République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Abderrahmane Mira - Bejaia

Faculté de la Technologie

Département de Technologie

Polycopié de cours

Première Année Technologie

Informatique 2

Enseignant :

Dr. OUARET Ahmed

Année Universitaire : **2024/2025**

A propos de ce cours

Ce support de cours concerne le module Informatique 2, destiné aux étudiants de première année (L1) Technologie, et s'inscrit dans la continuité du module Informatique 1. Ce cours vise à approfondir les notions fondamentales de la programmation et de la gestion des données en abordant trois chapitres essentiels : les variables indicées (tableaux), qui permettent de stocker, organiser et manipuler efficacement des ensembles de données ; les fonctions et procédures (Sous-programmes), qui favorisent une approche modulaire et réutilisable du code ; et enfin, les enregistrements et fichiers, qui introduisent des techniques de structuration et de sauvegarde de données complexes. À l'issue de ce cours, l'étudiant sera capable de concevoir et de développer des programmes répondant à des problématiques de niveau intermédiaire.

Pré-requis

Pour bien suivre ce cours, l'étudiant doit avoir acquis certaines connaissances sur :

- Une bonne maîtrise des bases de l'algorithmique et de la programmation acquise dans le module Informatique 1.
- La connaissance des structures de données fondamentales (variables, constantes et types scalaires).
- Une expérience pratique avec l'environnement de développement Pascal et la résolution d'exercices simples.

Les compétences visées

Ce cours vise à :

- Maîtriser l'utilisation des tableaux, vecteurs et matrices pour organiser et traiter efficacement des données.
- Concevoir et implémenter des sous-programmes (fonctions et procédures) afin de favoriser une programmation modulaire et réutilisable.
- Gérer les enregistrements et fichiers pour structurer, stocker et récupérer des informations complexes.
- Intégrer ces concepts pour développer des programmes informatiques complets et résoudre des problématiques de niveau intermédiaire.

Contenu de l'enseignement

Cours : 1h30 par semaine

TP : 1h30 par semaine

Méthode d'évaluation : 60% examen et 40% contrôle continu

Table des matières

Introduction générale

Introduction

Ce cours d'informatique 2, destiné aux étudiants de première année ST, vise à approfondir les connaissances fondamentales acquises précédemment tout en introduisant de nouveaux concepts essentiels pour la programmation moderne. En s'appuyant sur une approche théorique et pratique, le cours se structure autour de trois chapitres majeurs.

Le **premier chapitre** porte sur les variables indicées (tableaux), incluant vecteurs et matrices, et permet d'apprendre à organiser et manipuler efficacement des ensembles de données.

Le **deuxième chapitre** aborde les fonctions et procédures (sous-programmes), favorisant ainsi une programmation modulaire et réutilisable, indispensable pour la conception de programmes robustes.

Enfin, le **troisième chapitre** traite des enregistrements et fichiers, outils cruciaux pour la structuration, la gestion et la persistance des informations dans des applications informatiques.

À l'issue de ce cours, les étudiants seront mieux préparés à relever des défis de programmation plus complexes et à développer des solutions informatiques adaptées aux besoins contemporains.

CHAPITRE I

Les variables indicées

Chapitre I : Les variables indicées

I.1. Objectif de ce chapitre

A l'issu de ce chapitre, l'apprenant sera capable de :

- Définir ce qu'est un tableau et expliquer son utilité en informatique.
- Déclarer, initialiser et manipuler les éléments d'un tableau dans un programme.
- Effectuer des opérations courantes : recherche, tri et calculs sur les éléments d'un tableau.
- Comprendre et utiliser des tableaux multidimensionnels (matrices).
- Appliquer les tableaux pour résoudre des problèmes concrets en programmation.

I.2. Introduction

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

Exemple :

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

Solution :

Algorithme Note ;

Variables

N1, N2, N3, N4, N5 : réel ;

Début

Ecrire ('Enter la 1ère Note') ;

Lire(N1) ;

Ecrire ('Enter la 2ème Note') ;

Lire(N2) ;

Ecrire ('Enter la 3ème Note') ;

Lire(N3) ;

Ecrire ('Enter la 4ème Note') ;

```
Lire(N4) ;
Ecrire ('Enter la 5ème Note') ;
Lire(N5) ;
Ecrire('La note 1 multipliée par 3 est : ',N1*3) ;
Ecrire('La note 2 multipliée par 3 est : ',N2*3) ;
Ecrire('La note 3 multipliée par 3 est : ',N3*3) ;
Ecrire('La note 4 multipliée par 3 est : ',N4*3) ;
Ecrire('La note 5 multipliée par 3 est : ',N5*3) ;
```

Fin.

La même instruction se répète cinq fois. Imaginons que si l'on voudrait réaliser cet algorithme avec 100 notes, cela deviendrait fastidieux.

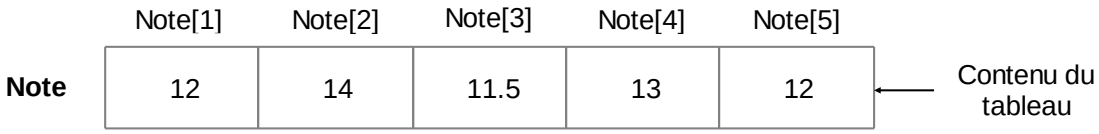
Comme les variables ont des noms différents, on ne peut pas utiliser de boucle, ce qui allonge considérablement le code et le rend très répétitif.

Pour résoudre ce problème, il existe un type de données qui permet de définir plusieurs variables de même type.

