jMax : entier
Début
    Pour i allant de 1 à N Faire
      Pour j allant de 1 à MFaire
       Lire (Mat [i, j])
      Fin pour
    Fin pour
      Max ??Mat [1, 1]
        iMax??1
       jMax??1
     Pour i allant de 1 à N Faire
       Pour j allant de 1 à MFaire
         Si (Mat [i, j] > Max) Alors
           Max??Mat [i, j]
           iMax??i
           jMax??j
         Fin si
       Fin pour
     Fin pour
    Écrire ('Valeur max=', Max)
   Écrire ('Indices valeur max=',iMax, jMax)
Fin
```

Chapitre 4 : **Les sous programmes (Procédures et 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 endroits.
- Un algorithme écrit de cette façon devient difficile à comprendre et à gérer dès qu'il dépasse deux pages ⇨ 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-algorithmes (programmes) qui sont: les procédures et les fonctions.

2. Les procédures

- Une procédure est un sous-programme identifié par *un nom*
- il peut être appelé dans *un autre programme* ou dans des différents lieux du même programme ou même dans un autre s*ous-programme*.
- il permet d'effectuer des actions par un simple appel comme une instruction en utilisant des données différentes.
- Une procédure renvoie plusieurs valeurs (par une) ou aucune valeur.

2.1 Déclaration d'une procédure :

Syntaxe :

```
Procédure nom_proc(liste et déclaration de paramètres);
Variables identificateurs : type ;

 Début
 |   Instruction(s) ;
 |
 FIN
```

- Une procédure a la même structure qu'un programme.
- Après le nom de la procédure, il faut donner la liste des paramètres (s'il y en a) avec leur type ***respectif.***
- Ces paramètres sont appelés ***paramètres formels***. Leur valeur n'est pas connue lors de la création de la procédure.

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_proc (liste de paramètres) ;
```

Remarque :

- Les paramètres utilisées lors de l'appel d'une procédure sont appelés ***paramètres effectifs***.
- Ces paramètres donneront leurs valeurs ***aux paramètres formels***.
- Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de ***la partie principale*** (algorithme principal)**.**

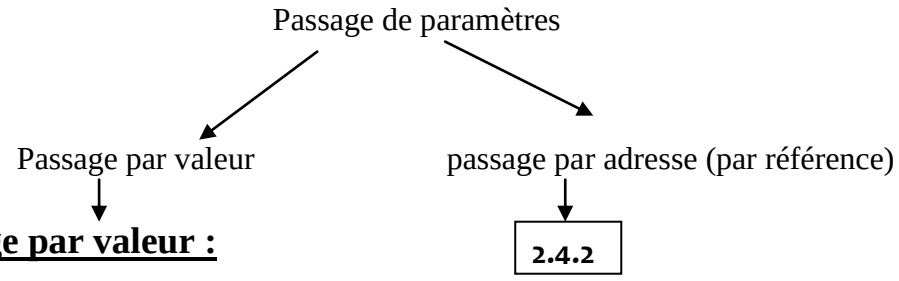
2.3 Notions de variables globales et de variables locales

En programmation informatique :

- une ***variable globale*** est une variable **déclarée** à ***l'extérieur*** du corps de toute ***Fonction*** ou **procédure** et pouvant donc être utilisée ***n'importe où*** dans le programme.
- une ***variable locale*** est une variable qui **ne peut être utilisée** que dans la ***Fonction*** ou le bloc où elle **est définie**.
- La variable **locale s'oppose** à la variable **globale** qui peut être utilisée dans **tout** le programme.

2.4 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.



2.4.1 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_valeur ;
Variables N : entier
Procédure P1(A : entier) //*Déclaration de la procédure P1
Début
| A ← A * 2 ;
| écrire(A) ;
FinProc

Début                          //*Algorithme principal
| N ← 5 ;
| P1(N) ;
| écrire(N) ;
Fin
```

Questions : faire le tracé d'exécution de cet algorithme ?qu'est-ce que vous remarquez. ?

Explication : Cet algorithme définit une ***procédure P1*** pour laquelle on utilise le passage de paramètres ***par valeur***.

- Lors de l'appel de la procédure, la valeur du paramètre **effectif N** est **recopiée** dans le **paramètres 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**.

2.4.2 Passage par référence ou par adresse :

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 du mot **clé *Var***.

Exemple : Reprenons l'exemple précédent

```
ALGORITHME Passage_par_référence ;
Variables N : entier
//*Déclaration de la procédure P1
Procédure P1 (Var A : entier)
Début
|   A ⟵ A * 2
|   écrire(A)
FinProc
//*Algorithme Principal
Début
|   N ⟵ 5
|   P1(N)
|   écrire(N)
Fin
```

Question : faire le tracé d'exécution de cet algorithme ?qu'es que vous remarquez ?

Justification :

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

- Dans cet algorithme le paramètre passé correspond à la ***référence (adresse)*** de la variable N.
- Elle est donc modifiée par l'instruction : A ← A ***2**
- **Remarque importante:** 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.***

EXEMPLE :

```
PROCEDURE somme (x,y :entier) ;
    Variables
       S :entier ;
Début
   S ← x + y ;
Fin
```

- Pour appeler la *procédure somme* de cet exemple, on écrirait :

☐ Somme (2.5 ,7) → faux (le type des paramètres effectifs doit être entier).

☐ Somme (2, 5, 9) → faux (le nombre des paramètres effectifs doit être 2 et non pas 3).

☐ Somme (2, 5) → juste (le type et le nombre des paramètres sont adéquats).

3. LES FONCTIONS

Les Fonctions sont des sous algorithmes admettant des paramètres et retournant un seul résultat (une seule valeur) de type simple qui peut apparaître dans une expression, dans une comparaison, à la droite d'une affectation, etc.

3.1 Les types de fonction :

- Une fonction intégrée dans une bibliothèque pouvant être appelé depuis n'importe quel programme principal pour calculer une valeur**.**
- ***Le type de cette fonction peut être prédéfinis*** qui est entier, réel, booléenne, caractère, chaine de caractère.

3.2 Déclaration d'une fonction

Syntaxe :

```
Fonction nom-de-fonction (Déclaration Des Paramètres):Type du résultat;
```

Exemple : Définir une fonction qui renvoie le plus grand de deux nombres différents**.**

Solution :

```
  //*Déclaration de la fonction Max
Fonction Max(X: réel, Y:réel) : réel ;
Début
    Si X > Y Alors
        ecrire(X)
    Sinon
        ecrire(Y)
    FinSi
Fin
```

3.3 Appel d'une fonction dans l'algorithme principal

- On utilise une fonction dans une expression ou un appel de procédure, exactement ***comme une variable***, mais en indiquant *ses paramètres* entre parenthèses et séparés par des virgules pour nous renvoyer un seul résultat.
- pour renvoyer la valeur calculée au l'algorithme ou programme principal, il faut écrire l'instruction suivante:

```
NomDeFonction ← Valeur Du Resultat;
```

Exemple : Ecrire un algorithme appelant, utilisant la fonction Max de l'exemple précédent.
Solution :

```
Algorithme Appel_fonction_Max
    Variables A, B, M : réel
    // * Déclaration de la fonction Max
    Fonction Max(X: réel, Y: réel) : réel
    DEBUT
        Si X > Y Alors
            max(x,y) ← X
        Sinon
            max(x,y)  ←  Y
        FinSi
    FIN

   //*Algorithme principal
    DEBUT
     Ecrire ("Donnez la valeur de A :")
     lire(A)
    Ecrire ("Donnez la valeur de B :")
     lire (B)

     //*Appel de la fonction Max
       M ← Max(A,B)

    Ecrire ("Le plus grand de ces deux nombres est : ", M)

   FIN
```

4. Portée des variables

- La portée d'une variable désigne le domaine de visibilité de cette variable.
- Une variable peut être déclarée dans deux emplacements distincts.
- Une variable déclarée dans la partie déclaration de l'algorithme principale est appelée variable globale :
  - Elle est ***accessible de n'importe où dans l'algorithme***, même depuis les procédures et les fonctions.
  - Elle existe pendant *toute* ***la durée de vie*** du programme.
- Une variable déclarée **à l'intérieur** d'une procédure (ou une fonction) est ***dite locale***.
- Elle n'est accessible qu'à la procédure au ***sein de laquelle elle définie***, les autres procédures ***n'y ont pas accès***.
- La durée de vie d'une variable locale est limitée à **la durée d'exécution** de sa procédure ou sa fonction.
- Il est possible de déclarer dans la procédure un identificateur (variable) utilisé dans un niveau englobant. Dans ce cas, ***la localité*** *masque* ***la globalité.***

```
Algorithme portée
 Variable x,y :entier ;
Procedure p1() :
  Variables
    A :entier
Debut
….
Finproc
//Algorithme principal
Début
  …..
Fin
```

```
X et Y sont des variables
globales  visibles dans tout
l'algorithme


A est une variable locale
visible uniquement à
l'intérieur de la
procédure
```

5. AVANTAGES DES PROCEDURES ET FONCTIONS

- ✓ Les procédures ou fonctions permettant *de* ***ne pas répéter plusieurs fois*** une même séquence d'instructions au *sein du programme (algorithme).*
- ✓ Structurent un algorithme en modules et augmentent *sa lisibilité* et sa *compréhensibilité*.
- ✓ facilitent la mise au point du programme (càd que *la compilation* et *la détection des erreurs* sera plus rapide en utilisant les procédures et/ou les fonctions).
- ✓ peuvent être même réalisées en dehors du contexte du programme, autrement dit, elles peuvent être rangées dans une bibliothèque d'outils et utilisées par n'importe quel autre programmes.

UNIVERSITE CONSTANTINE 1 - CAMPUS
CHAAB ERSAS
1ERE ANNEE TRONC COMMUN - ST
Module : Informatique I
Année universitaire: 2019 – 2020

TP n°06 : Les tableaux à une dimension (Vecteurs)

Partie I : Algorithmes

Exercice N°01

Ecrire un algorithme qui calcule la moyenne des valeurs d'un tableau de 30 éléments entiers.

Exercice N°02

Ecrire un algorithme qui calcule le nombre de valeurs positives et la somme des valeurs paires d'un tableau de 20 éléments.

Exercice N°03

Ecrire un algorithme qui permet de trouver la valeur maximale et la valeur minimale d'un tableau de 20 éléments réels.

Exercice N°04

Ecrire un algorithme qui affiche le nombre de répétition d'un nombre 'X' dans un tableau de 50 éléments réels.

Exercice N°05 (supplémentaire)

Ecrire un programme qui permet de trier un tableau de 25 éléments réels.

Partie II : Programmation en C

Soit T un tableau de 10 éléments entiers. Ecrire un programme en langage C qui détermine les positions du plus petit et plus grand élément de ce tableau.

UNIVERSITE CONSTANTINE 1 - CAMPUS
CHAAB ERSAS
1ERE ANNEE TRONC COMMUN - ST
Module : Informatique I
Année universitaire: 2019 – 2020

TP n°07 : Les tableaux à deux dimensions (Matrices)

Partie I : Algorithmes

Exercice N°01

Ecrire un algorithme qui permet de calculer la somme de deux matrices de 10 lignes et 15 colonnes d'éléments entiers.

Exercice N°02

Écrire un algorithme qui détermine la valeur maximale avec ses indices d'une matrice de 5 lignes et 5 colonnes d'éléments entiers.

Exercice N°03

Écrire un algorithme qui permet de calculer pour chaque ligne, la somme des valeurs paires et pour chaque colonne, le nombre des valeurs impairs d'une matrice de 4 lignes et 8 colonnes d'éléments entiers.

Exercice N°04 (supplémentaire)

Soit une matrice de 20 lignes et 20 colonnes d'éléments réels.

- Ecrire un algorithme qui permet de déterminer le numéro de la ligne dont le produit de ses éléments est maximale.
- Traduire cet algorithme en langage C.

Partie II : Programmation en C

Soit A une matrice d'ordre (NxM) de nombres entiers. Ecrire un programme en langage C qui permet de:

1) Lire la matrice A
2) Calculer la moyenne de tous les éléments de la matrice A
3) Calculer le nombre des éléments de la matrice qui sont supérieurs à cette moyenne.

UNIVERSITE CONSTANTINE 1 - CAMPUS
CHAAB ERSAS
1ERE ANNEE TRONC COMMUN - ST
Module : Informatique I
Année universitaire: 2019 – 2020

TP n°08 : Les enregistrements

Partie I : TD

1) Créer un nouveau type d'enregistrement **emp_t** qui comporte pour chaque employé d'une entreprise les informations suivantes :

   - Matricule : entier
   - Nom : chaîne de 30 caractères maximum
   - Date_recrutement : (Jour, Mois, An)
   - Etat_civil : caractère
   - Salaire : réel

2) Ecrire un bloc d'instructions qui permet de stocker les informations de 10 employés dans un tableau **TE.**
3) Ecrire une suite d'instructions qui à partir d'un numéro d'immatriculation affiche les informations de l'employé correspondant.
4) Ecrire un groupe d'instructions qui détermine le nombre des employés recrutés avant une année donnée.
5) Ecrire une suite d'instructions qui à partir du tableau TE affiche les employés dont le salaire est inférieur à une valeur donnée.

Partie II : TP

Ecrire un programme en langage C qui englobe tout ce qui précède.

UNIVERSITE CONSTANTINE 1 - CAMPUS
CHAAB ERSAS
1ERE ANNEE TRONC COMMUN - ST
Module : Informatique I
Année universitaire: 2019 – 2020

TP n°09.A : Les sous-algorithmes (Procédures)

Partie I : Algorithmes

Exercice N°01

Ecrire une procédure ***lire*** **(*a*)** qui permet de lire un nombre entier positif *a*.

Exercice N°02

Ecrire une procédure ***trier*** **(*p*,*g*)** qui permet de trier deux nombres entiers *p* et *g*.

Exercice N°03

Ecrire une procédure ***divisible*** **(*n*,*d*)** qui permet de savoir si l'entier *n* est divisible par *d* ou non. On pourra utiliser un nouveau paramètre nommé *r* afin de renvoyer le résultat ***divisible*** **(*n*,*d*,*r*)**.

Exercice N°04

Ecrire une procédure ***parfait*** **(*x*)** qui nous permet de dire si un nombre est parfait ou non, et qui appelle la procédure Divisible. On appelle un nombre parfait un nombre égal à la somme de ses diviseurs propres (un diviseur propre est un diviseur autre que le nombre lui-même). Ex: 6 = 1 + 2 + 3 et: 28 = 1 + 2 + 4 + 7 + 14

Partie II : Programmation en C

Ecrire un programme en langage C qui affiche tous les nombres parfaits compris entre deux valeurs entières positives lues *x* et *y*, en utilisant les procédures précédentes.

Exécutez le programme en donnant:

1) *x*=1 et *y*=30
2) *x*=500 et *y*=450
3) *x*=9000 et *y*=8000

Partie I : Algorithmes

Exercice N°01

```
Algorithme Moy30Val;
Constantes n=30;
Variables
   Tab: tableau [1..n] d'entiers;
   Som, i: entiers;  Moy: réel;

Début
   Pour i allant de 1 à n Faire
      Lire (Tab[i])
   FinPour;
   Som ← 0;

   Pour i allant de 1 à n Faire
      Som ← Som+ Tab[i]
   FinPour;

   Moy ← Som/n;
   Ecrire ('Moyenne = ', Moy)
Fin.
```

Exercice N°02

```
Algorithme Calcule;
Constantes n=20;
Variables
   Tab: tableau [1..n] d'entiers;
   i, svp, nvp: entiers;
   //nvp : nombre de valeurs positives
   //svp : somme des valeurs paires

Début
   Pour i allant de 1 à n Faire
      Lire (Tab[i])
   FinPour;

   nvp ← 0;  svp ← 0;

   Pour i allant de 1 à n Faire
      Si (Tab[i] >= 0) Alors
         nvp ← nvp + 1
      FinSi;

      Si (Tab[i] mod 2 = 0) Alors
         svp ← svp + Tab[i]
      FinSi
   FinPour;

   Ecrire('Le nombre de valeurs
               positives est: ', nvp);
   Ecrire('La somme des valeurs paires
                        est: ', svp)
Fin.
```

Exercice N°03

```
Algorithme ValMinMax;
Constantes n=20;
Variables
   T: tableau [1..n] de réels;
   i: entier;  max,min: réels;

Début
   Pour i allant de 1 à n Faire
      Lire (T[i])
   FinPour;
   max ← T[1];  min ← T[1];

   Pour i allant de 2 à n Faire
      Si (T[i] > max) Alors
         max ← T[i]
      FinSi;
      Si (T [i] < min) Alors
         min ← T[i]
      FinSi
   FinPour;

   Ecrire('Min=',min,' Max=', max);
Fin.
```

Exercice N°04

```
Algorithme Occurrence;
Constantes n=50;
Variables
   Vect: tableau [1..n] de réels;
   i, co: entiers;  X: réel;

Début
   Pour i allant de 1 à n Faire
      Lire (Vect[i])
   FinPour;
   Lire (X);

   co ← 0;
   Pour i allant de 1 à n Faire
      Si (Vect[i] = X) Alors
         co ← co + 1
      FinSi
   FinPour;

   Ecrire ('Le nombre de répétition
   de ',X,' est: ', co);
Fin.
```

Exercice N°05

```c
#include <stdio.h>
#define N 25
main()
{
   float V[N];
   int i, j; float T;

   printf("Entrer 25 nombres reels: \n");
   for (i=0; i<N ; i++)
        scanf("%f", &V[i]);

   for (i=0; i<N-1 ; i++)
      for (j=i+1; j<N ; j++)
         if (V[i] > V[j])
         {
            T = V[i];
            V[i] = V[j];
            V[j] = T;
         }

   for (i=0; i<N ; i++)
      printf("%f\n", V[i]);
}
```

Partie II : Programmation en C

```c
#include <stdio.h>
#define N 5
main()
{
   int T[N];
   int i, max, min, Pmax, Pmin;

   printf("Entrer 5 nombres entiers: \n");
   for (i=0; i<N ; i++)
         scanf("%i", &T[i]);

   max = T[0];   min = T[0];
   Pmax = 0;     Pmin = 0;
   for (i=1; i<N ; i++)
   {
      if (T[i] > max)
      {
           max  = T[i];
           Pmax = i;
      }
      if (T[i] < min)
      {
           min  = T[i];
           Pmin = i;
      }
   }
   printf("Min= %i\t PosMin= %i\n", min, Pmin);
   printf("Max= %i\t PosMax= %i\n", max, Pmax);
}
```

Partie I : Algorithmes

Exercice N°01

```
Algorithme Som2Mat;
Constantes n=10; m=15;
Variables
   A,B,S: matrice[1..n,1..m]d'entiers;
   i, j : entiers;
Début
  Pour i allant de 1 à n Faire Pour j
     allant de 1 à m Faire
        Lire (A[i,j])
     FinPour
  FinPour;
  Pour i allant de 1 à n Faire Pour j
     allant de 1 à m Faire
        Lire (B[i,j])
     FinPour
  FinPour;
  Pour i allant de 1 à n Faire Pour j
     allant de 1 à m Faire
        S[i,j]← A[i,j]+B[i,j]
     FinPour
  FinPour;
  Pour i allant de 1 à n Faire Pour j
     allant de 1 à m Faire
        Ecrire (S[i,j])
     FinPour
  FinPour;
Fin
```

Exercice N°02

```
Algorithme MaxIndices;
Constantes n=5; m=5;
Variables
   Mat: matrice[1..n,1..m]d'entiers;
   i,j, Max,iMax,jMax: entiers;