I.3. Définition

Un tableau est une suite d'éléments de même type, Il utilise plusieurs cases mémoire à l'aide d'un seul nom. Comme toutes les cases portent le même nom, elles se différencient par un numéro ou un indice.

Nous pouvons représenter schématiquement un tableau nommé Note composé de cinq cases, dans la mémoire comme suit :



Nous disposons alors de cinq variables. Pour les nommer, on indique le nom du tableau suivi de son indice entre crochets ou entre parenthèses : La première s'appelle Note[1], la deuxième Note[2], etc. jusqu'à la dernière Note[5].

Note[4] représente le 4ème élément du tableau **Note** et vaut 13.

I.4. Tableau à une dimension

I.4.1. Déclaration

La déclaration d'un tableau permet d'associer à un nom une zone mémoire composée d'un certain nombre de cases mémoires de même type.

Syntaxe

Algorithme :

Variable
Variable identificateur : tableau [Indice_min .. Indice_max] de type ;

Pascal :

Var
Variable identificateur : array [Indice_min .. Indice_max] of type ;

Ou bien

Algorithme
Constante
MAX = Indice_max ;
Variable
Variable identificateur : tableau [1..MAX] de type;

Pascal
Const
MAX = Indice_max ;
Var
Variable identificateur : Array [1..MAX] of type ;

Remarques :

- Le premier élément d'un tableau porte l'indice zéro ou l'indice 1 selon les langages (1 en langage Pascal).
- La valeur d'un indice doit être un nombre entier.
- La valeur d'un indice doit être inférieure ou égale au nombre d'éléments du tableau. Par exemple, avec le tableau tab[1..20], il est impossible d'écrire tab[0] et tab[21]. Ces expressions font référence à des éléments qui n'existent pas.
- C'est pas obligatoire d'utiliser une constante pour la taille maximale de tableau ; cependant, c'est une bonne pratique dans la programmation

Exemple :

L'instruction suivante déclare un tableau de 30 éléments de type réel :

Algorithme	Pascal
Variable Note : tableau [1..30] de réels ;	**Var** Note : array [1..30] of real ;

Note : c'est le nom du tableau (identificateur).

1 : c'est l'indice du premier élément du tableau.

30 : c'est l'indice du dernier élément du tableau (nombre d'éléments du tableau).

Exercice :

Déclarer deux tableaux nommés A et B composé chacun d'eux de 10 éléments de type chaine.

Solution :

Algorithme	Pascal
Variable A, B : tableau [1..10] de chaine ;	**Var** A, B : array [1..10] of string ;

Remarques :

- Lorsqu'on déclare un tableau, on déclare aussi de façon implicite toutes les variables indicées qui le constituent. Nous utiliserons souvent la valeur 1 comme premier indice (Pascal) mais on peut aussi utiliser un autre indice minimum, comme 0. Dans ce cas, l'indice maximum sera égal au nombre d'éléments -1.
- Dans le langage C, les cases du tableau sont numérotées à partir de 0. En Visual Basic l'indice minimum est 1.

Exemple 1 :

L'instruction suivante affecte à la variable X la valeur du premier élément du tableau Note :

Algorithme	Pascal
X ← Note [1] ;	X := Note [1] ;

Exemple 2 :

L'instruction suivante affiche le contenu du quatrième élément du tableau Note :

Algorithme	Pascal
Ecrire(Note[4]) ;	write(Note[4]) ;

Exemple 3 :

L'instruction suivante affecte une valeur introduite par l'utilisateur à l'élément trois du tableau Note :

Algorithme	Pascal
Lire(Note[3]) ;	read(Note[3]) ;

I.4.2. Initialisation

Un tableau peut être initialisé en utilisant deux méthodes :

- En utilisant des affectations ;
- En utilisant la lecture à partir d'un fichier de données ou à partir de clavier.

a) Par affectations : On utilise pour cela l'opération d'affectation.

Exemple :

V[1] := 24.00; *V[2] := 38.25*; *V[3] := 28.00*; *V[4] := 70.25*; *V[5] := 63.00* ; *V[6] := 96.25*;

b) Par lecture : On utilise pour cela l'opération de lecture **READ** : C'est la méthode la plus commode.

Exemple1 :

Exemple d'algorithme / programme Pascal permettant de lire et d'écrire (afficher) le tableau précédent :

Algorithme	Pascal

algorithme LireEcrireTableau
variables
v : tableau[1..6] de réel
i : entier
Début
ecrire('Introduire V :')
pour i:=1 **à** 6 **faire**
lire(v[i])
finPour
ecrire('Affichage V :')
pour i:=1 **à** 6 **faire**
ecrire(v[i])
finPour
Fin
	program LireEcrireTableau;
	var
	V : **array**[1..6] **of** real;
	i : integer;
	Begin
	writeln('Introduire V : ');
	for i:=1 **to** 6 **do**
	Read (V[I]);
	
	writeln ('Affichage V : ') ;
	for i:=1 **to** 6 **do**
	Write (V[I]);
	End.

I.4.3. Manipulation des tableaux à une dimension (vecteurs)

Les tableaux ont une grande importance en informatique, la quasi-totalité des problèmes utilisent des structures de données sous forme de tableaux. On peut en citer le domaine du calcul matriciel, les statistiques, les traitement de gestion, *etc*. La gestion et l'analyse de données nécessitent l'organisation des données dans des tableaux pour rendre possible leur traitement informatique.