Début
  Pour i allant de 1 à n Faire Pour j
     allant de 1 à m Faire
        Lire (Mat[i,j])
     FinPour
  FinPour;
  Max ← Mat[1,1];
  iMax ← 1;   jMax ← 1;
  Pour i allant de 1 à n Faire Pour j
     allant de 1 à m Faire
        Si (Mat[i,j]>Max) Alors
           Max ← Mat[i,j];
           iMax ← i;   jMax ← j
        FinSi
     FinPour
  FinPour;
  Ecrire ('Valeur maximale =', Max);
  Ecrire ('Indices =',iMax, jMax);
Fin.
```

Exercice N°03

```
Algorithme SVP_NVIP;
Constantes n=4; m=8;
Variables
   Mat: matrice[1..n,1..m]d'entiers;
   i,j,S,Co : entiers;

Début
  Pour i allant de 1 à n Faire Pour j
     allant de 1 à m Faire
        Lire (Mat[i,j])
     FinPour
  FinPour;

  Pour i allant de 1 à n Faire
     S←0;
     Pour j allant de 1 à m Faire Si
        (Mat[i,j]mod2 = 0) Alors
           S ← S + Mat[i,j];
        FinSi
     FinPour
     Ecrire('Ligne:',i,'Somme:'S);
  FinPour;

  Pour j allant de 1 à m Faire
     Co ← 0;
     Pour i allant de 1 à n Faire Si
        (Mat[i,j]mod2 <> 0) Alors
           Co ← Co + 1;
        FinSi
     FinPour
    Ecrire('Colonne:',j,'nombre:'co);
  FinPour;
Fin.
```

Exercice N°04

```
Algorithme Calcule;
Constantes n=20;
Variables
   X: matrice[1..n,1..n]de réels;
   i,j,ligne : entiers; Max,prod :
   réels;
Début
  Pour i allant de 1 à n Faire
     Pour j allant de 1 à n Faire
        Lire (X[i,j])
     FinPour
  FinPour;

  Prod ← 1;
  Pour j allant de 1 à n Faire
     Prod ← Prod * X[1,j]
  FinPour;

  ligne ← 1;
  Max ← Prod;

  Pour i allant de 2 à n Faire
     Prod ← 1;
     Pour j allant de 1 à n Faire
        Prod ← Prod * X[i,j]
     FinPour;
     Si (Prod > Max) Alors
        ligne ← i;
        Max ← Prod;
     FinSi
  FinPour;

  Ecrire(' Le numéro de la ligne
  est', ligne);
Fin.
```

Partie II : Programmation en C

```
#include <stdio.h>
#define N 3
#define M 4
main()
{
  typedef int A_Type[N][M];
  A_Type A;
  int i, j, som, co; float moy;

  printf("Entrer les elements de
              la matrice A: \n");
  for (i=0; i<N ; i++)
      for (j=0; j<M ; j++)
         scanf("%i", &A[i][j]);

  som = 0;
  for (i=0; i<N ; i++)
      for (j=0; j<M ; j++)
         som = som+ A[i][j];
  moy = (float)som/(N*M);

  co = 0;
  for (i=0; i<N ; i++)
      for (j=0; j<M ; j++)
         if (A[i][j] > moy)
            co++;

  printf("Le nombre des elements
  superieurs a %f est: %i", moy, co);
}
```

```
#include <stdio.h>
#define n 2
main()
{                              //Création des types
   typedef struct { int jour, mois, an;
                  } date_t;
   typedef struct {
              int mtc;
              char nom[30];
              date_t dr;
              char ec;
              float slr;
             } emp_t;
   typedef emp_t tab_t[n];
                              //Déclaration des variables
   tab_t TE;
   int i, co;
   int m, an;
   float slrmax;

   for (i=0; i<n; i++)
   {
        printf(" \n Employe %i : \n",i+1);
        printf(" Matricule : ");scanf("%i", &TE[i].mtc);
        printf(" Nom : ");      scanf("%s", &TE[i].nom);
        printf(" Date de recrutement \n");
        printf("\t Jour : ");   scanf("%i", &TE[i].dr.jour);
        printf("\t Mois : ");   scanf("%i", &TE[i].dr.mois);
        printf("\t An : ");     scanf("%i", &TE[i].dr.an);
        printf(" Etat civil : ");     scanf("%s", &TE[i].ec);
        printf(" Salaire : ");  scanf("%f", &TE[i].slr);
   }

   printf("\n Entrer un matricule : ");
   scanf("%i", &m);
   for (i=0; i<n; i++)
       if (TE[i].mtc == m)
       {
           printf("\n Matricule : %i", TE[i].mtc);
            printf("\n Nom       : %s", TE[i].nom);
            printf("\n Salaire   : %f", TE[i].slr);
       }

   printf("\n\n Entrer une annee : ");
   scanf("%i", &an);
   co=0;
   for (i=0; i<n; i++)
       if (TE[i].dr.an <= an)
           co=co+1;
   printf(" Le nombre des employes recrutes avant %i est : %i", an, co);

   printf("\n\n Entrer un salair : ");
   scanf("%f", &slrmax);

   printf(" Les employes ayant un salair inferieur a %f sont : ", slrmax);
   for (i=0; i<n; i++)
       if (TE[i].slr <= slrmax)
            printf("\n\t %i  %s  %f", TE[i].mtc, TE[i].nom, TE[i].slr);
}
```

UNIVERSITE CONSTANTINE 1 – CAMPUS CHAAB ERSAS 1ERE ANNEE TRONC COMMUN-ST
Module: Informatique II - Corrigé Série d'exercices n°09.A: Les procédures.

Partie I : Algorithmes

Exercice N°01

```
void lire(int *a)  // passage par adresse puisque a sera modifiée
 {
    printf("Entrer un nombre entier positif : ");
    scanf("%i", a);   // a=&x
 }
```

Exercice N°02

```
void trier(int *p, int *g)   // passage par adresse
 {
     int t;
     if (*p>*g)
     {   T  = *p;
         *p = *g;
         *g = t;   }
 }
```

Exercice N°03

```
void divisible(int n, int d, int *r) //n,d: par valeur r: par adresse
 {
     if (n%d == 0) *r = 1;
     else          *r = 0;      *r = (n%d == 0);
 }
```

Exercice N°03

```
void parfait(int x, int *p)   //x: par valeur  p: par adresse
 {
     int som,d,r;
     som = 1;
     for (d=2; d<=(x/2); d++)
     {
         divisible(x,d,&r);
         if (r)
               som=som+d;
     }
     *p = (som == x);
 }
```

Partie II : Programmation en C

```
#include <stdio.h>
            // ← inserer les quatre procedures
main()
{
    int x,y,n,p;
    lire(&x);
    lire(&y);
    trier (&x,&y);
    for (n=x; n<=y; n++)
    {
        parfait(n,&p);
        if (p)
             printf ("\t %i \n", n);
    }
}
```

L'exécution du programme :

1) x=1 et y=30 : → 1 6 28
2) x=500 et y=450 : → 496
3) x=9000 et y=8000 : → 8128

NB:

- On distingue deux genres de sous-programmes : les procédures et les fonctions mais le langage C permet de définir seulement les fonctions.
  En effet une procédure en langage C n'est autre qu'une fonction qui ne retourne aucun résultat dans son nom (void) !

- Si le passage est par valeur donc la fonction appelée traite l'information et ne modifie en aucun cas la variable passée par l'appelant ; par contre si le passage est par variable, la fonction appelée reçoit l'adresse (&) de la valeur et non pas la valeur elle-même. A l'aide de cette adresse (&) la machine va pointer (*) l'emplacement physique et traite l'information qui y est. Donc automatiquement cette information va éventuellement changer !

UNIVERSITE CONSTANTINE 1 – CAMPUS CHAAB ERSAS 1ERE ANNEE TRONC COMMUN-ST
Module: Informatique II - Corrigé Série d'exercices n°09.A: Les fonctions.

Partie I : TD

Exercice N°01

```
int divisible(int n, int d)
 {
     if (n%d == 0) return 1;
     else          return 0;
 }
```

ou bien

```
return (n%d == 0);
```

Exercice N°02

```
void lire(int *n)
 {
    printf("Entrer un nombre entier positif entre 100 et 999 : ");
    do{
        scanf("%i", n);
        if (*n < 100)
                  printf ("Plus grand ! \n");
        else
                if (*n > 999)
                          printf ("Plus petit ! \n");
    }while((*n < 100) || (*n > 999));
 }
```

Exercice N°03

```
int cubique(int n)
 {
     int c,d,u;
     c = n / 100;
     d = (n % 100) / 10;
     u = n % 10;
     return (n == c*c*c+d*d*d+u*u*u);
     //ou return (n == (int)(pow(5,3)+pow(1,3)+pow(3,3)));
 }
```

Partie II : TP

```
#include <stdio.h>
main()
{
    int x;
    char rep;
    do{
         lire(&x);
         if (cubique(x))
              printf("%i est cubique \n", x);
         else
              printf("%i n'est pas cubique \n", x);
         printf("Avez-vous un autre nombre a tester ? ");
         scanf("%s", &rep);
    }while((rep == 'o')||(rep == 'O'));
}
```

L'exécution du programme :

1) x = 21 : Plus grand !
2) x = 089 : Plus grand !
3) x = 135 : n'est pas cubique
4) x = 153 : est cubique
5) x = 257 : n'est pas cubique
6) x = 370 : est cubique
7) x = 371 : est cubique
8) x = 372 : n'est pas cubique
9) x = 399 : n'est pas cubique
10) x = 0407 : est cubique
11) x = 0555 : n'est pas cubique
12) x = 1123 : Plus petit !

CHAPITRE : LES ENREGISTREMENTS

1. STRUCTURE DE DONNEES HETEROGENES

- Si on souhaite stocker des données qui n'ont pas forcément le même type au sein d'une même structure, par exemple : les informations liées à un produit quelconque, La première question qui se pose est : quelle est la structure de donnée nécessaire ?
- Il est clair que dans un tableau, tous les éléments le constituant ont obligatoirement le même type, Les tableaux ne sont pas une solution adéquate.
- La solution est de créer une nouvelle structure de type ' *enregistrement*' qui *comporte* à la fois les données *numériques* (quantité, prix unitaire, prix total) et les données (*alphanumériques* (Référence, désignation),
- Ce nouveau type ou cette *une nouvelle structure* permet de représenter des données hétérogènes (c'est-à-dire *de différents types)* par le regroupage de plusieurs éléments qui peuvent être simples fondée sur d'autres types existants ou complexes.

2. STRUCTURE D'UN ENREGISTREMENT (NOTION DE CHAMPS) :

- Un enregistrement est une structure de données qui est un peu complexe par rapport aux structures vues précédemment qui sont : les types primitifs tels que : entier, réel, caractère), chaine de caractères, et les vecteurs a une et deux dimension (tableaux et matrices).
- Cette structure (enregistrement) nous permette de représenter un ensemble de données dites ***champs*** *hétérogènes (pas nécessairement le même type)* dans la même variable.
- En comparaison avec la structure de vecteur, on peut considérer un enregistrement comme étant un tableau *à une dimension*, *où chaque case de ce tableau représente une composante d'un type donné qui s'appelle* ***un champ****.*
- *La différence réside dans :*
  - Une case de tableau est accessible à travers un indice entier, par contre un champ est repéré par un nom (identificateur) ;
  - Les cases du tableau sont toutes du même type (structure homogène), par contre, les champs de l'enregistrement ne sont pas obligatoirement du même type (structure hétérogène).
  - La déclaration de tableau en algorithmique se faite dans la clause variable seulement, mais la déclaration de l'enregistrement se réalise par les deux clauses type et variables.

2.1 DEFINITION

Un enregistrement appelé aussi un article ou record en anglais est composé d'un nombre fixe de champs défini par l'utilisateur qui peuvent être de types différents. La liste des champs est composée de définitions identiques à celle de variables. Par exemple l'enregistrement « date » est composé de trois champs « jour », « mois » et année.

2.2 DECLARATION

La déclaration des types structures se fait dans une section spéciale des algorithmes appelée ***Type,*** qui précède la section des variables et succède à la section (clause) des constantes.

La syntaxe générale pour déclarer un enregistrement est comme suit :

En algorithmique	En langage C
Type *<id_Enreg> =* ***Enregistrement*** *<id_ch1> : <type1>* *<id_ch1> : <type2>* *...* *<id_chN> : <typeN>* ***Fin_ <id_Enreg>***	***Typedef struct {*** *<type1> <id_ch1> ;* *<type2> <id_ch1> ;* *...* *<typeN> <id_chN> ;* ***}id-structure ;*** *.....*

Où <id_ch1>, <id_ch2>, …, <id_chN> sont ***les identificateurs*** *des* ***champs*** *et <type1>, <type2>, …, <typeN>* ***leurs types*** *respectifs.*

Pour l'exemple du Produit précédent, la déclaration de l'enregistrement sera comme suit :

En algorithmique	***En langage C***
Type *Produit = Enregistrement* *Reference : Chaîne [5]* *Designation : Chaîne[25]* *Quantite : Entier* *Prix_U : Réel* *Fin_produit* ***Variables*** *Prod : produit*	***Typedef struct*** *{* *char ref[5] ;* *char Designation[25];* *int quantite* *float prix_u ;* *} produit ;*

2.2.1 TYPE ENUMERE

- les variables de ce type prennent une valeur parmi un ensemble. par exemple pour un feu de circulation, la couleur est : vert, orange ou rouge.
- le langage c propose deux syntaxes pour définir ce type.

A). première syntaxe :

enum nouveautype {liste de symboles} ;

- Cette syntaxe définit un nouveau type qui s'appelle ***enum nouveautype***
- le domaine des variables de ce type est la liste des symboles fournis.
- ces symboles sont équivalents à des constantes entières commençant par 0.

En langage C	En algorithmique
Enum jour{dimanche, lundi, mardi, jour=(dimanche,lundi mercredi,jeudi,vendredi **Enum** jour j ; Jour j ; Main(){ J= jeudi ; ……..	Déclaration **Type** jour=(dimanche,lundi,mardi,mercredi jeudi,vendredi, samedi} ; , **variables** J : jour ; **début** J ← jeudi ; **Fin**

B DEUXIEME SYNTAXE :

Cette syntaxe permet de définir un nouveau (sans évoquer le mot clé ***enum***).

Syntaxe : **Typedef enum {liste de symboles} nouveautype** ;

Exemple : **Typede**f enum{dimanche, lundi, mardi, mercredi, jeudi, vendredi, samedi} jour;
Jour j ;

REMARQUES

- Chaque champ a exactement les mêmes propriétés qu'une variable de même type et doit avoir un nom différent.
- Les éléments d'un enregistrement peuvent être à leur tour des structures (tableaux, enregistrements, ……)

3. MANIPULATION DES STRUCTURES D'ENREGISTREMENTS

Comme pour les tableaux, il n'est pas possible de manipuler un enregistrement globalement, La manipulation d'un enregistrement se fait au travers de « ses champs ».

3.1 ACCES AUX CHAMPS D'UN ENREGISTREMENT

- Alors que les éléments d'un tableau sont accessibles au travers de leur indice, les champs d'un enregistrement sont accessibles à travers leur nom, grâce à l'opérateur '.'.
- Pour référencer ou accéder à un champ quelconque d'un enregistrement, on utilise La variable enregistrement (identificateur) suivie d'un point puis du nom du champ auquel on veuille accéder. Syntaxe :

  <Variable_Enregistrement>.<Nom du champ>
- Par exemple, pour accéder au champ *Quantite* de la variable *prod,* de l'exemple précédent, on utilise l'expression : **prod. Quantite**.

3.2 AFFECTATION

L'affectation de valeurs aux différents champs d'une variable de type d'enregistrement se fait en respectant la syntaxe :

En algorithmique	En langage C
Nom_variable ← valeur **Exemple** **prod. Quantite ← 50**	**Nom_variable = valeur ;** **Exemple** **prod. Quantite = 50**

NB : il est possible d'affecter une variable enregistrement dans une autre de même structure, dans ce cas tous les champs seront recopies.

3.3 LECTURE

Cette opération sert à remplir les champs d'un enregistrement par les valeurs entrées par l'utilisateur à travers le clavier, Sa syntaxe est :

En algorithmique	En langage C
Lire(Nomvariable.champ) **Exemple** **Lire(prod. Designation)**	**Scanf(''code format'',& Nom_variable .champ)** **Exemple** **scanf(''%s'',&prod. Designation) ;**

3.4 ECRITURE

Cette opération sert à afficher sur écran, le contenu des champs d'un enregistrement composant l'enregistrement, Sa syntaxe est :

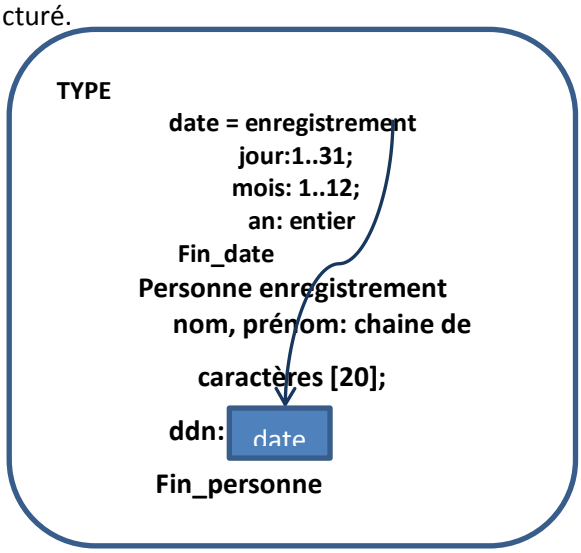
En algorithmique	En langage C
ecrire(Nomvariable.champ) Exemple **ecrire(prod.** Prix_U**)**	**printf(''code format'', Nom_variable .champ)** Exemple **printf (''%f'',** prod**.** Prix_U**) ;**

Remarque 1: la seule opération possible sur les enregistrements de même type est l'affectation

Remarque 2 : Nous ne pouvons pas lire ou écrire globalement un enregistrement, pour faire, il faudra lire ou écrire chaque champ qui le compose.

Remarque 3 : Un type structuré (enregistrement) peut être utilisé comme type pour des champs d'un autre type structuré.

Exemple :



- Pour accéder à l'année de naissance d'une personne, il faut utiliser deux fois l'opérateur '.'

Remarque 4: On peut définir et initialiser une structure enregistrement, en même temps et cela est possible grâce aux deux syntaxes suivantes :

- **Syntaxe 1**
  Struc nom-type nom-variable = {valeurs-champs} ;
- **Syntaxe 2**
  Nom-typedef nom-variable = {valeurs-champs} ;

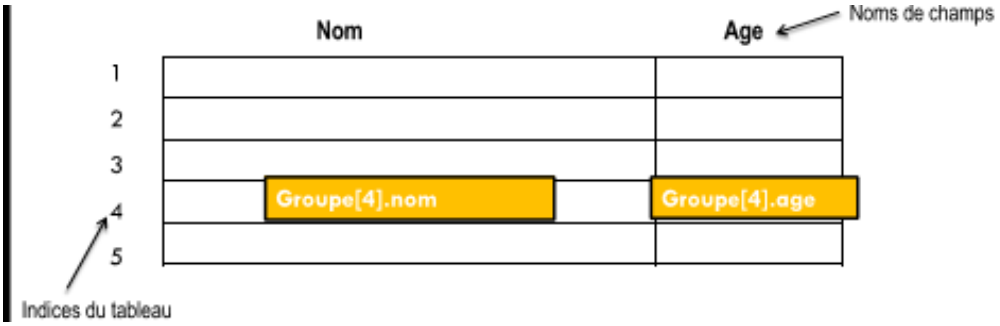
3.5 Tableau d'enregistrement

- Il arrive souvent que l'on veuille traiter non pas **un seul enregistrement** mais **plusieurs**.
- Par exemple, on veut pouvoir traiter un groupe de personne. On ne va donc pas créer autant de variables du type personne qu'il y a de personnes.
- On va créer un tableau regroupant toutes les personnes du groupe. « Il s'agit alors d'un ***tableau d'enregistrements*** »
  Exemple

En algorithmique	En langage C
Type Personne = enregistrement nom: chaine de caractère [15]; age: entier; Fin_personne **Variables** groupe: tableau [1..20] de personne;	***#define N 20*** ***Typedef struct*** *{* *Char nom[15] ;* *int age ; } personne ;* *main(){* typedef personne groupe[N];

Illustration :

- Chaque élément du tableau est un enregistrement, contenant plusieurs variables de type différent.
- On accède à un enregistrement par **son indice** dans le tableau.
- **groupe[4]** représente *la quatrième* personne du groupe
- **groupe[4].nom** représente **le nom** de la quatrième personne du groupe.
- **groupe[4].age** représente **l'âgé** de la quatrième personne du group



3.5 L'instruction avec faire

Pour ***simplifier*** l'écriture et éviter l'utilisation de la notion nom_variable.champ , nous pouvons utiliser l'accès **avec ….faire**

```
Type Tpersonne = record
        Age: integer;
        nom: String[20];
   end;

Var pers1: Tpersonne;
```

```
1- Accès direct
Pers1.age := 24;
Pers1.nom := 'Ali';

2- Accès avec l'instruction « With »
With pers1 do
   age := 24;
   nom := 'ALI';
End;
```

Exercices sur les enregistrements

Exercice 1 :

Le principe d'un type énuméré est de donner un nom symbolique pour chacune des valeurs que peut prendre une variable. Exemple :

Type Mois = (JANVIER, FÉVRIER, ..., DÉCEMBRE)

Question : Cette déclaration est-elle correcte? Expliquez

Exercice 2 :

Créer des types intervalles heure, minute et seconde, puis un type temps_t.qui comporte ces intervalles.

Exercice 3 :

Déclarer des nouveaux types ou structure qui permettent de stocker :

- Un joueur de basket qui se caractérise par son nom, sa date de naissance, sa nationalité, et sa taille.
- Une association de N joueurs de basket

Exercice 4 :

Créer un tableau qui contient 120 étudiants d'une section donnée. Chaque étudiant s'identifié par le nom, le prénom, la moyenne, ses notes concernant 9 modules ainsi que le résultat (admis, ajourné).

Exercice 5 :

Un des services médicaux dans un hôpital étatique décide d'automatiser son archive. Pour cela il estime d'enregistrer les fiches de ses patient sur l'ordinateur en s'adaptant uns structure hétérogène. Cette dernière doit comporter les informations suivantes :

- Un identifiant du dossier (entier)
- Un Nom (chaine de 25 caractères) et un prénom (25 caractères)
- Un numéro d'assurance (10 caractères maximum)
- Un numéro de tel (08 caractères maximum)
- Une adresse (Rue, désignation, commune, wilaya=code sur 2 chiffres)
- Description de la maladie
- Date d'entrée et date de sortie(JJ.MM.AA)

Exercice 6 :

Un compte **CCP** concerne un *étudiant* de la fac se spécifié par les données suivantes :

- Nom et prénom de propriétaire de compte (35 caractères au maximum)
- Numéro de compte (clé)
- Montant en lettres
- Montant en chiffres
- Nom et prénom de bénéficiaire (35 au maximum)

Questions :

Q1 : Déclarer le type de cette structure

Q2 : Saisir les informations pour 20 étudiants.

Solution des exercices sur Enregistrements

Exercices avec solutions :

Exercice 1 :

Le principe d'un type énuméré est de donner un nom symbolique pour chacune des valeurs que peut prendre une variable. Exemple :

Type Mois = (JANVIER, FÉVRIER, ..., DÉCEMBRE)

Question : Cette déclaration est-elle correcte? Expliquez

SOLUTION :

Cette déclaration n'est pas correcte parce qu'il faut énumérer toutes les valeurs (les noms symboliques) de la variable Mois.

Exercice 2 :

Créer des types intervalles heure, minute et seconde, puis un type temps_t qui comporte ces intervalles.

SOLUTION :

```
TYPE
   Heure_t = 0..23;
   Minute_t = 0..59;
  Seconde_t = 0..59;
  temps_t = enregistrement
        H : Heure_t;
        M : Minute_t;
        S: seconde_t;
   Fin_ temps_t;
```

Exercice 3 :

Déclarer des nouveaux types ou structure qui permettent de stocker :

- Un joueur de basket qui se caractérise par son nom, sa date de naissance, sa nationalité, et sa taille.
- Une association de N joueurs de basket

SOLUTION :

```
TYPE
      t_date = RECORD
       date = enregistrement
          jour:1..31;
          mois: 1..12;
          an: entier
       fin t_date ;
joueur  _t = enregistrement
          Nom : chaine de caractères [25];
          date : t_date;
          Nationalité: seconde_t;
         Taille : réel ;
 Fin_ temps_t;
Variables
Association : tableau [1..N]de joueur_t ;      % la variable association est  de type tableau %
```

Exercice 4 :

Créer un tableau qui contient 120 étudiants d'une section donnée. Chaque étudiant s'identifié par le nom, le prénom, la moyenne, ses notes concernant 9 modules ainsi que le résultat (admis, ajourné

SOLUTION :

```
Type
   Etud = enregistrement
     Nom,prénom :chaine de caractères
     Notes :tabeau[1 ..9] de réel   % le champ notes est de type tableau %
     Moy-générale : réel
     Resutat :(admis,ajourné)
 Variables
  PV : tableau [1..120] de ETUD
  % exemple de tableau dont les éléments sont des enregistrements%
```

Exercice 5 :

Un des services médicaux dans un hôpital étatique décide d'automatiser son archive. Pour cela il estime d'enregistrer les fiches de ses patient sur l'ordinateur en s'adaptant uns structure hétérogène. Cette dernière doit comporter les informations suivantes :

- Un identifiant du dossier (entier)
- Un Nom (chaine de 25 caractères) et un prénom (25 caractères)
- Un numéro d'assurance (10 caractères maximum)
- Un numéro de tel (14 caractères maximum)
- Une adresse (Rue, désignation, commune, wilaya=code sur 2 chiffres)
- Description de la maladie
- Date d'entrée et date de sortie(JJ.MM.AA)

SOLUTION :

```
Type
   Adresse_t=enregistrement
      Rue : entier
      Désignation : chaine de caractères
      Commune : chaine de caractères[20]
     Wilaya :  1..48
fin Adresse_t ;
   malade_t = enregistrement
    ident : entier
    Nom,prénom :chaine de caractères [25]
    Assurance : chaine de caractères [10]
    Tel : chaine de caractères [14]
     Adres : Adresse_t
   Description : chaine de caractères
     Date_E,date_S : t_date;     % comme déjà vu  précédemment  dans l'exercice n° 3
fin malade_t ;
```

Solution des exercices sur Enregistrements

Exercice 6 :

Un compte **CCP** concerne un *étudiant* de la fac se spécifié par les données suivantes :

- Nom et prénom de propriétaire de compte (40 caractères au maximum)
- Numéro de compte (clé)
- Montant en lettres
- Montant en chiffres
- Nom et prénom de bénéficiaire (40 au maximum)

Questions :

Q1 : Déclarer le type de cette structure

Q2 : Saisir les informations pour 20 étudiants.

SOLUTION :

Q1 :

```
Type
   etudiant_t = enregistrement
   num_cpt : entier
   Nomp : chaine de caractères [40]
   Mantant_l : chaine de caractères
   Mantant_c : réel
   Nom_benf : chaine de  caractères [40]
fin etudiant_t ;
```

Q2 :

```
Variables
Etud : etudiant_t
i :entier
Debut
 Pour i allant de 1 à 20 Faire
    Avec Etud faire
        Ecrire ('entrer les informations concernant le ', i, 'étudiant') ;
         Ecrire ( 'numero de compte :')
          Lire (num_cpt)
         Ecrire ( 'nom et prénom  de propriétaire :')
          Lire (Nomp)
         Ecrire ( 'mantant en lettres :')
          Lire (Mantant_l )
        Ecrire ( 'mantant en chiffres :')
          Lire (Mantant_c)
        Ecrire ('nom et prénom  de bénéficiaire :')
          Lire (Nom_benf )
     Fin avec
 Fin pour
```

Nb :

- Dans cette solution, on a utilisé l'instruction **avec** … **faire** pour éviter la répétition de l'écriture du variable **etud**
- on peut avoir une deuxième solution sans l'utilisation de **Avec .. faire**.
- Dans ce cas, on *supprime* l'instruction *avec …faire* et on *se base* sur la notion **nom de Variable point nom de champ**
- On ***remplacera*** donc chaque *lecture d'un champ* par **etud.champ** exemple :

Lire (num_cpt) sera remplacé par **Lire (Etud. num_cpt) ,** Lire **(**Mantant_l **)** sera **Lire(Etud.Nomp),** lire**(**Mantant_l **)**sera lire**(Etud.mantant_l) ……..Etc.**

Sommaire

Série 1 : Type Tableau (Vecteurs et Matrices)

Exercice 1 : Lecture et Affichage d'un vecteur

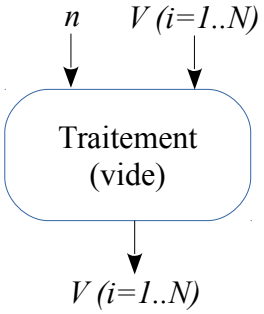
Écrire un algorithme/programme PASCAL qui permet de lire et afficher un vecteur V de N composantes réelles.

Solution

L'algorithme

```
Algorithme exercie_1
     Variables
          V : Tableau [1..100] de Réel
          i,n : entier
Début
     Lire(n)
     Pour i←1 à n faire
          Lire( V[i] )
     Fin-Pour

     Pour i←1 à n faire
          Écrire( V[i] )
     Fin-Pour
Fin
```



Le programme PASCAL

```
01 Program exercie_1;
02 Uses wincrt ;
03 var
04     V : array[1..100] of Real;
05     i, n : integer ;
06 Begin
07     {Les entrées}
08     Write('Donner la taille du vecteur : ');
09     Read(n);
10     Writeln('Donner les composantes du vecteur : ');
11     For i:=1 to n do
12          Read( V[i] );
13
14     {Les sorties}
15     Writeln('Affichage du Vecteur : ');
16     For i:=1 to n do
17          Write( V[i]:10:3 );{Afficher sur 10 caractères et}
18                             {2 chiffres après la virgule}
19 End.
```

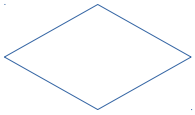
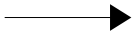
Explication

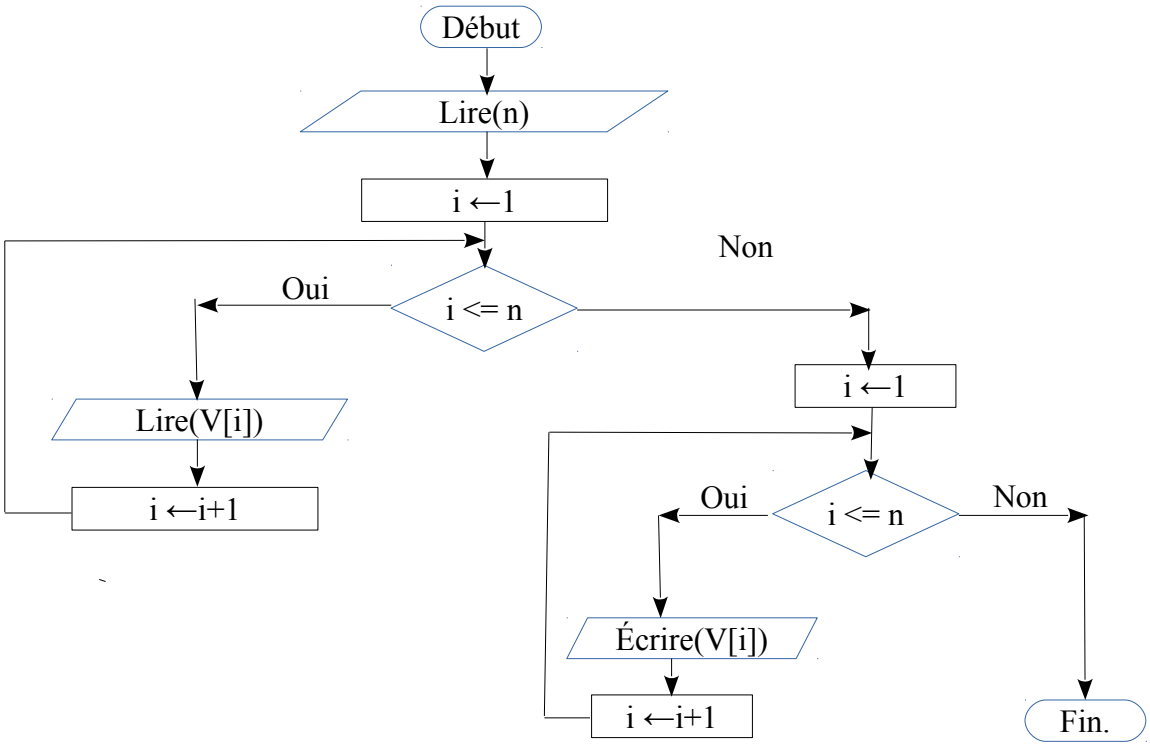
Ce programme montre comment réaliser la lecture et l'écriture d'un vecteur (une séquence de cases mémoire contigüe de même type). Pour les deux opérations (lecture et écriture) on aura besoin toujours d'une boucle **Pour**. Lors de la déclaration, on réserve 100 cases réelles (le taille maximale du vecteur), et on utilise la variable **n** pour déterminer la taille qu'on veut utiliser (par exemple 5 cases). L'accès à une composante d'indice **i** se fait comment suit : **V[i]**. Ainsi, pour réaliser la lecture de la case 2, on écrira **Lire(V[2])**, et **Écrire(T[4])** pour afficher la valeur de la 4ème case.

Il faut noter que ce programme ne réalise pas de traitement, il continent uniquement des entrées et des sorties.

Organigramme

L'organigramme est une façon de montrer un algorithme (ou un programme) sous forme d'actions ***schématisées*** et leurs enchaînement (***flèches***)

Les différentes figures d'organigramme	
Figure	***Sémantique / Sense***
	Représente le début et la Fin de l'organigramme
	Entrées / Sorties : Lecture des données et écriture des résultats.
	Calculs, Traitements
	Tests et décision : on écrit le test à l'intérieur du losange
	Ordre d'exécution des opérations (Enchaînement)
	Connecteur



Exercice 2 : Somme et Moyenne des éléments d'un vecteur

Écrire un algorithme/un programme PASCAL qui permet Calculer la somme et la moyenne des éléments d'un vecteur V réel de taille N.

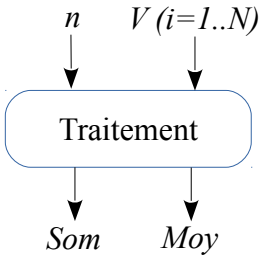
Solution

L'algorithme

```
Algorithme exercie_2
     Variables
          V : Tableau [1..100] de Réel
          i,n : entier
          Som, Moy : Réel
Début
     Lire(n)
     Pour i←1 à n faire
          Lire( V[i] )
     Fin-Pour

     Som ← 0
     Pour i←1 à n faire
          Som ← Som + V[i]
     Fin-Pour
     Moy ← Som / n

     Ecrire(Som, Moy)
Fin
```



Le programme PASCAL

```
01 Program exercie_2;
02 Uses wincrt ;
03 var
04     V : array[1..100] of Real;
05     i, n : integer ;   Som, Moy : real;
06 Begin
07     {Les entrées}
08     Write('Donner la taille du vecteur : ');
09     Read(n);
10     Writeln('Donner les composantes du vecteur : ');
11     For i:=1 to n do
12          Read( V[i] );
13
14     {Les traitements}
15     Som:=0; {Som = V[1] + V[2] + V[3] + … + V[n]}
16     For i:=1 to n do
17          Som:= Som + V[i];
18
19     Moy:=Som/n; {La moyenne égale à la somme sur le nombre d'éléments}
20
21     {Les sorties}
22     Writeln('La somme est : ', Som:8:3);
23     Writeln('La moyenne est : ', Moy:8:3);
24 End.
```

Explication

Soit V un vecteur de n éléments réels, le calcul de la somme de ces éléments se fait par la formule mathématique : V[1]+V[2]+...+V[n]. Si on met l'identificateur Som pour la variable qui représente la somme de ces cases, donc on aura :

$$\text{Som} = V[1]+V[2]+...+V[n] = \sum_{i=1}^{n} V[i]$$

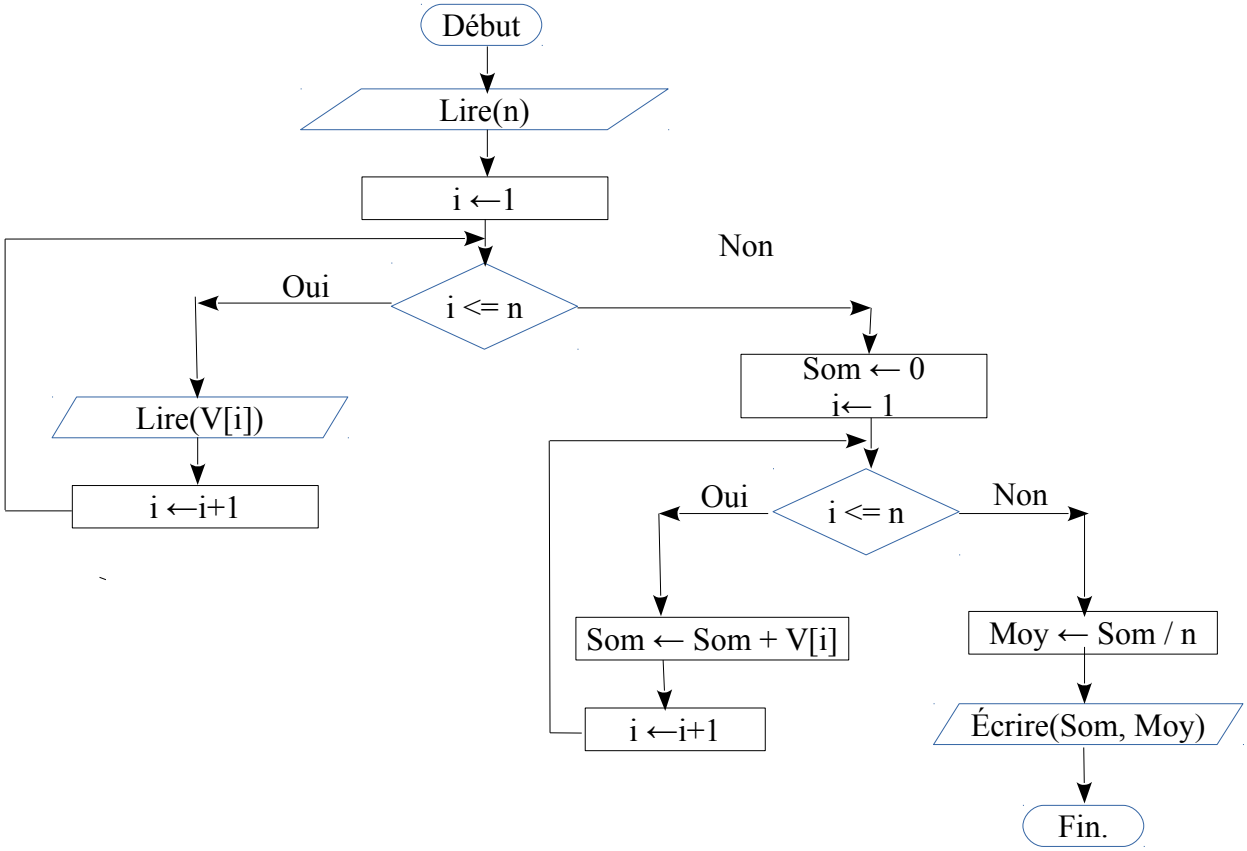
On a vu, auparavant, que le symbole de la somme sera remplacer par un boucle **Pour**, avec le terme répétitif, pour cette somme, égale à V[i], ce qui donne :

Pour i← 1 **à** n **faire**

Som ← Som + V[i]

Remarquer que les bornes de la sommes (i=1 à n) sont les mêmes bornes de la boucle Pour (i←1 à n). Et pour la moyenne, on sait que : Moyenne = Somme / le nombre d'éléments sommés.

Organigramme



Exercice 3 : L'inverse des éléments d'un vecteur

1 Écrire un algorithme/programme PASCAL qui permet d'inverser les éléments d'un vecteur de type réel T dans un autre vecteur V.

2 Réaliser la même opération dans le même vecteur T (sans utiliser le vecteur V).

Solution

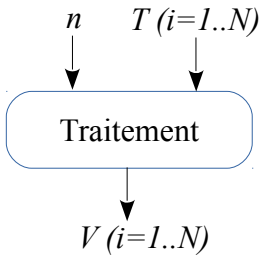
1- Inverser les éléments du vecteur T dans le vecteur V

L'algorithme