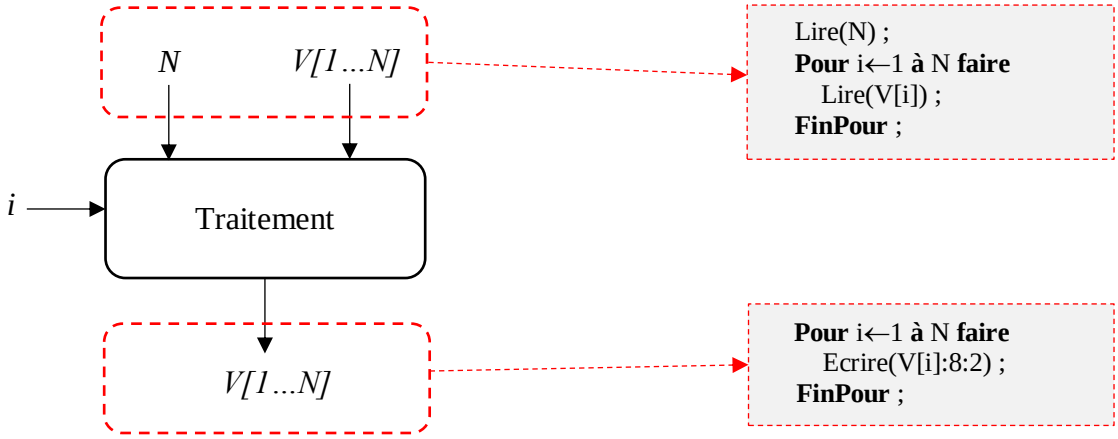
I.5. Exercices avec corrigés

Exercice N°01 : Lecture et Affichage d'un vecteur

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

Corrigé de l'exercice N°01 :

Les variables d'entrée, variable de sorite et la partie traitement sont présentées sur le schéma ci-dessous:

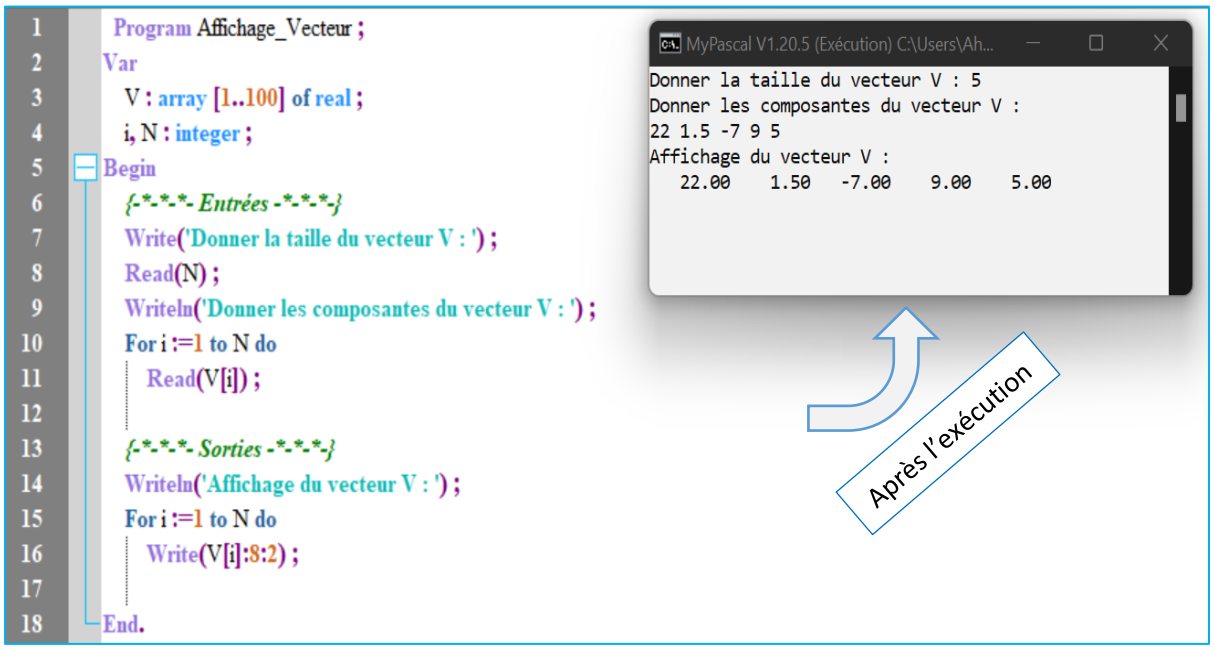


Remarques :

- Il faut noter que ce programme ne réalise pas de traitement, il ne contient que des entrées et des sorties.
- La variable ***i*** est une variable de traitement ou intermédiaire, utilisée pour parcourir le vecteur T.

Algorithme	Pascal
Algorithme Affichage_Vecteur ; **Variables**	**Program** Affichage_Vecteur ; **Var**

V : **Tableau** [1..100] **de réel** ;
i, N : **entier** ;
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donner la taille du vecteur V : ') ;
Lire(N) ;
Ecrire('Donner les composantes du vecteur V : ') ;
Pour i←1 **à** N **faire**
Lire(V[i]) ;
FinPour ;
{-*-*-*- Sorties -*-*-*-}
Ecrire('Affichage du vecteur V : ') ;
Pour i←1 **à** N **faire**
Ecrire(V[i]:8:2) ;
FinPour ;
Fin.
	V : **array** [1..100] **of real** ;
	i, N : **integer** ;
	Begin
	{-*-*-*- Entrées -*-*-*-}
	Write('Donner la taille du vecteur V : ') ;
	Read(N) ;
	Writeln('Donner les composantes du vecteur V : ') ;
	For i :=1 **to** N **do**
	Read(V[i]) ;
	{-*-*-*- Sorties -*-*-*-}
	Writeln('Affichage du vecteur V : ') ;
	For i :=1 **to** N **do**
	Write(V[i]:8:2) ;
	End.



Explication :

Ce programme montre comment lire et écrire un vecteur. Pour les deux opérations (lecture et écriture), nous aurons toujours besoin d'une boucle **Pour**. Lors de la déclaration, nous réservons 100 cases réelles (la taille maximale du vecteur), et nous utilisons la variable **n** pour

déterminer la taille que nous voulons utiliser (par exemple 5 cases). L'accès à une composante d'indice **i** se fait comme suit : **V[i]**. Ainsi, pour lire la case 2, nous écrivons **Lire(V[2])**, et **Ecrire(T[4])** pour afficher la valeur de la 4ème case.

Exercice N°02 :

Ecrire un algorithme/programme Pascal permettant de saisir 10 notes et de les afficher après avoir multiplié toutes ces notes par un coefficient fourni par l'utilisateur :

Corrigé de l'exercice N°02 :

Algorithme	Pascal

Algorithme tableau_note ;
Variables
Note : tableau [1..10] de réels ;
Coef, i : entier ;
Début
Ecrire('Entrer le coefficient : ') ;
Lire(Coef) ;
{Remplissage du tableau Note}
Pour i ← 1 **à** 10 **faire**
Ecrire('Entrer la valeur de la note ', i,' : ') ;
Lire(Note[i]) ;
Fin-Pour ;
Ecrire('Affichage des notes*Coef : ') ;
Pour i ← 1 **à** 10 **faire**
Ecrire(Note[i]*coef :8:2) ;
Fin-Pour ;
Fin.
	Program tableau_note ;
	Var
	Note : array [1..10] of real ;
	Coef, i : integer ;
	Begin
	write('Entrer le coefficient : ') ;
	read(Coef) ;
	{Remplissage du tableau Note}
	For i := 1 **to** 10 **do**
	Begin
	write('Entrer la valeur de la note ', i,' : ') ;
	read(Note[i]) ;
	End ;
	Write('Affichage des notes*Coef : ') ;
	For i := 1 **to** 10 **do**
	write(Note[i]*coef :8:2) ;
	End.

```
1  Program tableau_note ;
2  Var
3    Note : array [1..10] of real ;
4    Coef, i : integer ;
5  Begin
6    write('Entrer le coefficient : ') ;
7    read(Coef) ;
8    {Remplissage du tableau Note}
9    For i := 1 to 10 do
10     Begin
11       write('Entrer la valeur de la note ', i,' : ') ;
12       read(Note[i]) ;
13     End ;
14   Write('Affichage des notes*Coef : ') ;
15   For i := 1 to 10 do
16     write(Note[i]*coef :8:2) ;
17 End.
```

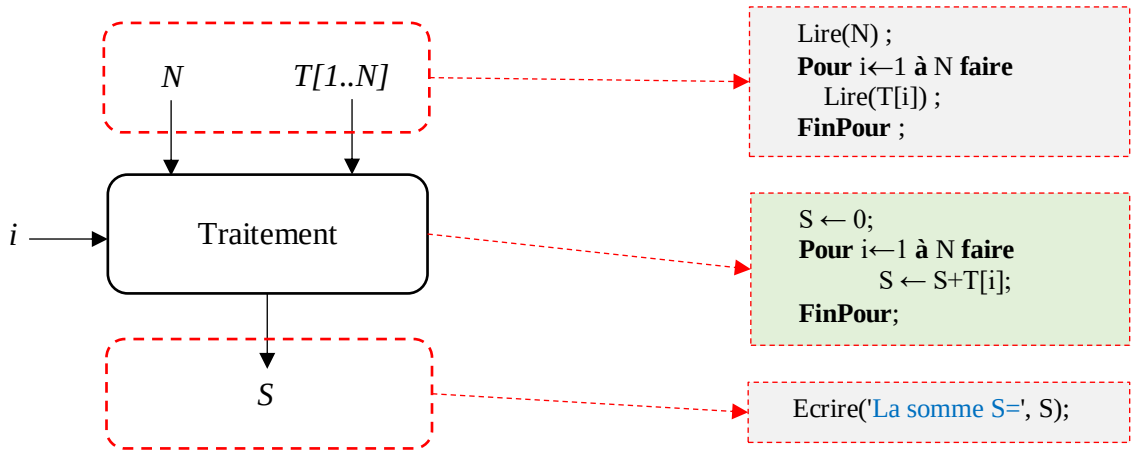
```
MyPascal V1.20.5 (Exécution)
Entrer le coefficient : 3
Entrer la valeur de la note 1 : 12
Entrer la valeur de la note 2 : 4
Entrer la valeur de la note 3 : 5
Entrer la valeur de la note 4 : 5
Entrer la valeur de la note 5 : 7
Entrer la valeur de la note 6 : 7
Entrer la valeur de la note 7 : 5
Entrer la valeur de la note 8 : 3
Entrer la valeur de la note 9 : 9
Entrer la valeur de la note 10 : 6
Affichage des notes*Coef :    36.00   12.00   15.00   15.00
   21.00   21.00   15.00    9.00   27.00   18.00
```

Exercice N°03 :

Ecrire un algorithme/programme Pascal qui calcule la somme des éléments d'un tableau.