```
Algorithme exo3_1
      Variables
            T,V : Tableau [1..100] de Réel
            i,n : entier
Début
      Lire(n)
      Pour i←1 à n faire
            Lire( T[i] )
      Fin-Pour

      Pour i←1 à n faire
            V[i] ← T[n-i+1]
      Fin-Pour


      Pour i←1 à n faire
            Écrire( V[i] )
      Fin-Pour
Fin
```



Le programme PASCAL

```
01 Program exo3_1;
02 Uses wincrt ;
03 var
04      T, V : array[1..100] of Real;
05      i, n : integer ;
06 Begin
07      {Les entrées}
08      Write('Donner la taille du vecteur T : ');
09      Read(n);
10      Writeln('Donner les composantes du vecteur T : ');
11      For i:=1 to n do
12            Read( T[i] );
13
14      {Les traitements}
15      For i:=1 to n do
16            V[i] := T[n-i+1];
17
18      {Les sorties}
19      Writeln('L''affichage du vecteur V : ');
20      For i:=1 to n do
21            Write( V[i] :10:2);
22
23 End.
```

Explication

Mettre les éléments du vecteur T dans un autre vecteur V, mais dans les sens inverse, c'est à dire :

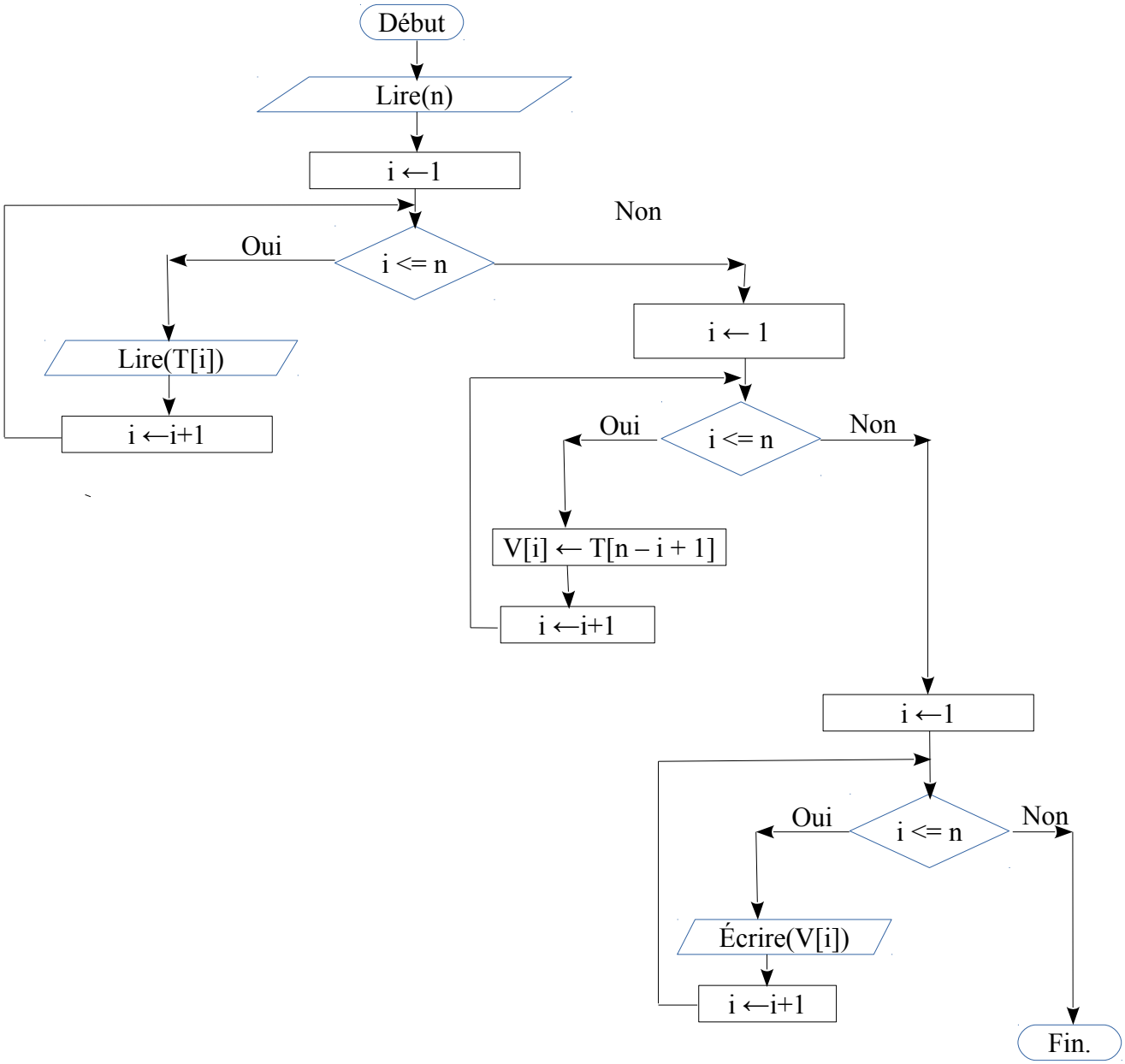
V[1] ← T[n] ; V[2] ← T[n-1] ; V[3] ← T[n-2] ; …….. ; V[n] ← T[1] ;

La correspondance entre les indices de V et ceux de T sont commet suit :

1 ⟷ n – 0 2 ⟷ n – 1 3 ⟷ n – 2 4 ⟷ n – 3 ...

Si on généralise par rapport à l'indice i, on aura : i ⟷ n – (i-1), donc aura l'affectation suivante : Pour toute valeur *i* entre ***1*** et ***n***, *V[i] ← T[n-i+1]*

Organigramme



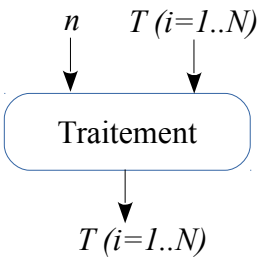
2- Inverser les éléments du vecteur T dans lui même (sans utiliser un autre vecteur)

L'algorithme

```
Algorithme exo3_2
        Variables
                T : Tableau [1..100] de Réel
                i,n : entier
                Z:Réel
Début
        Lire(n)
        Pour i←1 à n faire
                Lire( T[i] )
        Fin-Pour

        Pour i←1 à (n div 2) faire
                Z ← T[i]
                T[i] ← T[n-i+1]
                T[n-i+1] ← Z
        Fin-Pour


        Pour i←1 à n faire
                Écrire( T[i] )
        Fin-Pour
Fin
```



Le programme PASCAL

```
01 Program exo3_2;
02 Uses wincrt ;
03 var
04      T : array[1..100] of Real;
05      i, n : integer ;  Z:real;
06 Begin
07      {Les entrées}
08      Write('Donner la taille du vecteur T : ');
09      Read(n);
10      Writeln('Donner les composantes du vecteur T : ');
11      For i:=1 to n do
12              Read( T[i] );
13
14      {Les traitements}
15      For i:=1 to (n div 2) do {Inverser par permutation de cases}
16       Begin                   {La case i est permutée avec la case n-i+1}
17              Z := T[i];       {Conserver T[i] dans Z}
18              T[i] := T[n-i+1]; {On peut écraser T[i] par T[n-i+1]}
19              T[n-i+1] := Z;    {Maintenant on met Z dans T[n-i+1]}
20       end;
21
22      {Les sorties}
23      Writeln('L''affichage du vecteur T : ');
24      For i:=1 to n do
25              Write( T[i] :10:2);
26
30 End.
```

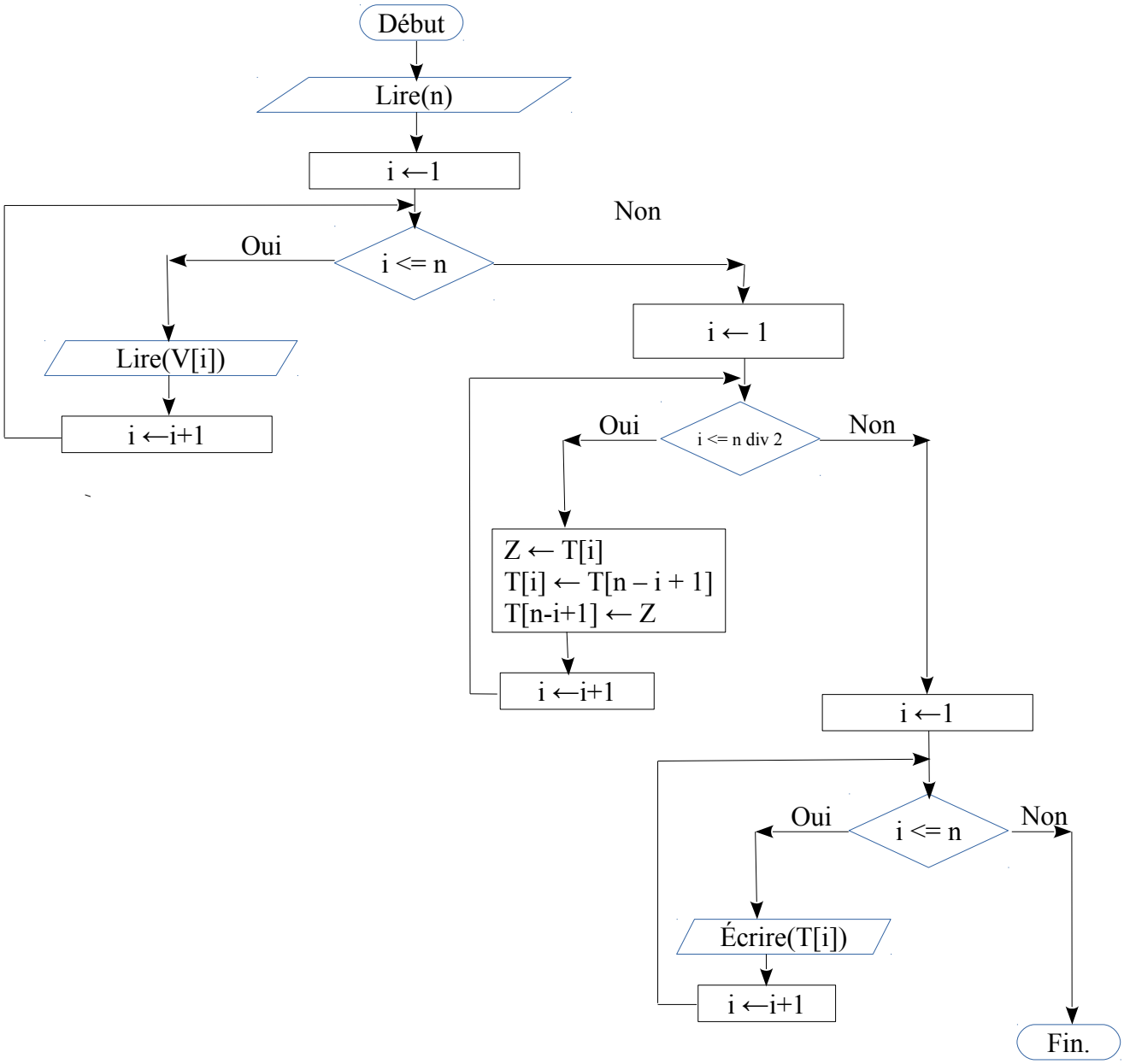
Explication

La solution qui vient immédiatement à l'esprit est une suite d'affectations : T[i] ← T[n-i+1]. Le problème ici est que la valeur de T[i] sera écrasée par T[n-i+1]. Donc ce cas, il faut penser à réaliser une permutation entre les deux cases : *i* et *n-i+1*. Et ce ci avec les trois affectations :

Z ← T[i] ; T[i] ← T[n-i+1] ; T[n-i+1] ← Z ; (comme la permutation entre x et y). Il faut faire attention, il faut réaliser cette permutation entre un case de la première moitié du vecteur avec la

case correspondante de la deuxième moitié du vecteur, donc il faut arrêter les permutations au milieu du vecteur. Ce qui veut dire que le nombre de permutations à réaliser un vecteur de taille *n* est (*n div 2*). (pour n=5, on aura 2 permutations possibles 1 avec 5 et 2 avec 4) (pour n=6, on aura 3 permutations possibles : 1 avec 6, 2 avec 5 et 3 avec 4).

Organigramme



Exercice 4 : Min et le Max d'un vecteur et leurs positions

1 Écrire un algorithme/programme PASCAL qui permet de rechercher le plus petit élément dans un vecteur réel V ainsi que sa position.

2 Écrire un programme PASCAL qui permet de rechercher le plus grand élément dans un vecteur réel V ainsi que sa position.

Solution

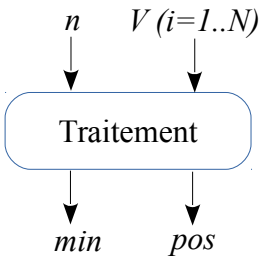
1- Recherche du min et sa position dans vecteur

L'algorithme

```
Algorithme exo4_1_Min
      Variables
            V : Tableau [1..100] de Réel
            i,n, Pos : entier   Min:Réel
Début
      Lire(n)
      Pour i←1 à n faire
            Lire( V[i] )
      Fin-Pour

      Min ← V[1]
      Pos ← 1
      Pour i←1 à n faire
            Si T[i] < Min alors
               Min ←  V[i]
               Pos ← i
            Fin-Si
      Fin-Pour


      Écrire( Min, Pos)
Fin
```



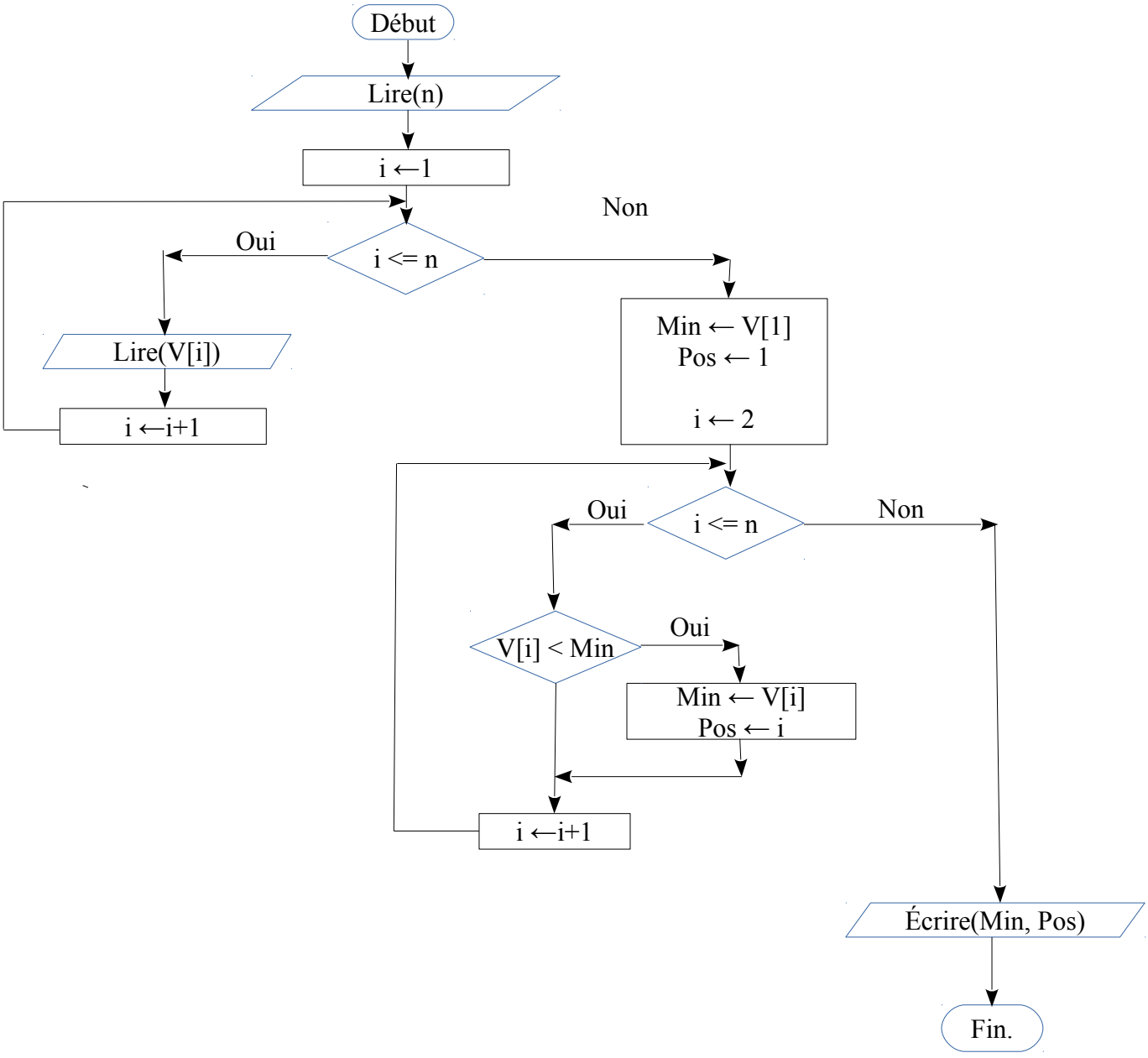
Le programme PASCAL