Corrigé de l'exercice N°03 :

Les variables d'entrée, variable de sorite et la partie traitement sont présentées dans le schéma ci-dessous :

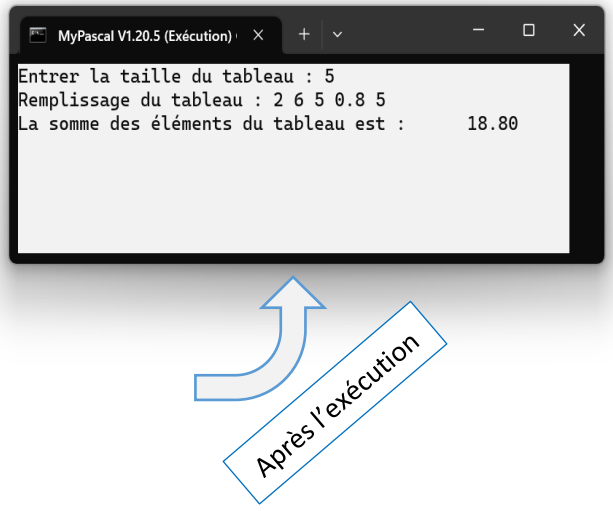


On suppose que le nombre d'éléments du tableau ne dépasse pas 100.

Algorithme	Pascal
Algorithme Somme_tableau ;	**Program** Somme_tableau ;
Constante M=100 ;	**Const** M=100 ;
Variable T : tableau [1..M] de réels ;	**Var** T : array [1..M] of real ;
N, i : entier ;	N, i : integer ;
S : réel ;	S : real ;
Début	**Begin**

Ecrire('Entrer la taille du tableau : ') ;
Lire(N) ;
Si (N>M) **alors**
N ← M ;
Fin-Si ;
Ecrire('Remplissage du tableau : ') ;
Pour i ← 1 **à** N **faire**
Lire(T[i]) ;
Fin-Pour ;
S ← 0 ;
Pour i ← 1 **à** N **faire**
S ← S+T[i] ;
Fin-Pour ;
Ecrire('La somme des éléments du tableau est : ', S:10:2) ;
Fin.
	write('Entrer la taille du tableau : ') ;
	read(N) ;
	if (N>M) **then**
	N := M ;
	
	write('Remplissage du tableau : ') ;
	For i := 1 **to** N **do**
	read(T[i]) ;
	
	S := 0 ;
	For i := 1 **to** N **do**
	S := S+T[i] ;
	
	write('La somme des éléments du tableau est : ', S:10:2) ;
	End.

```
1  Program Somme_tableau ;
2  Const M=100 ;
3  Var T : array [1..M] of real ;
4          N, i : integer ;
5          S : real ;
6  Begin
7    write('Entrer la taille du tableau : ') ;
8    read(N) ;
9    if (N>M) then
10     N := M ;
11
12   write('Remplissage du tableau : ') ;
13   For i := 1 to N do
14     read(T[i]) ;
15
16   S := 0 ;
17   For i := 1 to N do
18     S := S+T[i] ;
19
20   write('La somme des éléments du tableau est : ', S:10:2) ;
21 End.
```

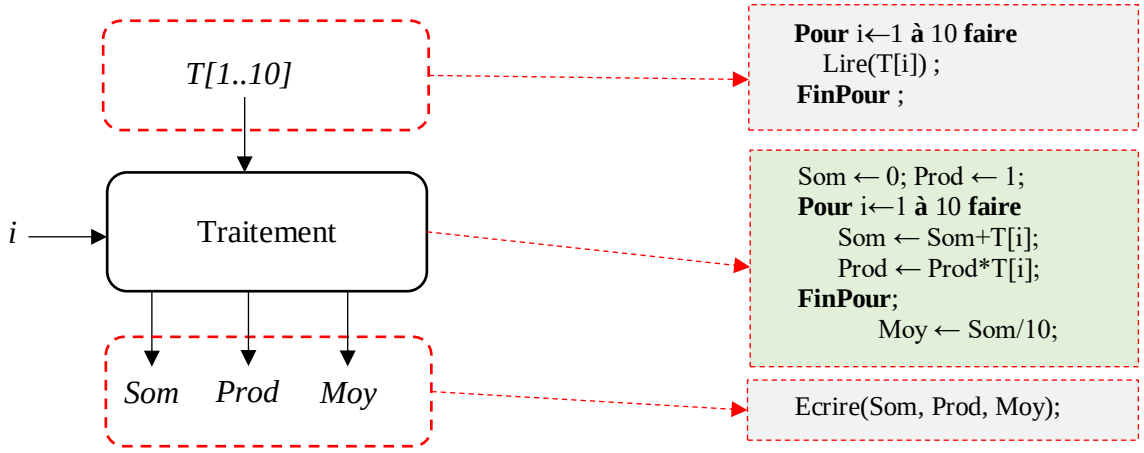


Exercice N°04 : (Calcul de la somme, le produit et la moyenne des éléments numériques d'un tableau)

Ecrire un algorithme permettant de calculer la somme, le produit et la moyenne d'un tableau de dix réels. Traduire l'algorithme en Pascal.