```
01 Program exo4_1_Min;
02 Uses wincrt ;
03 var
04      T, V : array[1..100] of Real;
05      i, n, Pos : integer ;  Min:Real;
06 Begin
07      {Les entrées}
08      Write('Donner la taille du vecteur V : ');
09      Read(n);
10      Writeln('Donner les composantes du vecteur V : ');
11      For i:=1 to n do
12            Read( V[i] );
13
14      {Les traitements}
15      Min:=V[1]; Pos:=1;  {On suppose que le Min est la valeur de la première case}
16      For i:=2 to n do    {On parcours toutes les autres cases}
17            If V[i] < Min then {Pour chaque case qui est inférieur au Min}
18             begin
19               Min := V[i]; {On actualise le Min (corriger le Min) }
20               Pos := i;    {On actualise sa position}
21             end;
22
23      {Les sorties}
24      Write('Le Min du V est : ',Min:2:2,' et sa position est : ', Pos);
25 End.
```

Explication

La solution de la recherche du min dans un vecteur ainsi que sa position commence par la supposition que le premier élément du vecteur est le minimum (la ligne 15 du programme précédent : *Min:=V[1] ; Pos:=1;*). Par la suite, on parcours toutes les autres cases de l'indice *2* jusqu'à l'indice *n* pour vérifier s'il y parmi ces cases celles qui vérifiassent la condition *V[i]<Min*, pour chaque élément *V[i]* qui vérifie la condition précédente, on met à jour le *Min* par *V[i]* et la valeur de *Pos* par *i*.

Organigramme



Déroulement

Déroulement du programme (ou l'algorithme) pour *n=6* et *V= [5, 12, -1, 2, -8, 10]*.

Les valeur de *n* et *V* données précédemment correspondent aux entrées du programme (c'est à dire les lignes du code n° *08* à *12*). Le déroulement se fait pour la partie des traitements (n° de ligne entre 15 et 21).

Instructions	***Variables du Programme***				
	n	***i***	***V***	***Min***	***Pos***
Entrées (N° de ligne : 08-12)	5	/	[5, 12, -1, 2, -8, 10]	/	/
Min := V[1] ; Pos := 1 ;	"	/		5	1
For i :=2 if V[2]<Min (12 < 5)→ False (on n'entre pas au test **If**)	"	2			
For i :=3 if V[3]<Min (-1 < 5)→ True (on entre au test **If**) Min := V[i] ; (Min ← -1) Pos := i; (Pos ← 3)	"	3	"	-1	3
For i :=4 if V[4]<Min (2 < -1)→ False (on n'entre pas au test **If**)	"	4	"	"	"
For i :=5 if V[5]<Min (-8 < 5)→ True (on entre au test **If**) Min := V[i] ; (Min ← -8) Pos := i; (Pos ← 5)	"	5	"	-8	5
For i :=6 if V[6]<Min (10 < -8)→ False (on n'entre pas au test **If**) Arrêter la boucle **For** (car *i* = *n*)	"	6	"	"	"
Write(Min, Pos)	"	"	"	-8	5

Donc le min est **-8** et sa position est la case N° **5**.

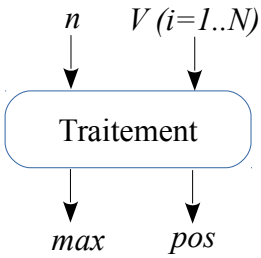
1- Recherche du max et sa position dans vecteur

L'algorithme

```
Algorithme exo4_2_Max
      Variables
            V : Tableau [1..100] de Réel
            i,n, Pos : entier   Max:Réel
Début
      Lire(n)
      Pour i←1 à n faire
                  Lire( V[i] )
      Fin-Pour

      Max ← V[1]
      Pos ← 1
      Pour i←2 à n faire
            Si T[i] > Max alors
               Max ←  V[i]
               Pos ← i
            Fin-Si
      Fin-Pour


      Écrire( Max, Pos)
Fin
```



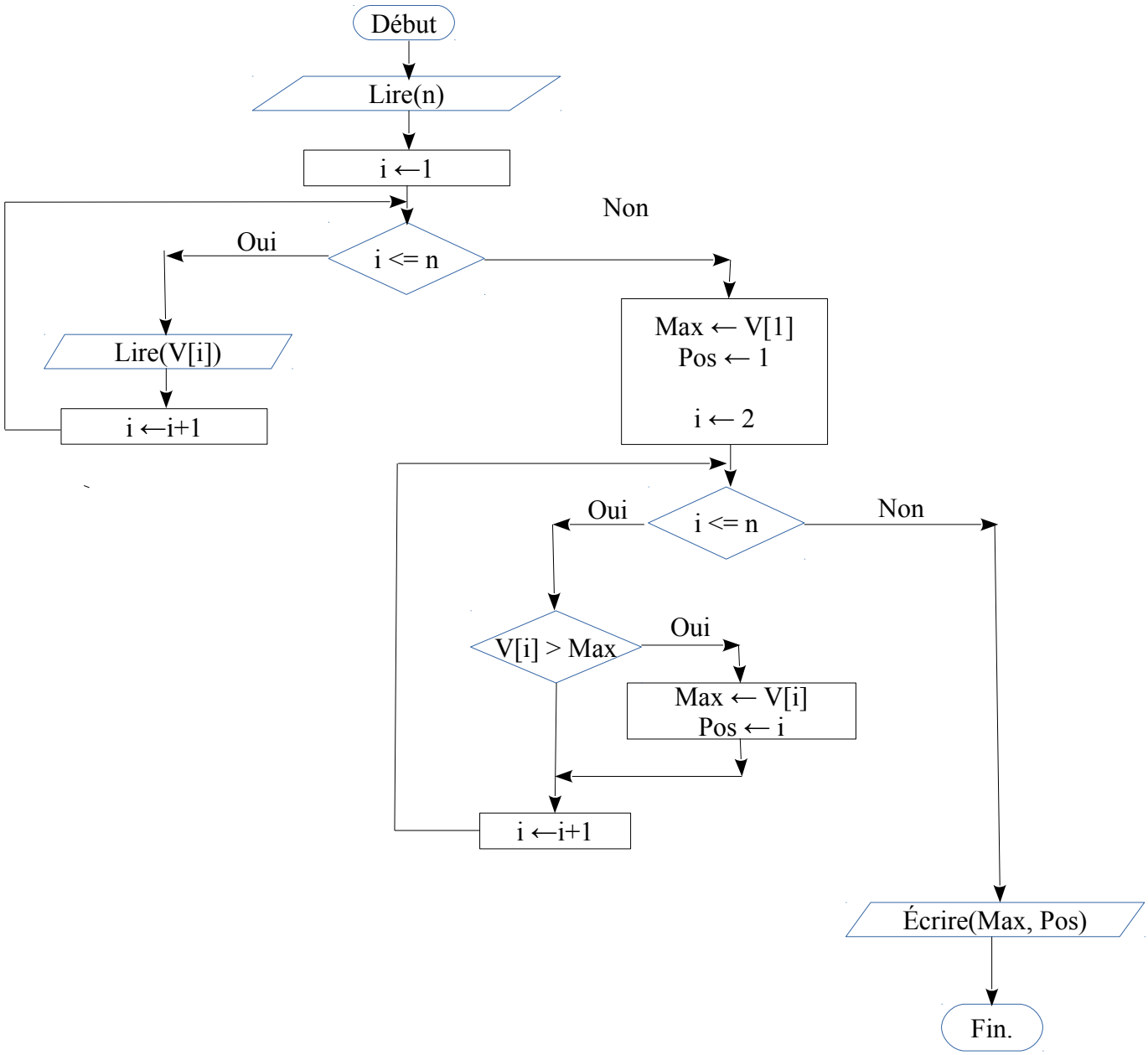
Le programme PASCAL

```
01 Program exo4_2_Max;
02 Uses wincrt ;
03 var
04      T, V : array[1..100] of Real;
05      i, n, Pos : integer ;  Max:Real;
06 Begin
07      {Les entrées}
08      Write('Donner la taille du vecteur V : ');
09      Read(n);
10      Writeln('Donner les composantes du vecteur V : ');
11      For i:=1 to n do
12                  Read( V[i] );
13
14      {Les traitements}
15      Max:=V[1]; Pos:=1; {On suppose que le Max est la valeur de la première case}
16      For i:=2 to n do    {On parcours toutes les autres cases}
17            If V[i] > Max then {Pour chaque case qui est supérieur au Max}
18             begin
19               Max := V[i]; {On actualise le Max (corriger le Max) }
20               Pos := i;    {On actualise sa position}
21             end;
22
23      {Les sorties}
24      Write('Le Max du V est : ',Max:2:2,' et sa position est : ', Pos);
25 End.
```

Explication

Le Même principe que la recherche du Min. Juste la condition qui est modifiée (*V[i] > Max*)

Organigramme



Déroulement

Déroulement du programme (ou l'algorithme) pour *n=6* et *V= [5, 12, -1, 2, -8, 10]*.

Les valeur de *n* et *V* données précédemment correspondent aux entrées du programme (c'est à dire les lignes du code n° *08* à *12*). Le déroulement se fait pour la partie des traitements (n° de ligne entre 15 et 21).

Instructions	***Variables du Programme***				
	n	***i***	***V***	***Max***	***Pos***
Entrées (N° de ligne : 08-12)	5	/	[5, 12, -1, 2, -8, 10]	/	/
Max := V[1] ; Pos := 1 ;	"	/		5	1
For i :=2 if V[2]>Max (12 > 5)→ True (on entre au test **If**) Max := V[i] ; (Max ← 12) Pos := i; (Pos ← 2)	"	2		12	2
For i :=3 if V[3]]>Max (-1 > 12)→ False (on n'entre pas au test **If**)	"	3	"	"	"
For i :=4 if V[4]]>Max (2 > 12)→ False (on n'entre pas au test **If**)	"	4	"	"	"
For i :=5 if V[5]<Min (-8 > 12)→ False (on n'entre pas au test **If**)	"	5	"	"	"
For i :=6 if V[6]>Max (10 > 12)→ False (on n'entre pas au test **If**) Arrêter la boucle **For** (car *i* = *n*)	"	6	"	"	"
Write(Max, Pos)	"	"	"	12	2

Donc le max est **12** et sa position est la case N° **2**.

Exercice 5 : La recherche d'une valeur dans un vecteur

Soit V un vecteur de type réel de taille N. Écrire un algorithme/programme PASCAL qui permet de rechercher si une valeur réelle x existe ou non dans le vecteur V. Dans le cas ou x existe dans V, on affiche aussi sa position.

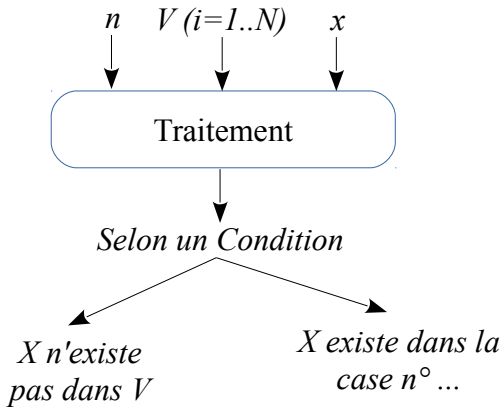
Solution ***Solution 01 : utiliser un booléen Trouve***

L'algorithme

```
Algorithme exo5_a
      Variables
            V : Tableau [1..100] de Réel
            i,n, Pos : entier   x:Réel
            Trouve : booléen
Début
  Lire(n)
  Pour i←1 à n faire
    Lire( V[i] )
  Fin-Pour
  Lire(x)

  Trouve ← False     i ← 1
  Tantque (i<=n) ET (Trouve = False) Faire
    Si T[i] = x alors
      Trouve ←  True
      Pos ← i
    Fin-Si
    i ← i+1
  Fin-Tantque

  Si Trouve = True Alors
    Écrire('X existe dans la position : ', Pos)
  Sinon
    Écrire('X n''existe pas dans V')
  Fin-Si
Fin
```



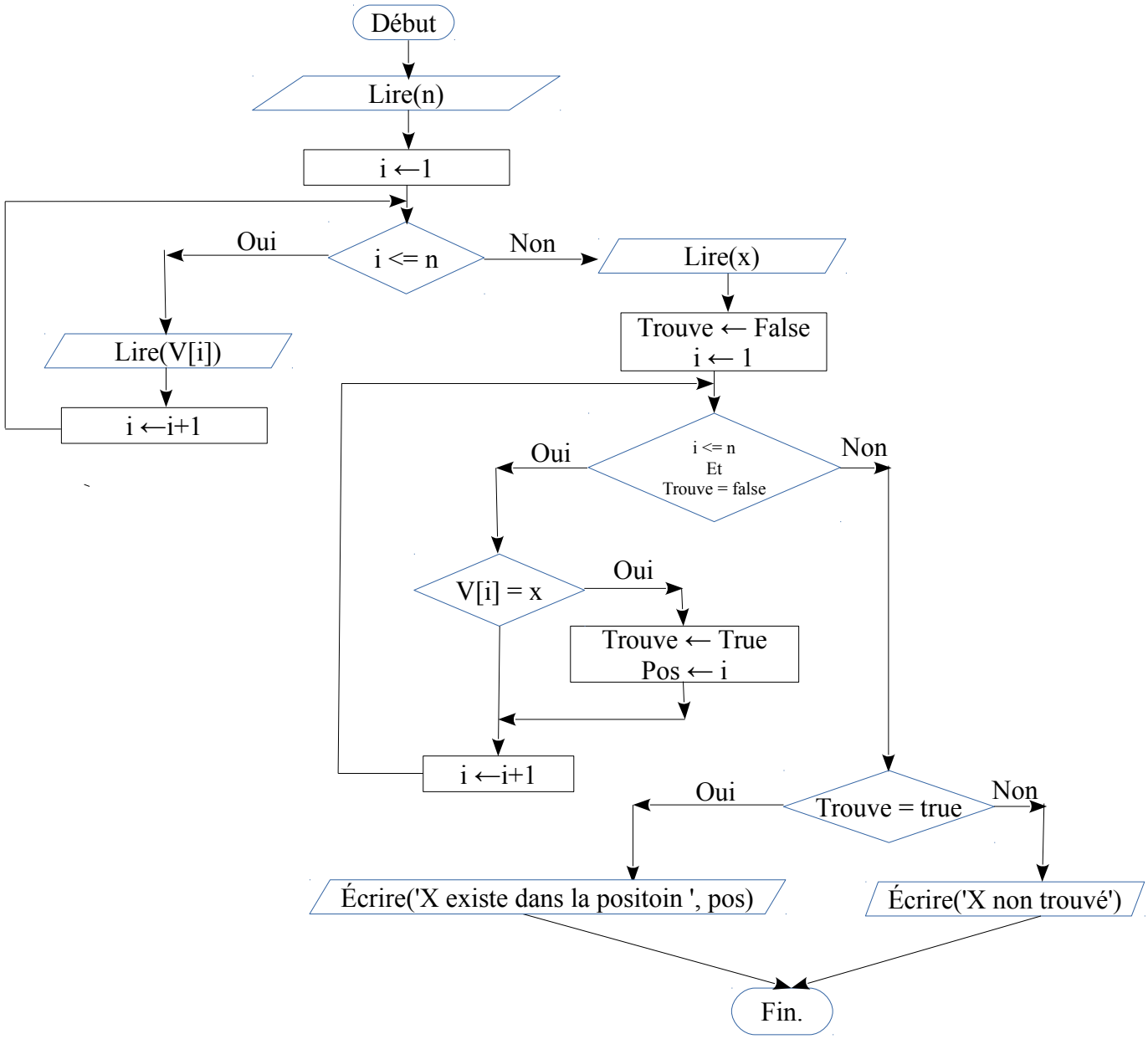
Le programme PASCAL

```
01 Program exo5_a;
02 Uses wincrt ;
03 var
04      T, V : array[1..100] of Real;
05      i, n, Pos : integer ;  x:Real; Trouve:boolean;
06 Begin
07      {Les entrées}
08      Write('Donner la taille du vecteur V : ');
09      Read(n);
10      Writeln('Donner les composantes du vecteur V : ');
11      For i:=1 to n do
12                 Read( V[i] );
13      Write('Donner la valeur de x : '); Read(x);
14
15      {Les traitements}
16      Trouve:=false; i:=1; {Initialiser Trouve à false}
17      While (i<=n) AND (Trouve = false) do
18       begin
19            If V[i]=x then {Si une case égale à x}
20             begin
21                Trouve:=true; Pos:=i;
22             End;
23            i := i+1;
24       end;
25
26      {Les sorties}
27      If Trouve=true Then
28         Write('x existe dans le vecteur et sa positon est : ', Pos)
29      Else
30         Write('x n''existe pas dans le vecteur');
31 End.
```

Explication

Les entrées du programme sont le vecteur V et la valeur de x à rechercher. Il y a une boucle de recherche qui permet de parcourir toutes les case de 1 jusqu'à n et elle s'arrête dans deux cas : Soit V[i]=x (donc on trouve la valeur x et on continue pas la recherche), soit i>n (on a parcouru toutes les case sans trouver aucun valeur). La variable booléenne Trouve représente l'existence de x dans V ou non : Si Trouve = True, donc on a trouvé x sinon on l'a pas trouve (ou on a pas encore trouvé x). La boucle de recherche **`Tantque`** `(i<=n)` **`ET`** `(Trouve = False)` **`Faire`** signifie : Tantque on a pas terminé le tableau (i<=n) et toujours on pas encore trouve x (Trouve = False) donc on entre à la boucle, une teste permet de vérifie la case actuelle si est égale à x (Si V[i]=x Alors), donc ce cas on sauvegarde la position (qui est i : pos ← i) et on indique qu'on a trouvé la valeur x (Trouve←true), ensuite, on se met à la case suivante : i←i+1, et ainsi de suite.

Organigramme



Déroulement

1- Déroulement du programme (ou l'algorithme) pour *n=6* et *V= [5, 12, -1, 2, -8, 10]*. et *x=2*

Instructions	***Variables du Programme***						
	n	***i***	***V***	***x***	***Trouve***	***Pos***	***Réponse (résultat)***
Entrées (N° de ligne : 08-13)	6	/	[5, 12, -1, 2, -8, 10]	2	/	/	
Trouve := false ; i:=1 ;	"	1		"	False	"	

While (i<=n)AND(Trouve=false)
(**1**<=6)And(false=false)
→ True (on entre à While)
If (V[**1**] = x) (5=2) → False
(On n'entre pas à If)
i:= i+1 ; (i=2)
	"	**2**		"	"	"	

While (i<=n)AND(Trouve=false)
(**2**<=6)And(false=false)
→ True (on entre à While)
If (V[**2**] = x) (12=2) → False
(On n'entre pas à If)
i:= i+1 ; (i=3)
	"	**3**	"	"	"	"	

While (i<=n)AND(Trouve=false)
(**3**<=6)And(false=false)
→ True (on entre à While)
If (V[**3**] = x) (-1=2) → False
(On n'entre pas à If)
i:= i+1 ; (i=4)
	"	**4**	"	"	"	"	

While (i<=n)AND(Trouve=false)
(**4**<=6)And(false=false)
→ True (on entre à While)
If (V[**4**] = x) (2=2) → True
(On entre à If)
Trouve := true
Pos:= i (Pos=4)
i:= i+1 ; (i=5)
	"	**5**	"	"	**True**	**4**	

While (i<=n)AND(Trouve=false) (**5**<=6)And(true=false) True And False→ False (on n'entre pas à While)	"		"	"	"	"	
If Trouve = true → True (on entre à If) Write('x existe dans ', pos)	"		"	"	"	"	X existe dans la position 4

Donc le programme affichera le résultat : **X existe dans le vecteur et sa position est : 4**

1- Déroulement du programme (ou l'algorithme) pour *n=6* et *V= [5, 12, -1, 2, -8, 10].* et *x=55*

Instructions	***Variables du Programme***						
	n	***i***	***V***	***x***	***Trouve***	***Pos***	***Réponse (résultat)***
Entrées (N° de ligne : 08-13)	6	/	[5, 12, -1, 2, -8, 10]	5	/	/	
Trouve := false ; i:=1 ;	"	1		"	**False**	"	

While (i<=n)AND(Trouve=false)
(**1**<=6)And(false=false)
→ True (on entre à While)
If (V[**1**] = x) (5=55) → False
(On n'entre pas à If)
i:= i+1 ; (i=2)
	"	**2**		"	"	"	

While (i<=n)AND(Trouve=false)
(**2**<=6)And(false=false)
→ True (on entre à While)
If (V[**2**] = x) (12=55) → False
(On n'entre pas à If)
i:= i+1 ; (i=3)
	"	**3**	"	"	"	"	

While (i<=n)AND(Trouve=false)
(**3**<=6)And(false=false)
→ True (on entre à While)
If (V[**3**] = x) (-1=55) → False
(On n'entre pas à If)
i:= i+1 ; (i=4)
	"	**4**	"	"	"	"	

While (i<=n)AND(Trouve=false)
(**4**<=6)And(false=false)
→ True (on entre à While)
If (V[**4**] = x) (2=55) → False
(On n'entre pas à If)
i:= i+1 ; (i=5)
	"	**5**	"	"	"	""	

While (i<=n)AND(Trouve=false)
(**5**<=6)And(false=false)
→ True (on entre à While)
If (V[**5**] = x) (-8=55) → False
(On n'entre pas à If)
i:= i+1 ; (i=6)
	"	**6**	"	"	"	"	

While (i<=n)AND(Trouve=false)
(**6**<=6)And(false=false)
→ True (on entre à While)
If (V[**5**] = x) (10=55) → False
(On n'entre pas à If)
i:= i+1 ; (i=7)
		7	"	"	"	"	

While (i<=n)AND(Trouve=false) (**7**<=6)And(false=false) False And True → False (on n'entre pas à While)		"	"	"	"	"	
If Trouve = true → Fase (on n'entre pas à If → On entre à Else) Write('x n''existe pas dans ')	"		"	"			X n'existe pas dans le vecteur

Donc le programme affichera le résultat : **X n'existe pas dans le vecteur**.

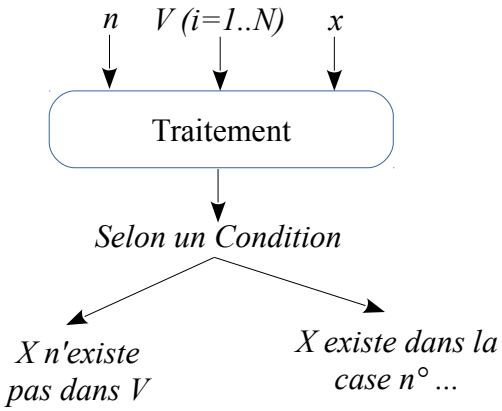
Solution 02 : utiliser l'indice i (selon la valeur de i)

L'algorithme