Corrigé de l'exercice N°04 :

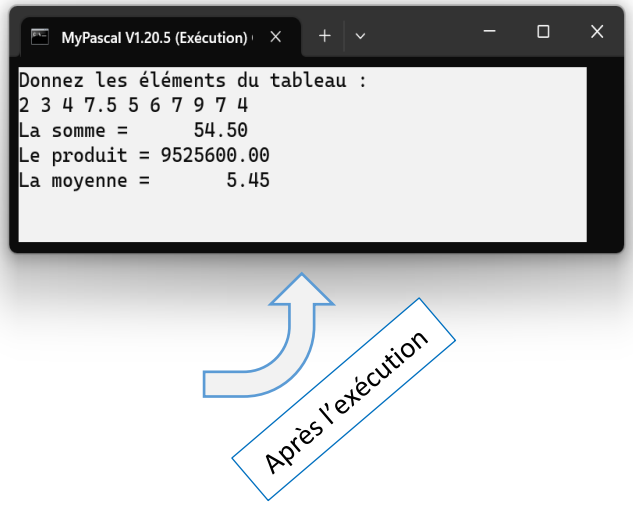
Les variables d'entrée, variable de sorite et la partie traitement sont présentées dans le schéma ci-dessous :



Algorithme	Pascal

Algorithme Som_Moy_Prod_Tab ;
Variables
T : tableau [1..10] de réel ;
i : entier ;
Som, Prod, Moy : réel ;
Début
Ecrire('Donnez les éléments du tableau : ') ;
Pour i ← 1 **à** 10 **faire**
Lire(T[i]) ;
Fin-Pour ;
Som ← 0 ;
Prod ← 1 ;
Pour i ← 1 **à** 10 **faire**
Som ← Som+T[i] ;
Prod ← Prod*T[i] ;
Fin-Pour ;
Moy ← Som/10 ;
Ecrire('La somme = ', Som:10:2) ;
Ecrire('Le produit = ', Prod:10:2) ;
Ecrire('La moyenne = ', Moy:10:2) ;
Fin.
	Program Som_Moy_Prod_Tab ;
	Var
	T : array [1..10] of real ;
	i : integer ;
	Som, Prod, Moy : real ;
	Begin
	writeln('Donnez les éléments du tableau : ') ;
	For i := 1 **to** 10 **do**
	read(T[i]) ;
	Som := 0 ;
	Prod := 1 ;
	For i := 1 **to** 10 **do**
	Begin
	Som := Som+T[i] ;
	Prod := Prod*T[i] ;
	End ;
	Moy := Som/10 ;
	writeln('La somme = ', Som:10:2) ;
	writeln('Le produit = ', Prod:10:2) ;
	writeln('La moyenne = ', Moy:10:2) ;
	End.

```
1  Program Som_Moy_Prod_Tab ;
2  Var
3    T : array [1..10] of real ;
4    i : integer ;
5    Som, Prod, Moy : real ;
6  Begin
7    writeln('Donnez les éléments du tableau : ') ;
8    For i := 1 to 10 do
9      read(T[i]) ;
10
11   Som := 0 ;
12   Prod := 1 ;
13   For i := 1 to 10 do
14     Begin
15       Som := Som+T[i] ;
16       Prod := Prod*T[i] ;
17     End ;
18   Moy := Som/10 ;
19
20   writeln('La somme = ', Som:10:2) ;
21   writeln('Le produit = ', Prod:10:2) ;
22   writeln('La moyenne = ', Moy:10:2) ;
23 End.
```



Exercice N°05 : (La recherche du plus petit et plus grand élément dans un tableau)

Ecrire un algorithme permettant de déterminer la valeur maximale et minimale dans un tableau de dix entiers, avec leurs positions dans le tableau. Traduire l'algorithme en Pascal.

Corrigé de l'exercice N°05 :

Algorithme	Pascal

Algorithme Max_Min_Tab ;
Variables
T : tableau [1..10] d'entier ;
Max, Min, Pos_Max, Pos_Min, i : entier ;
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donnez les éléments du tableau : ') ;
Pour i ← 1 **à** 10 **faire**
Lire(T[i]) ;
Fin-Pour ;
{-*-*-*- Traitement -*-*-*-}
Max ← T[1] ;
Pos_Max ← 1 ;
Pour i ← 2 **à** 10 **faire**
Si (Max<T[i]) **Alors**
	Program Max_Min_Tab ;
	Var
	T : array [1..10] of integer ;
	Max, Min, Pos_Max, Pos_Min, i : integer ;
	Begin
	{-*-*-*- Entrées -*-*-*-}
	writeln('Donnez les éléments du tableau : ') ;
	For i := 1 **to** 10 **do**
	read(T[i]) ;
	{-*-*-*- Traitement -*-*-*-}
	Max := T[1] ;
	Pos_Max := 1 ;
	For i := 2 **to** 10 **do**
	if (Max<T[i]) **then**
	Begin

```
        Max ← T[i] ;
        Pos_Max ← i ;
    Fin-Si ;
Fin-Pour ;

Min ← T[1] ;
Pos_Min ← 1 ;
Pour i ← 2 à 10 faire
   Si (Min>T[i]) Alors
      Min ← T[i] ;
      Pos_Min ← i ;
   Fin-Si ;
Fin-Pour ;

{-*-*-*- Sorties -*-*-*-}
Ecrire('La valeur maximale = ', Max,
        'occupant la position ', Pos_Max) ;
Ecrire('La valeur minimale = ', Min, 'occupant
        la position ', Pos_Min) ;

Fin.
```

```
        Max := T[i] ;
        Pos_Max := i ;
    End ;

Min := T[1] ;
Pos_Min := 1 ;
For i := 2 to 10 do
   if (Min>T[i]) then
    Begin
      Min := T[i] ;
      Pos_Min := i ;
    End ;

{-*-*-*- Sorties -*-*-*-}
writeln('La valeur maximale = ', Max,
        'occupant la position ', Pos_Max) ;
writeln('La valeur minimale = ', Min,
        'occupant la position ', Pos_Min) ;

End.
```

```
Program Max_Min_Tab ;
Var
  T : array [1..10] of integer ;
  Max, Min, Pos_Max, Pos_Min, i : integer ;
Begin
  {-*-*-*- Entrées -*-*-*-}
  writeln('Donnez les éléments du tableau : ') ;
  For i := 1 to 10 do
    read(T[i]) ;
  {-*-*-*- Traitement -*-*-*-}
  Max := T[1] ;
  Pos_Max := 1 ;
  For i := 2 to 10 do
    if (Max<T[i]) then
      Begin
        Max := T[i] ;
        Pos_Max := i ;
      End ;
  Min := T[1] ;
  Pos_Min := 1 ;
  For i := 2 to 10 do
    if (Min>T[i]) then
      Begin
        Min := T[i] ;
        Pos_Min := i ;
      End ;
  {-*-*-*- Sorties -*-*-*-}
  writeln('La valeur maximale = ', Max, ' occupant la position ', Pos_Max) ;
  writeln('La valeur minimale = ', Min, ' occupant la position ', Pos_Min) ;
End.
```

```
MyPascal V1.20.5 (Exécution)
Donnez les éléments du tableau :
5 6 7 99 5 3 0 5 7 2
La valeur maximale = 99 occupant la position 4
La valeur minimale = 0 occupant la position 7
```