```
Algorithme exo5_b
      Variables
            V : Tableau [1..100] de Réel
            i,n: entier   x:Réel
Début
  Lire(n)
  Pour i←1 à n faire
    Lire( V[i] )
  Fin-Pour
  Lire(x)

  i ← 1
  Tantque (i <= n) ET (T[i] <> x) Faire
    i ← i+1
  Fin-Tantque

  Si i<=n Alors
    Écrire('X existe dans la position : ', i)
  Sinon
    Écrire('X n''existe pas dans V')
  Fin-Si
Fin
```



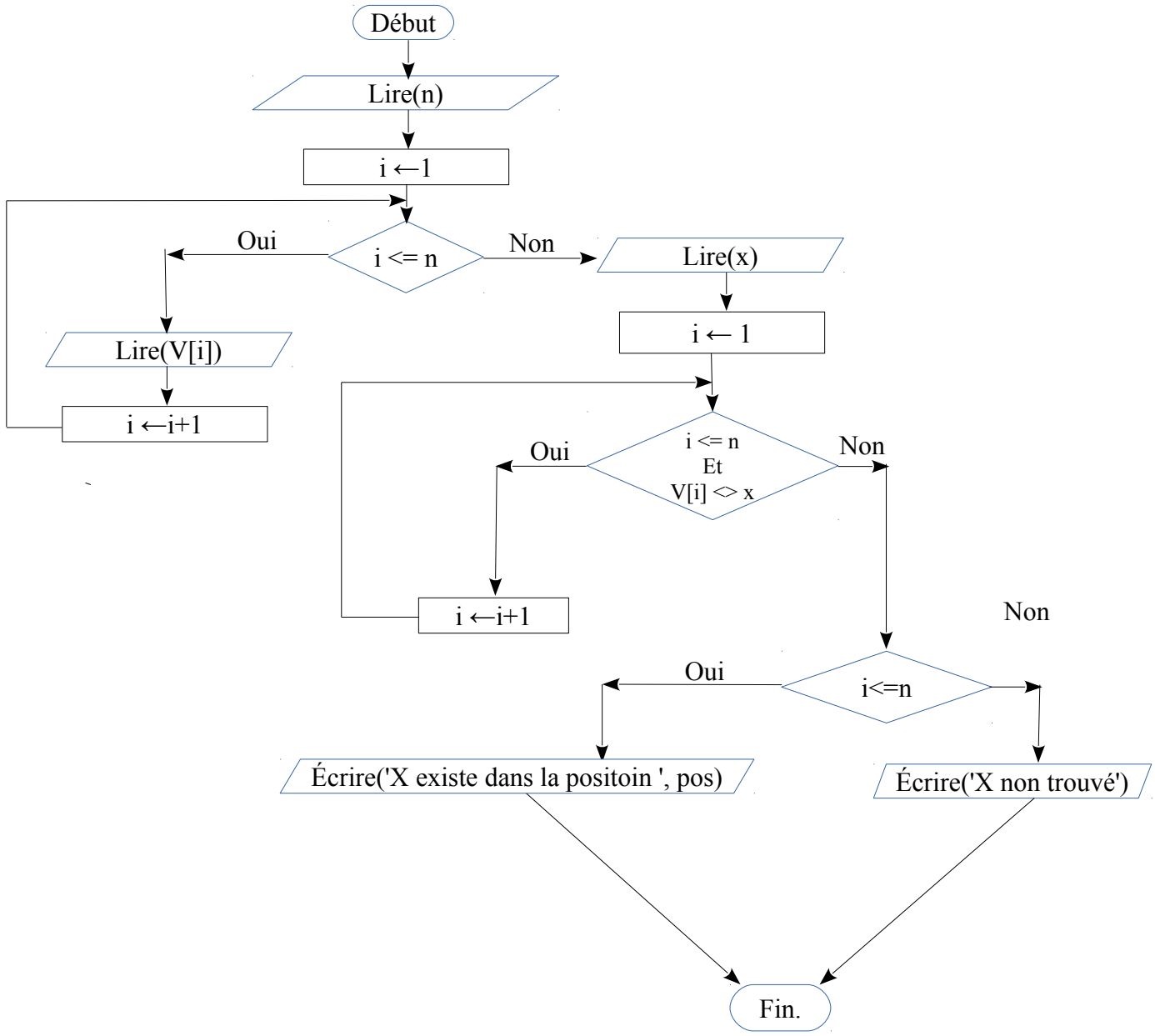
Le programme PASCAL

```
01  Program exo5_b;
02  Uses wincrt ;
03  var
04       T, V : array[1..100] of Real;
05       i, n : integer ;  x:Real;
06  Begin
07       {Les entrées}
08       Write('Donner la taille du vecteur V : ');
09       Read(n);
10       Writeln('Donner les composantes du vecteur V : ');
11       For i:=1 to n do
12                 Read( V[i] );
13       Write('Donner la valeur de x : '); Read(x);
14
15       {Les traitements}
16       i:=1; {Initialiser Trouve à false}
17       While (i<=n) AND (V[i] <> x) do
18          i := i+1;
19
20       {Les sorties}
21       If i <= n Then
22          Write('x existe dans le vecteur et sa positon est : ', i)
23       Else
24          Write('x n''existe pas dans le vecteur');
25  End.
```

Explication

Dans cette solution, la boucle recherche s'arrête lorsque la condition `(i<=n)` **AND** `(V[i]<>x)` est fausse, c-à-d, elle devient `(i>n)` **OR** `(V[i]=x)` (Soit i>n ou bien V[i]=x). Lorsque la boucle est terminée, on aura deux cas par rapport à la valeur de *i* : Soit i>n ou bien i<=n. Dans le cas où i>n, çà veut dire qu'on a parcouru toutes cases du vecteur sans trouver aucun élément qui égale à x, sinon (i<n), ça veut dire que la boucle a été arrêtée à la case i où on trouvé la valeur x, c'est à dire V[i]=x.

Organigramme



Déroulement

1- Déroulement du programme (ou l'algorithme) pour *n=6* et *V= [5, 12, -1, 2, -8, 10]*. et *x=2*

Instructions	***Variables du Programme***				
	n	***i***	***V***	***x***	***Réponse (résultat)***
Entrées (N° de ligne : 08-13)	6	/	[5, 12, -1, 2, -8, 10]	2	
i:=1 ;	"	1		"	
While (**i**<=n)AND(V[**i**]<>x) (**1**<=6)And(5<>2) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=2)	"	**2**	"	"	
While (**i**<=n)AND(V[**i**]<>x) (**2**<=6)And(12<>2) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=3)	"	**3**	"	"	
While (**i**<=n)AND(V[**i**]<>x) (**3**<=6)And(-1<>2) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=4)	"	**4**	"	"	
While (**i**<=n)AND(V[**i**]<>x) (**4**<=6)And(2<>2) (True)And(False)→ False (on n'entre pas à While)	"	"	"	"	
If i<=n (4 ≤ 6) → True (on entre à If) Write('x existe dans ', i)	"		"	"	X existe dans la position 4

Donc le programme affichera le résultat : **X existe dans le vecteur et sa position est : 4**

1- Déroulement du programme (ou l'algorithme) pour *n=6* et *V= [5, 12, -1, 2, -8, 10].* et *x=55*

Instructions	***Variables du Programme***				
	n	***i***	***V***	***x***	***Réponse (résultat)***
Entrées (N° de ligne : 08-13)	6	/	[5, 12, -1, 2, -8, 10]	2	
i:=1 ;	"	1		"	
While (**i**<=n)AND(V[**i**]<>x) (**1**<=6)And(5<>55) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=2)	"	**2**	"	"	
While (**i**<=n)AND(V[**i**]<>x) (**2**<=6)And(12<>55) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=3)	"	**3**	"	"	
While (**i**<=n)AND(V[**i**]<>x) (**3**<=6)And(-1<>55) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=4)	"	**4**	"	"	
While (**i**<=n)AND(V[**i**]<>x) (**4**<=6)And(2<>55) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=5)	"	**5**	"	"	
While (**i**<=n)AND(V[**i**]<>x) (**5**<=6)And(-8<>55) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=6)		**6**			
While (**i**<=n)AND(V[**i**]<>x) (**6**<=6)And(10<>55) (True)And(True)→ True (on entre à While) i:= i+1 ; (i=7)		**7**			
While (**i**<=n)AND(V[**i**]<>x) (**7**<=6)And(0<>2) (False)And(True)→ False (on n'entre pas à While)					
If i<=n (7 ≤ 6) → False (on n'entre pas à If, on entre à Else) Write('x n''existe pas dans le vecteur')	"		"	"	X n'existe pas dans le vecteur

Donc le programme affichera le résultat : **X n'existe pas dans le vecteur**.

Exercice 6 : Lecture et affichage d'une matrice

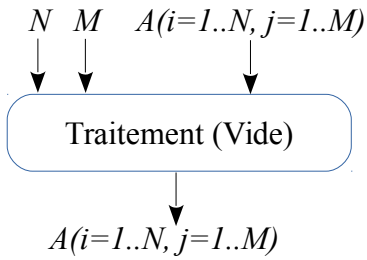
Écrire un algorithme/programme PASCAL qui permet de lire et afficher une matrice de type réel A de N lignes et M colonnes.

Solution ***Solution 01 : utiliser un booléen Trouve***

L'algorithme

```
Algorithme exo6_lecture_ecriture
      Variables
            A:Tableau [1..10,1..10] de Réel
            i,j, N, M : entier
Début
  Lire(N, M)
  Pour i←1 à N faire
    Pour j←1 à M faire
      Lire( A[i,j] )
    Fin-Pour
  Fin-Pour


  Pour i←1 à N faire
    Pour j←1 à M faire
      Écrire( A[i,j] )
    Fin-Pour
  Fin-Pour
Fin
```



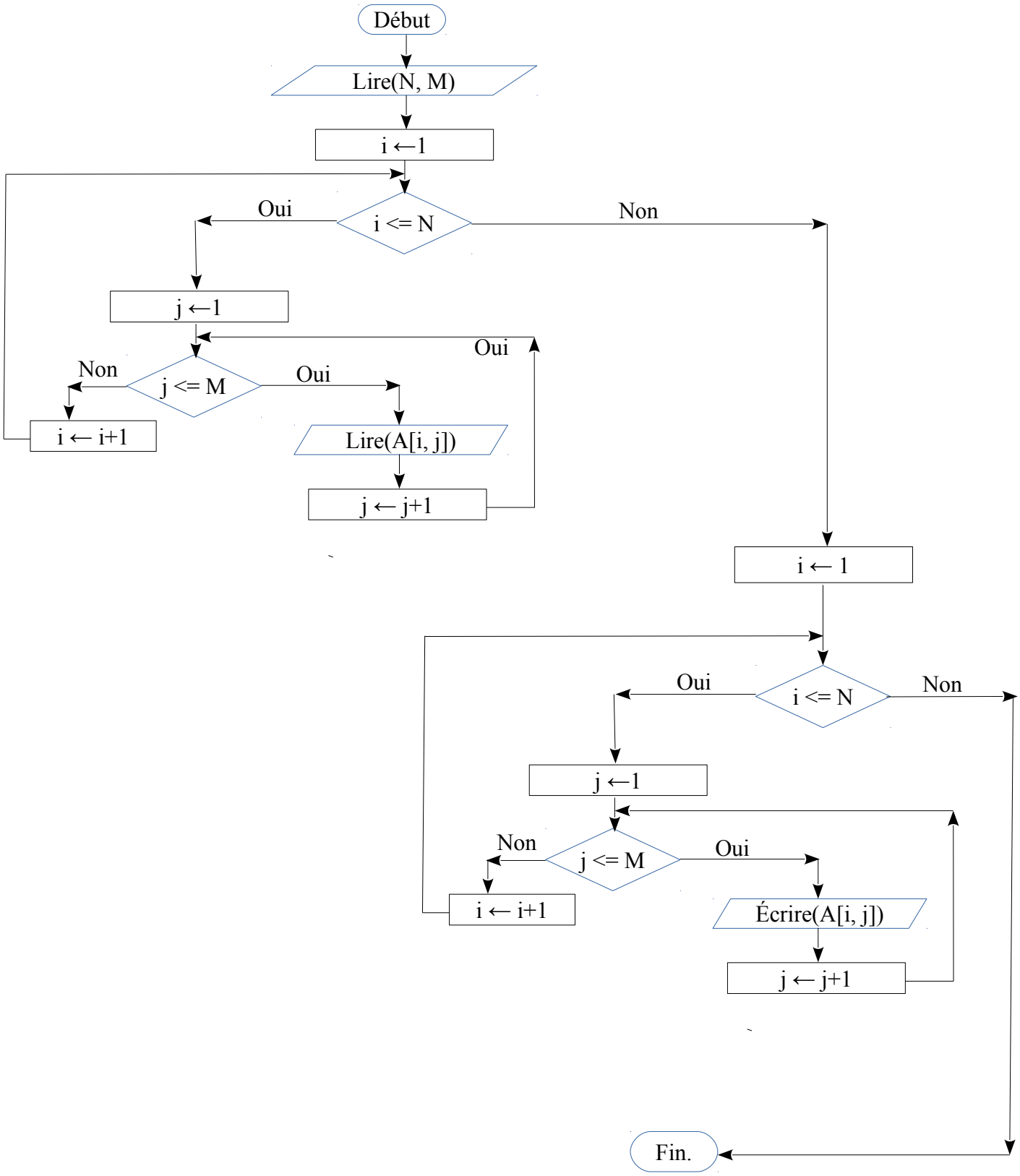
Le programme PASCAL

```
01  Program exo6_lecture_ecriture;
02  Uses wincrt ;
03  var
04       A : array[1..10, 1..10] of Real;
05       i, j, N, M : integer;
06  Begin
07       {Les entrées}
08       Write('Donner les dimensions de la Matrice A : ');
09       Read(N, M);
10       Writeln('Donner les composantes de la matrice A : ');
11       For i:=1 to N do
12            For j:=1 to M do
13                 Read( A[i, j] );
14
15       {Les traitements}
16
17
18       {Les sorties}
19       Writeln('L''affichage de la matrice A : ');
20       For i:=1 to N do
21        begin
22            For j:=1 to M do
23                 Write( V[i]:8:2 );
24
25            Writeln; {Sauter la ligne}
26        end;
27  End.
```

Explication

Ce programme illustre comment déclarer, lire et écrire une matrice d'ordre N*M. Pour lire ou écrire une matrice, on aura besoin de deux boucles imbriqués, la boucle externe pour les lignes (**For** i) et la boucle interne pour les colonnes (**For** j).

Organigramme



Exercice 7 : Somme, Moyenne et Produit des éléments d'une matrice

Soit une matrice A réelle d'ordre NxM.

1 Écrire un algorithme/programme PASCAL qui calcule la somme et la moyenne des éléments de la matrice A.

2 Écrire un algorithme/programme PASCAL qui permet de calculer la somme de chaque ligne et le produit de chaque colonne.

Solution

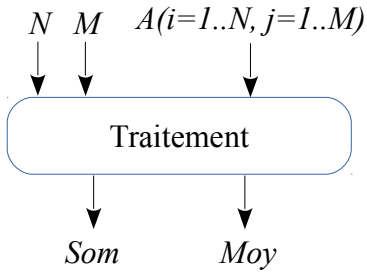
1- Somme et la moyenne des éléments de la matrice A

L'algorithme

```
Algorithme exo7_1
      Variables
            A:Tableau [1..10,1..10] de Réel
            i,j, N, M : entier
            Som, Moy : réel
Début
  Lire(N, M)
  Pour i←1 à N faire
    Pour j←1 à M faire
      Lire( A[i,j] )
    Fin-Pour
  Fin-Pour

  Som ← 0
  Pour i←1 à N faire
    Pour j←1 à M faire
        Som ← Som + A[i, j]
    Fin-Pour
  Fin-Pour
  Moy ← Som / (N*M)

  Écrire (Som, Moy);
Fin
```



Le programme PASCAL

```
01  Program exo7_1;
02  Uses wincrt ;
03  var
04       A : array[1..10, 1..10] of Real;
05       i, j, N, M : integer;
06       Som, Moy:real;
07  Begin
08       {Les entrées}
09       Write('Donner les dimensions de la Matrice A : ');
10       Read(N, M);
11       Writeln('Donner les composantes de la matrice A : ');
12       For i:=1 to N do
13            For j:=1 to M do
14                 Read( A[i, j] );
15
16       {Les traitements}
17       Som:=0; {Initialiser toujours la somme à ZÉRO}
18       For i:=1 to N do
19            For j:=1 to M do
20                 Som := Som + A[i, j];
21
22       Moy := Som / (N*M); {La moyenne égale à la somme sur nombre d'éléments}
23
24       {Les sorties}
25       Write('La somme est : ',Som:2:2,'  La Moyenne est : ',Moy:2:2);
26  End.
27
```

Explication

La somme des éléments d"une matrices est donnée par la formule : A[1, 1]+A[1, 2]+ … + A[1,M]+ A[2, 1]+A[2, 2]+ … + A[2,M]+ … …. … + A[N, 1]+A[N, 2]+ … + A[N,M]. On peut écrire cette formule comme suit :

Som = A[1, 1]+A[1, 2]+ … + A[1,M] //La somme de la première ligne

+ A[2, 1]+A[2, 2]+ … + A[2,M] //La somme de la deuxième ligne

+ A[3, 1]+A[3, 2]+ … + A[3,M] //La somme de la troisième ligne

……

……

+ A[N, 1]+A[N, 2]+ … + A[N,M] //La somme de la dernière ligne

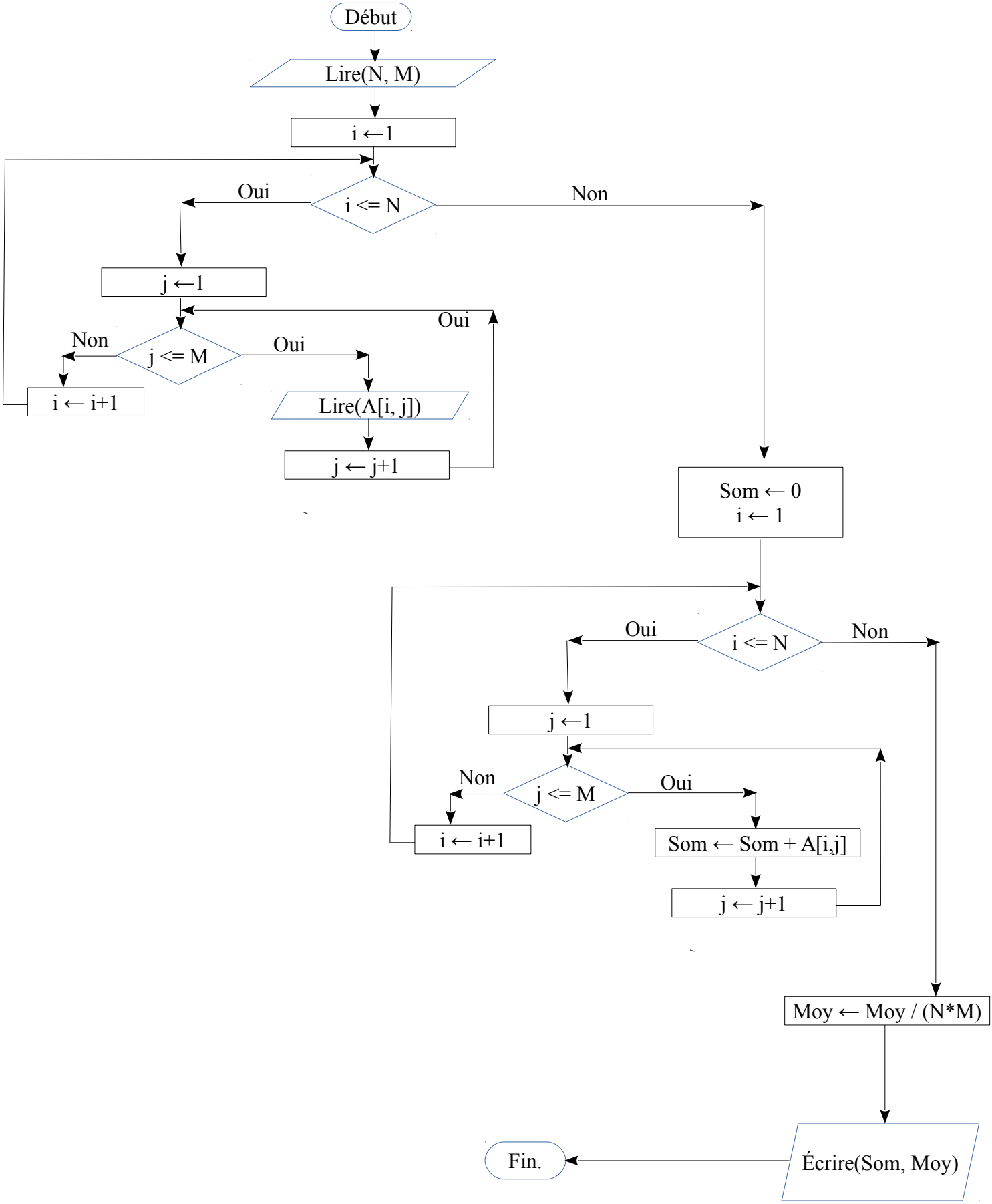
Ça devient :

$$\text{Som} = \sum_{j=1}^{M} A[1,j] + \sum_{j=1}^{M} A[2,j] + \ldots + \sum_{j=1}^{M} A[i,j] + \ldots + \sum_{j=1}^{M} A[N,j]$$

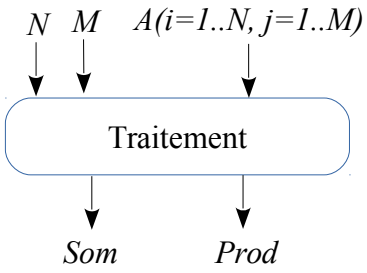
$$\text{Som} = \sum_{i=1}^{N} \sum_{j=1}^{M} A[i,j]$$

Chaque symbole de somme sera replacé par une boucle POUR, donc, on aura deux boucles imbriquées, qui permettent de répéter l'instruction : *Som* ← *Som* + *A[i, j]*

Organigramme



2- Somme de chaque ligne et Produit de chaque colonnes



L'algorithme

```
Algorithme exo7_2
     Variables
          A:Tableau [1..10,1..10] de Réel
          i,j, N, M : entier
          Som, Moy : Tableau[1..10] de Réel
Début
  Lire(N, M)
  Pour i←1 à N faire
    Pour j←1 à M faire
     Lire( A[i,j] )
    Fin-Pour
  Fin-Pour

  Pour i←1 à N faire
    Som[i] ← 0
    Pour j← à M faire
       Som[i] ← Som[i] + A[i, j]
    Fin-Pour
  Fin-Pour

  Pour j←1 à M faire
    Prod[j] ← 1
    Pour i←1 à N faire
       Prod[j] ← Prod[j] * A[i, j]
    Fin-Pour
  Fin-Pour

  Pour i←1 à N faire
    Écrire( Som[i] )
  Fin-Pour

  Pour j←1 à M faire
    Écrire( Prod[j] )
  Fin-Pour
Fin
```

Le programme PASCAL

```
01  Program exo7_2;
02  Uses wincrt ;
03  var
04      A : array[1..10, 1..10] of Real;
05      i, j, N, M : integer;
06      Som, Prod : array[1..10] of Real;
07  Begin
08      {Les entrées}
09      Write('Donner les dimensions de la Matrice A : ');
10      Read(N, M);
11      Writeln('Donner les composantes de la matrice A : ');
12      For i:=1 to N do
13           For j:=1 to M do
14                Read( A[i, j] );
15
16      {Les traitements}
17      For i:=1 to N do
18       begin
19           Som[i] := 0; {Initialiser la somme de la ligne i à 0}
20           For j:=1 to M do
21                Som[i] := Som[i] + A[i, j];
22       end;
23
24      For j:=1 to M do
25       begin
26           Prod[j] := 1; {Initialiser le produit de la colonne j à 1}
27           For i:=1 to N do
28                Prod[j] := Prod[j] * A[i, j];
29       end;
30
31      {Les sorties}
32      Writeln;
33      Writeln('La somme des lignes de la matrice A : ');
34      For i:=1 to N do
35           Write(Som[i]:10:2);
36
37      Writeln;
38      Writeln('Le produit des colonnes de la matrice A : ');
39      For j:=1 to M do
40           Write(Prod[i]:10:2);
41  End.
42
```

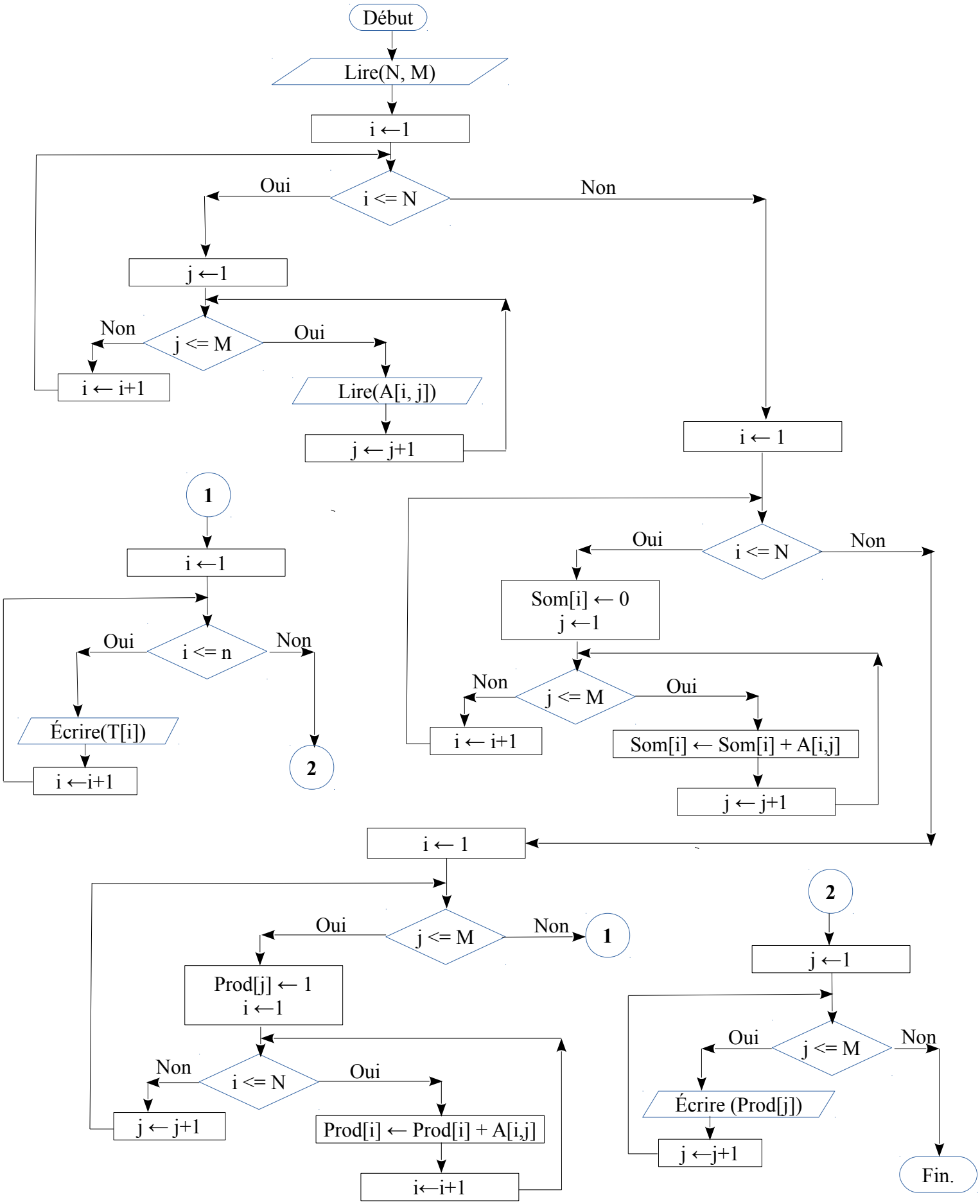
Explication

Pour une matrice A de N lignes et M Colonnes, on aura, pour les lignes N sommes et, pour les colonnes, M Produits. Donc, on déclare les sommes des lignes comme un vecteur *Som* dont la taille est N, et les produits des colonnes comme un vecteur *Prod* dont la taille est M.

Donc on aura, pour chaque ligne i : $Som[i] = \sum_{j=1}^{M} A[i,j]$

Et pour chaque colonne j : Prod[j] = $\prod_{i=1}^{N} A[i,j]$

Organigramme



Exercice 8 : Min et le Max dans une matrice et leurs positions

Soit A une matrice réelle d'ordre NxM.

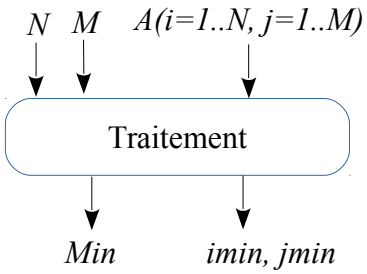
1 Écrire un algorithme/programme PASCAL qui permet de rechercher le plus petit élément dans la matrice A ainsi que sa position.

2 Écrire un algorithme/programme PASCAL qui permet de rechercher le plus grand élément dans la matrice A ainsi que sa position.

Solution

1- Recherche du min et sa position dans une matrice

L'algorithme




```
Algorithme exo8_1_Min
     Variables
          A : Tableau [1..10, 1..10] de Réel
          i, j, N, M, imin, jmin : entier
          Min:Réel
Début
     Lire(N, M)
     Pour i←1 à N faire
       Pour j←1 à M faire
          Lire( A[i, j] )
       Fin-Pour
     Fin-Pour

     Min ← A[1, 1]
     imin ← 1
     jmin ← 1

     Pour i←1 à N faire
       Pour j←1 à M faire
          Si A[i, j] < Min alors
             Min ←  A[i, j]
             imin ← i
             jmin ← j
          Fin-Si
       Fin-Pour
     Fin-Pour


     Écrire( Min, imin, jmin)
Fin
```

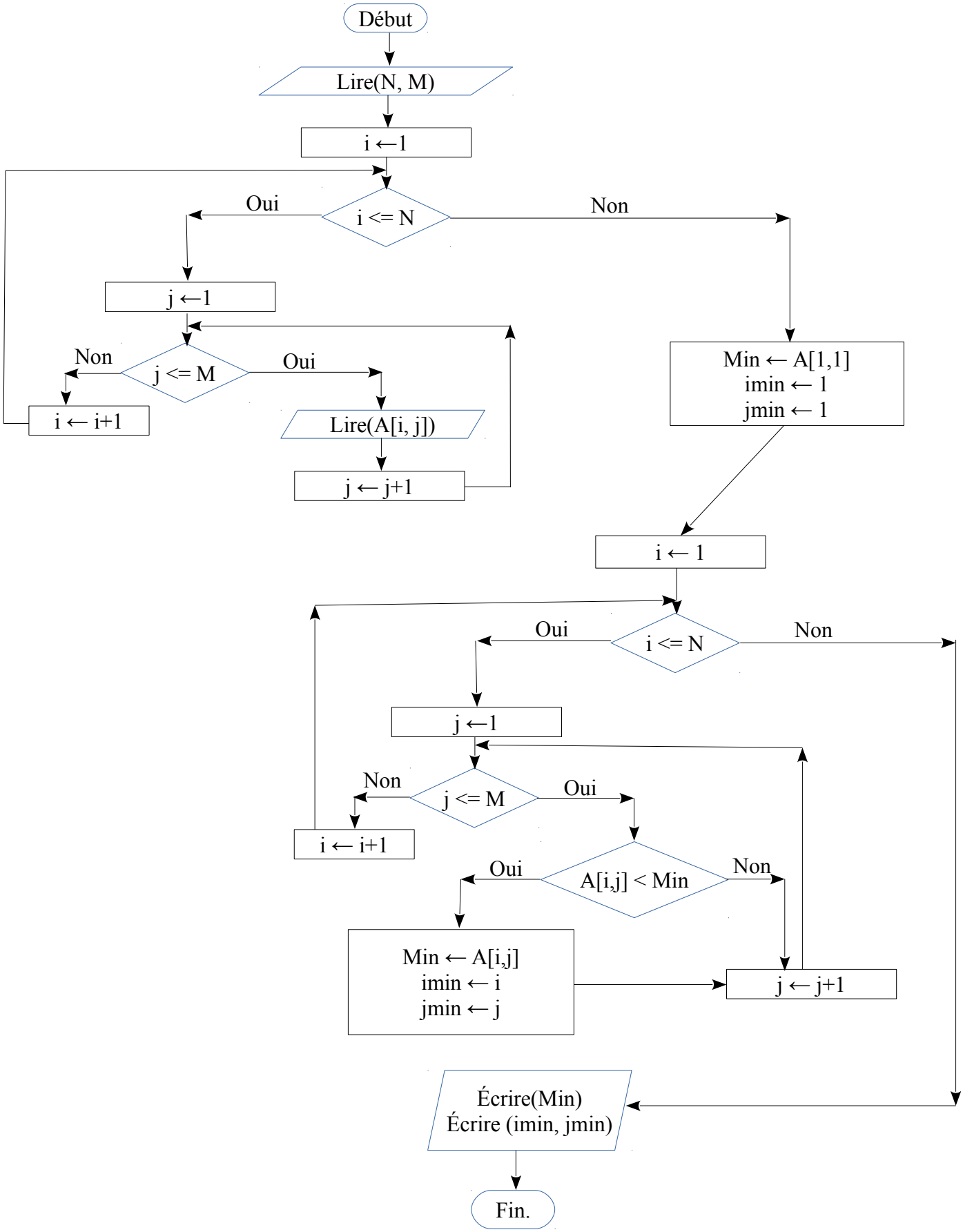
Le programme PASCAL

```
01  Program exo8_1_Min;
02  Uses wincrt ;
03  var
04      A : array[1..10, 1..10] of Real;
05      i, j, N, M, imin, jmin : integer;
06      Min : real;
07  Begin
08      {Les entrées}
09      Write('Donner les dimensions de la Matrice A : ');
10      Read(N, M);
11      Writeln('Donner les composantes de la matrice A : ');
12      For i:=1 to N do
13           For j:=1 to M do
14                Read( A[i, j] );
15
16      {Les traitements}
17      Min := A[1, 1];
18      imin:= 1;
19      jmin := 1;
20
21      For i:=1 to N do
22        For j:=1 to M do
23         if A[i, j] < Min Then
24           begin
25             Min := A[i,j];
26             imin := 1;
27             jmin := 1;
28           end;
29
30      {Les sorties}
31      Write('Min=', Min:2:2, 'Et sa position est : ', imin,' ',jmin);
32  End.
```

Explication

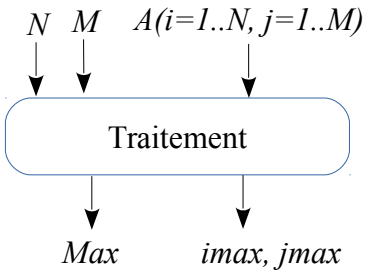
La solution consiste aux parcours des éléments de la matrice pour les comparer un à un à la valeur supposée minimale de la matrice (la ligne N° 23), dés qu'un élément est inférieur au minimum, ce dernier sera mis à jour ainsi que sa position imin (la ligne) et jmin (la colonne) (les lignes N° 25, 26 et 27). La valeur initiale du Min et la première case de la matrice, c'est à dire A[1,1], par conséquent, sa position est : imin=1 et jmin=1.

Organigramme



1- Recherche du max et sa position dans une matrice

L'algorithme




```
Algorithme exo8_2_Max
     Variables
          A : Tableau [1..10, 1..10] de Réel
          i, j, N, M, imax, jmax : entier
          Max:Réel
Début
     Lire(N, M)
     Pour i←1 à N faire
       Pour j←1 à M faire
          Lire( A[i, j] )
       Fin-Pour
     Fin-Pour

     Max ← A[1, 1]
     imax ← 1
     jmax ← 1

     Pour i←1 à N faire
       Pour j←1 à M faire
          Si A[i, j] > Max alors
             Max ←  A[i, j]
             imax ← i
             jmax ← j
          Fin-Si
       Fin-Pour
     Fin-Pour