Après l'exécution

Exercice N°06 :

Que fait l'algorithme suivant ?

```
Algorithme X ;
Variables
    NB : Tableau [1..5] d'entier ;
    i : entier ;
Début
Pour i ← 1 à 5 faire
   NB[i] ← i*i ;
Fin-Pour;
Pour i ← 1 à 5 faire
    Ecrire(NB[i]) ;
Fin-Pour;
Fin.
```

Corrigé de l'exercice N°06 :

Cet algorithme remplit un tableau avec cinq valeurs : 1, 4, 9, 16, 25. Il les affiche ensuite à l'écran.

Voici une simplification :

```
Algorithme X ;
Variables
    NB : Tableau [1..5] d'entier ;
    i : entier ;
Début
   Pour i ← 1 à 5 faire
      NB[i] ← i*i ;
      Ecrire(NB[i]) ;
   Fin-Pour ;
Fin.
```

Exercice N°07 :

Que fait l'algorithme suivant ?

```
Algorithme X ;
Variables
    N : Tableau [1..6] d'entier ;
    i,k : entier ;
Début
   N[1] ← 1 ;
   Pour k ← 2 à 6 faire
      N[k] ← N[k-1]+2 ;
   Fin-Pour;
   Pour i ← 1 à 6 faire
      Ecrire(N[i]) ;
   Fin-Pour;
Fin.
```

Peut-on simplifier cet algorithme pour avoir le même résultat ?

Corrigé de l'exercice N°07 :

Cet algorithme remplit un tableau avec les six valeurs : 1, 3, 5, 7, 9, 11. Il les affiche ensuite à l'écran.

Voici une simplification :

```
Algorithme X ;
Variables
    N : Tableau [1..6] d'entier ;
    i,k : entier ;
Début
   N[1] ← 1 ;
Ecrire(N[1]) ;
Pour k ← 2 à 6 faire
   N[k] ← N[k-1]+2 ;
   Ecrire(N[k]) ;
Fin-Pour;
Fin.
```

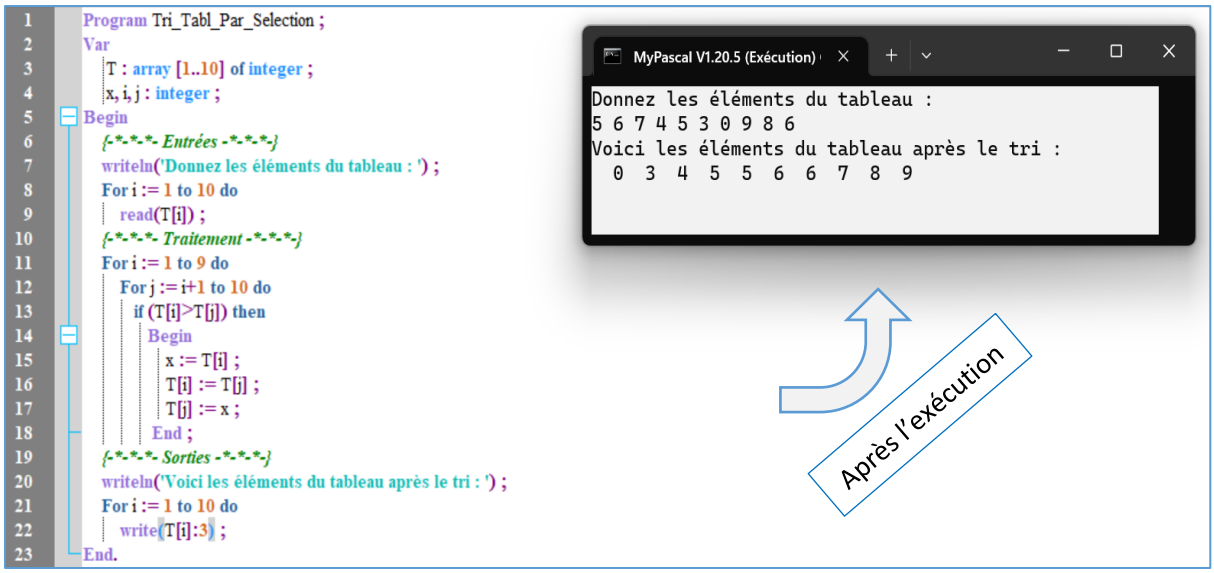
Exercice N°08 : (Le tri d'un tableau par sélection)

Ecrire un algorithme permettant de trier un tableau de dix entiers en ordre croissant. Traduire l'algorithme en Pascal.