     Écrire( Max, imax, jmax)
Fin
```

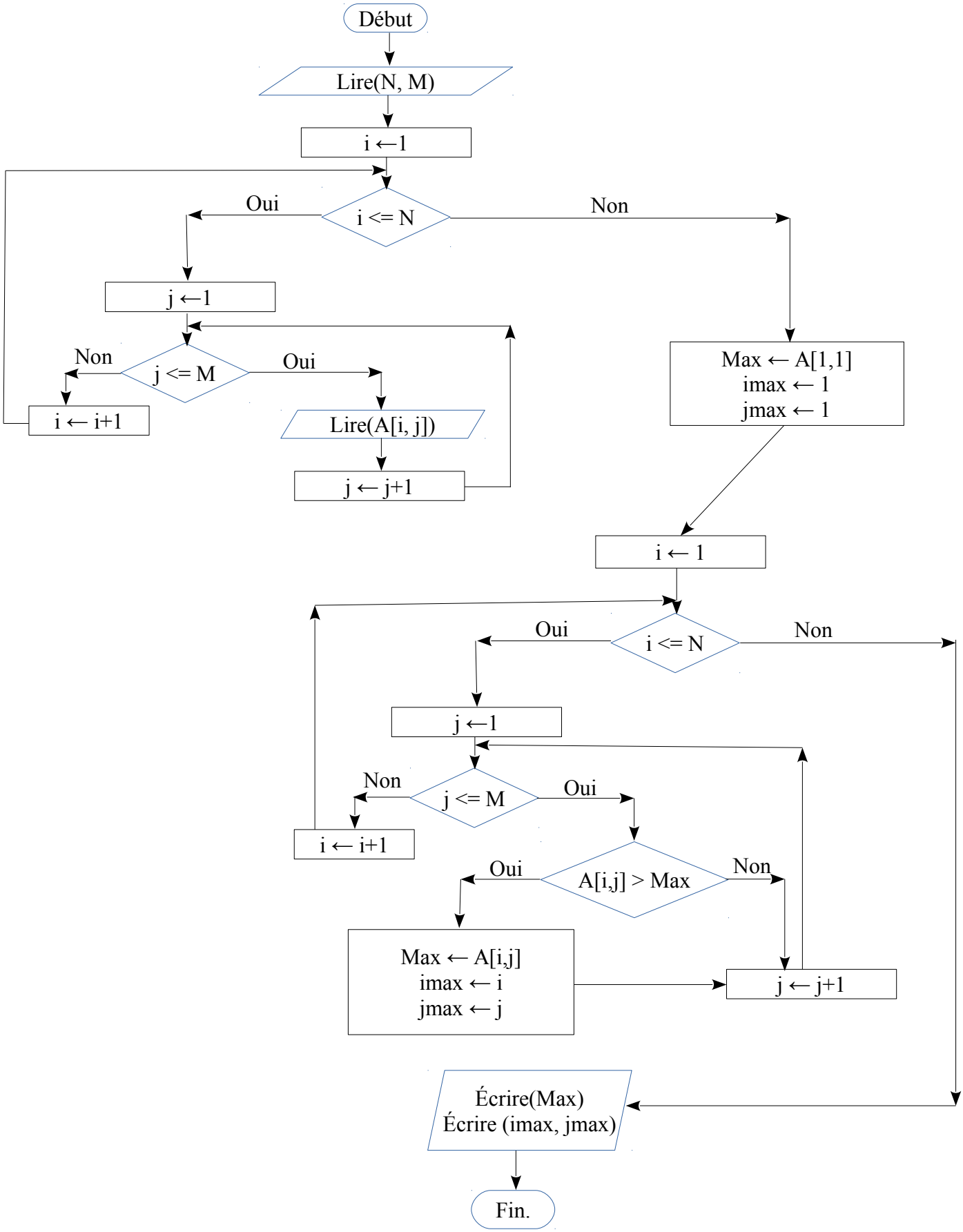
Le programme PASCAL

```
01  Program exo8_2_Max;
02  Uses wincrt ;
03  var
04      A : array[1..10, 1..10] of Real;
05      i, j, N, M, imax, jmax : integer;
06      Max : real;
07  Begin
08      {Les entrées}
09      Write('Donner les dimensions de la Matrice A : ');
10      Read(N, M);
11      Writeln('Donner les composantes de la matrice A : ');
12      For i:=1 to N do
13           For j:=1 to M do
14                Read( A[i, j] );
15
16      {Les traitements}
17      Max := A[1, 1];{On suppose que la première case est la max}
18      imax:= 1; {Donc sa position est la ligne 1}
19      jmax := 1; {et la colonne 1}
20
21      For i:=1 to N do {Pour toute ligne i}
22        For j:=1 to M do {Pour toute colonne j}
23         if A[i, j] > Max Then {Si la case A[i, j] est > au Max}
24           begin
25             Max := A[i,j]; {On actualise le Max}
26             imax := 1; {sa ligne imax}
27             jmax := 1; {sa colonne jmax}
28           end;
29
30      {Les sorties}
31      Write('Max=', Max:2:2, 'Et sa position est : ', imax,' ',jmax);
32  End.
```

Explication

Le même principe que le min et sa position

Organigramme



Exercice 9 : Transposée d'une matrice. Somme de deux matrices

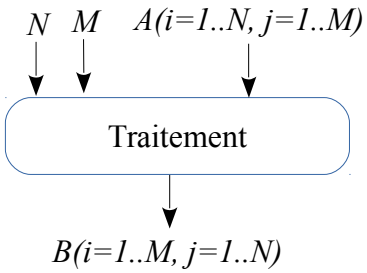
1 Écrire un algorithme/programme PASCAL qui permet de calculer la transposée d'une matrice réelle A d'ordre NxM.

2 Écrire un algorithme/programme PASCAL qui permet de réaliser la somme de matrices réelles A et B d'ordre NxM.

Solution

1- Transposée d'une matrice

L'algorithme



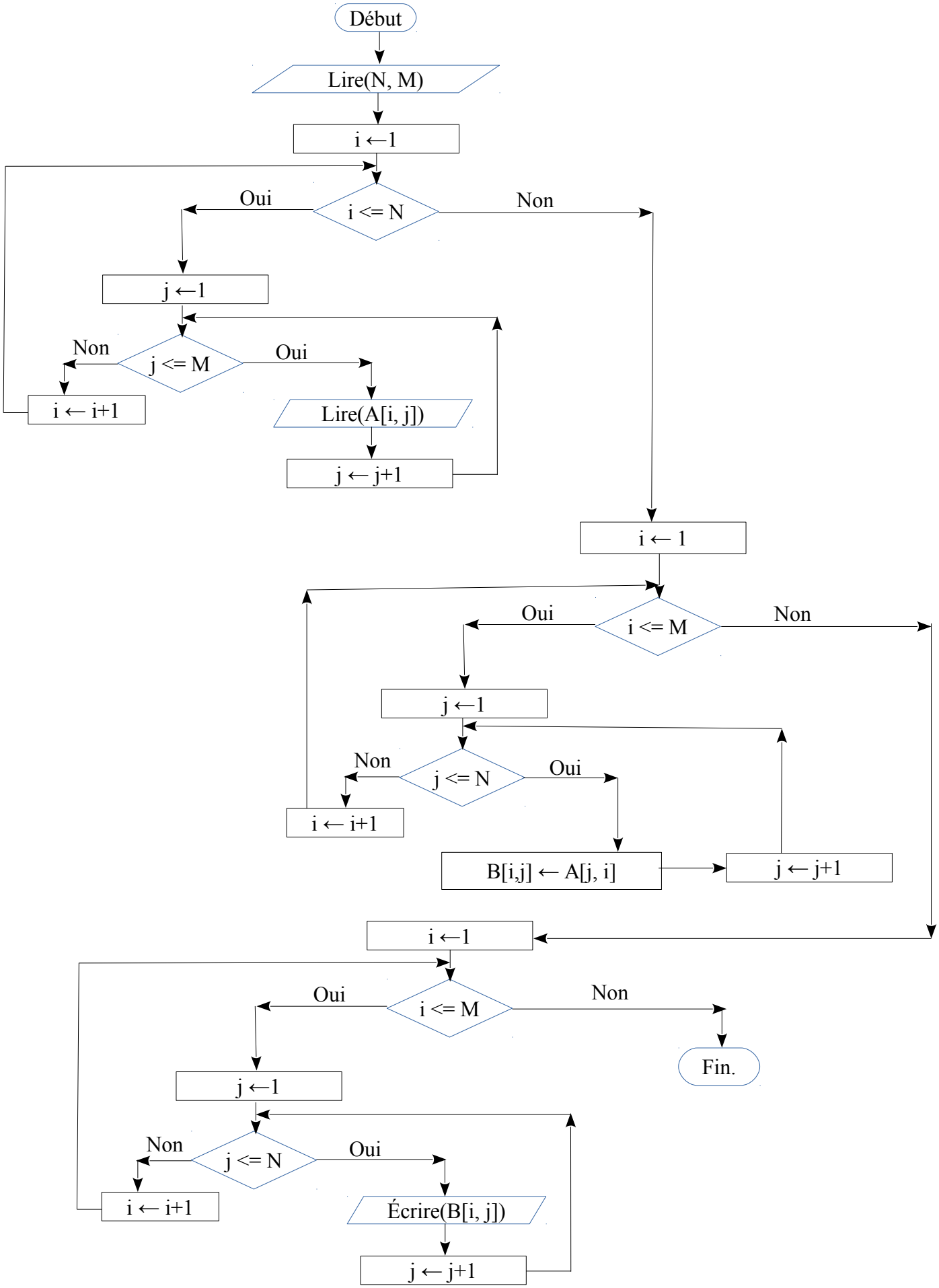
Explication

Le transposé d'une matrice A d'ordre NxM est un matrice B d'ordre MxN. Chaque ligne de A devient une colonne de B (ou chaque colonne de A devient une ligne pour B).

Chaque case B[i, j] correspond à la case A[j, i] tel que : i=1, …, M et j=1, …, N.

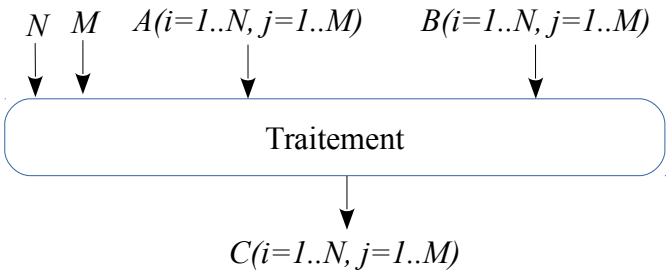
Essayer de faire l'algorithme et le programme PASCAL.

Organigramme



2- Somme de deux matrices

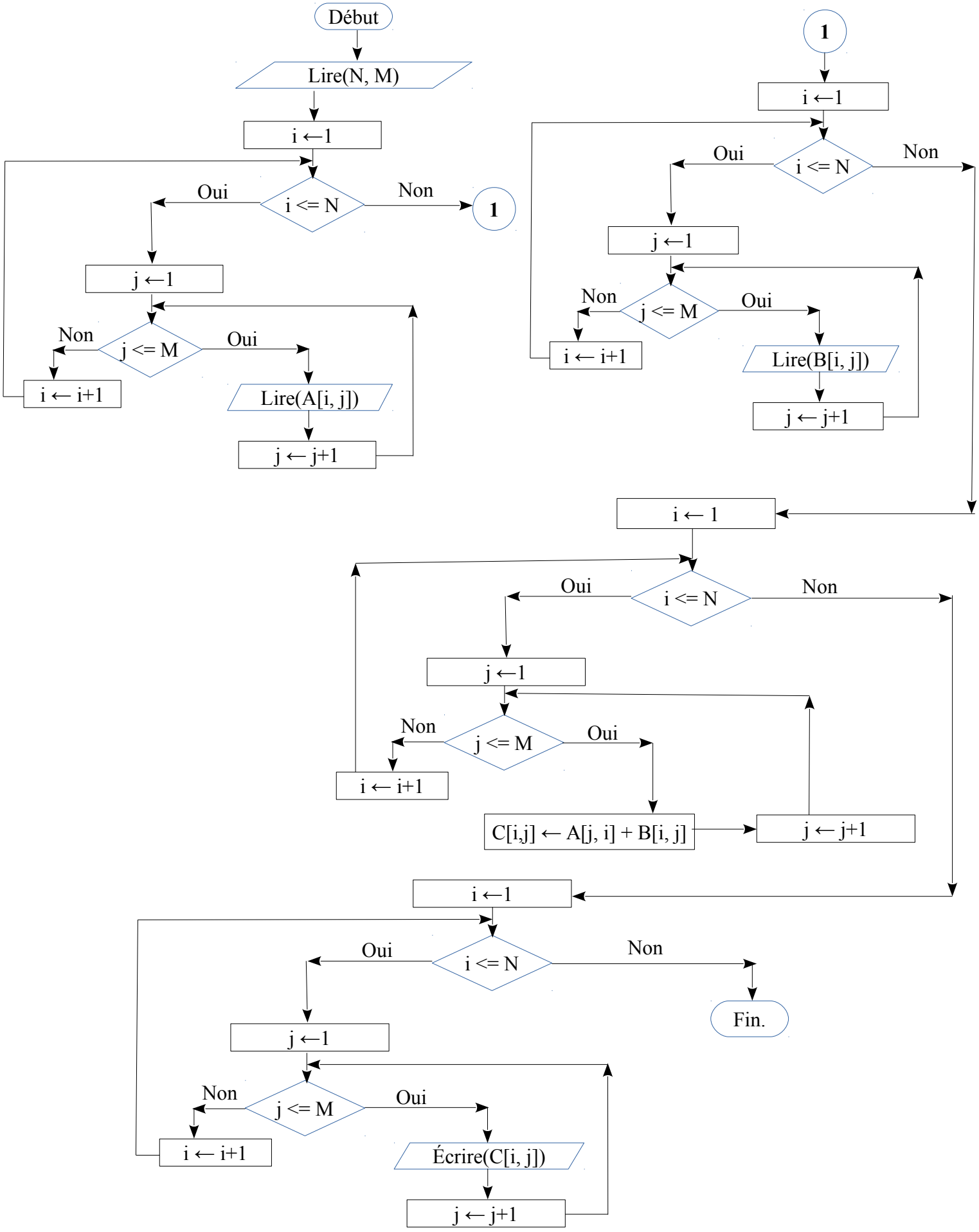
L'algorithme



Explication

Pour réaliser la somme de deux matrices A et B, il faut que ces dernières soient de même dimensions (nombre de lignes et nombre de colonne). La matrice C résultat de la somme est aussi du même dimension que A et B. Chaque case C[i, j] est égale à A[i,j] + B[i, j] (pour i=1, …, N et j = 1… M).

Organigramme

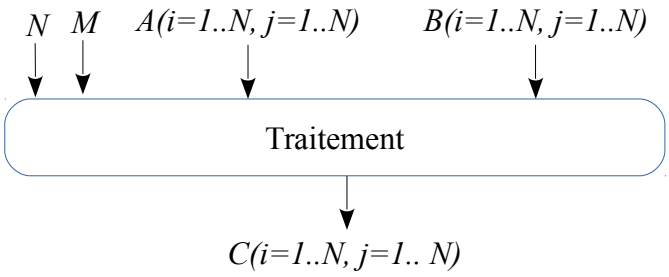


Exercice 10 : Produit de deux matrices

Soit A et B deux matrices carrées d'ordre N. Écrire un algorithme/programme PASCAL qui permet de calculer le produit de A et B.

Solution

L'algorithme



Explication

Pour réaliser le produit de deux matrices A et B, il faut que le nombre de colonnes de A soit égale au nombre de ligne de B. La matrice C résultat aura le même nombre de lignes de A et le même nombre de colonnes de B. Ainsi, on écrit : A(NxM) x B(MxL) = C(NxL) (A est une matrice de N lignes et M colonnes, B est une matrice de M lignes et L colonnes, donc C elle aura N lignes et L colonnes).

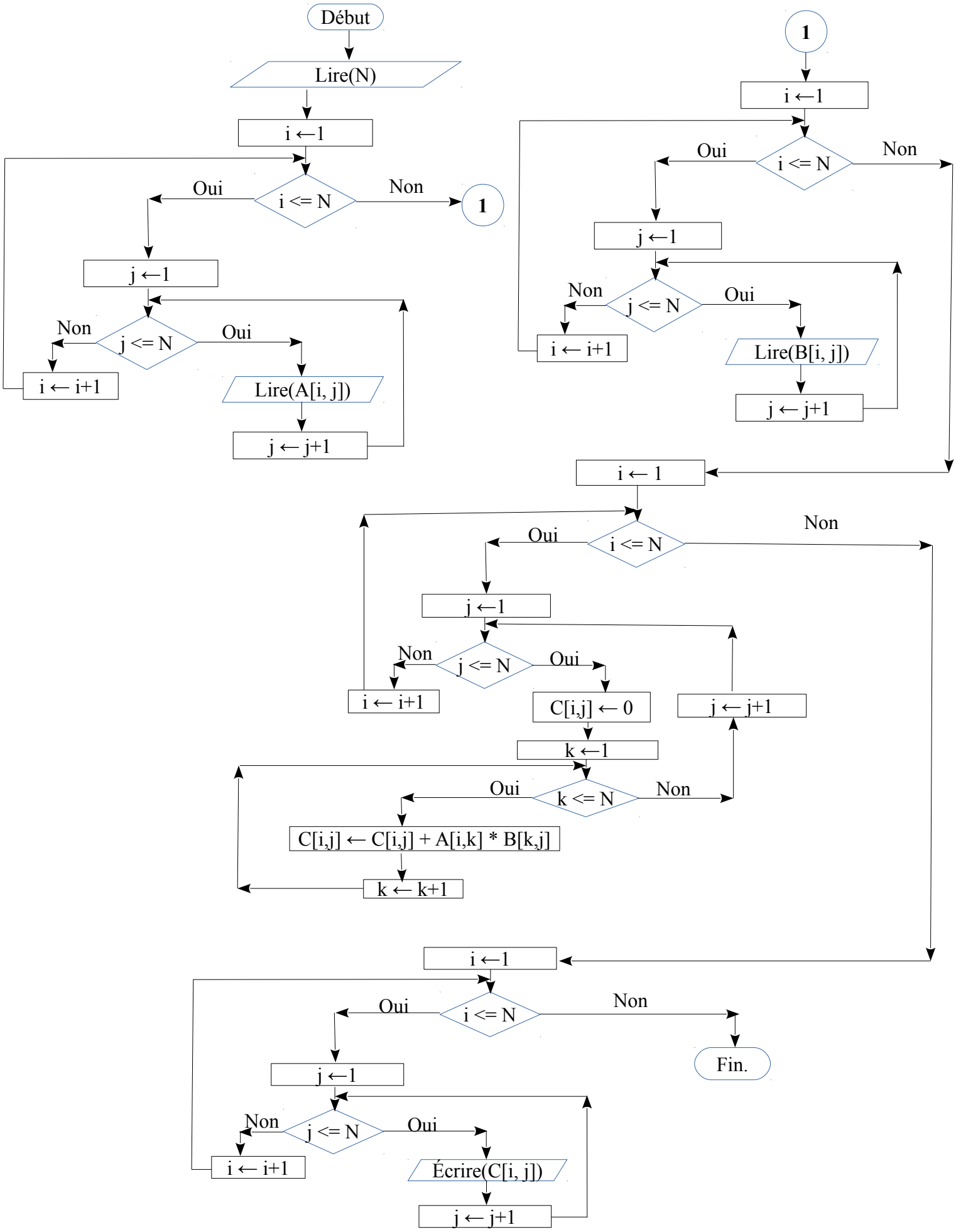
Chaque case de C sera calculée en utilisant la formule suivante :

C[i, j] = A[i, 1]*B[1, j] + A[i, 2]*B[2, j] + A[i, 3]*B[3, j] + … + A[i, M]*B[M, j]

Pour chaque i=1...N et j=1...L

Donc, C[i, j] = $\sum_{k=1}^{M} A[i,k] \times B[k,j]$

Dans le cas de matrices carrée d'ordre N (comme le cas de cet exercice), la formule devient comme suit : C[i, j] = $\sum_{k=1}^{N} A[i,k] \times B[k,j]$


Début
Lire(N)
i ←1
i <= N
Oui
Non
1
j ←1
j <= N
Non
Oui
i ← i+1
Lire(A[i, j])
j ← j+1
1
i ←1
i <= N
Oui
Non
j ←1
j <= N
Non
Oui
i ← i+1
Lire(B[i, j])
j ← j+1
i ← 1
i <= N
Oui
Non
j ←1
j <= N
Non
Oui
i ← i+1
C[i,j] ← 0
j ← j+1
k ←1
k <= N
Oui
Non
C[i,j] ← C[i,j] + A[i,k] * B[k,j]
k ← k+1
i ←1
i <= N
Oui
Non
Fin.
j ←1
j <= N
Non
Oui
i ← i+1
Écrire(C[i, j])
j ← j+1