Corrigé de l'exercice N°08 :

Algorithme	Pascal

Algorithme Tri_Tabl_Par_Selection ;
Variables
T : tableau [1..10] d'entier ;
x, i, j : entier ;
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donnez les éléments du tableau : ') ;
Pour i ← 1 **à** 10 **faire**
Lire(T[i]) ;
Fin-Pour ;
{-*-*-*- Traitement -*-*-*-}
Pour i ← 1 **à** 9 **faire**
Pour j ← i+1 **à** 10 **faire**
Si (T[i]>T[j]) **Alors**
x ← T[i] ;
T[i] ← T[j] ;
T[j] ← x ;
Fin-Si ;
Fin-Pour ;
Fin-Pour ;
{-*-*-*- Sorties -*-*-*-}
Ecrire('Voici les éléments du tableau après le tri : ') ;
Pour i ← 1 **à** 10 **faire**
Ecrire(T[i]:3) ;
Fin-Pour ;
Fin.
	Program Tri_Tabl_Par_Selection ;
	Var
	T : array [1..10] of integer ;
	x, i, j : integer ;
	Begin
	writeln('Donnez les éléments du tableau : ') ;
	For i := 1 **to** 10 **do**
	read(T[i]) ;
	{-*-*-*- Traitement -*-*-*-}
	For i := 1 **to** 9 **do**
	For j := i+1 **to** 10 **do**
	if (T[i]>T[j]) **then**
	Begin
	x := T[i] ;
	T[i] := T[j] ;
	T[j] := x ;
	End ;
	{-*-*-*- Sorties -*-*-*-}
	writeln('Voici les éléments du tableau après le tri : ') ;
	For i := 1 **to** 10 **do**
	write(T[i]:3) ;
	End.



Exercice N°09 :

Ecrire un algorithme permettant de saisir un tableau de N éléments des valeurs réelles non nulles. Une fois la saisie terminée, le programme affichera le nombre de valeurs négatives et le nombre de valeurs positives.

Corrigé de l'exercice N°09 :

Algorithme	Pascal

Algorithme Val_Neg_Val_Pos;
Variables
T : Tableau [1..100] de réels;
N, i, NVP, NVN : entier;
 	
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donner la taille du vecteur T : ');
Lire(N);
Ecrire('Donner les composantes non nulles du vecteur T');
Pour i ← 1 **à** N **faire**
Lire(T[i]);
Fin-Pour ;
{-*-*-*- Traitement -*-*-*-}
NVP ← 0;
NVN ← 0;
	Program Val_Neg_Val_Pos;
	Var
	T : array [1..100] of real;
	N, i, NVP, NVN : integer;
	
	Begin
	{-*-*-*- Entrées -*-*-*-}
	write('Donner la taille du vecteur T : ');
	read(N);
	writeln('Donner les composantes non nulles du vecteur T');
	For i:=1 **to** N **do**
	read(T[i]);
	
	{-*-*-*- Traitement -*-*-*-}
	NVP:=0;
	NVN:=0;
	for i:=1 **to** N **do**

```
Pour i ← 1 à N faire
   Si T[i]>0 Alors
      NVP ← NVP+1
   Sinon
      NVN ← NVN+1;
   Fin-Si ;
Fin-Pour ;
{-*-*-*- Sorties -*-*-*-}
Ecrire('Le nombre de valeurs positives est : ',
        NVP);
Ecrire('Le nombre de valeurs négatives est :',
        NVN);
Fin.
```

```
if T[i]>0 then
   NVP:=NVP+1
else
   NVN:=NVN+1;

{-*-*-*- Sorties -*-*-*-}
writeln('Le nombre de valeurs positives est : ',
        NVP);
write('Le nombre de valeurs négatives est :',
        NVN);
End.
```

```
Program Val_Neg_Val_Pos;
Var
  T : array [1..100] of real;
  N, i, NVP, NVN : integer;

Begin
  {-*-*-*- Entrées -*-*-*-}
  write('Donner la taille du vecteur T : ');
  read(N);
  writeln('Donner les composantes non nulles du vecteur T');
  For i:=1 to N do
    read(T[i]);

  {-*-*-*- Traitement -*-*-*-}
  NVP:=0;
  NVN:=0;
  for i:=1 to N do
    if T[i]>0 then
      NVP:=NVP+1
    else
  NVN:=NVN+1;

  {-*-*-*- Sorties -*-*-*-}
  writeln('Le nombre de valeurs positives est : ', NVP);
  write('Le nombre de valeurs négatives est :', NVN);
End.
```

```
MyPascal V1.20.5 (Exécution)
Donner la taille du vecteur T : 6
Donner les composantes non nulles du vecteur T
4 -5 8 9 -3 1
Le nombre de valeurs positives est : 4
Le nombre de valeurs négatives est :2
```

Après l'exécution

Exercice N°10 :

Soit T un tableau contenant N entiers (10≤N≤50). On propose d'écrire un programme Pascal qui permet d'éclater T en deux tableaux :

TN (contenant les éléments négatifs de T) et **TP** (contenant les éléments positifs de T).

Exemple :

T	2	7	-4	0	-7	4	3	0	-8	-1	3

TN	-4	-7	-8	-1

TP	2	7	0	4	3	0	3

Corrigé de l'exercice N°10 :

Algorithme	Pascal

Algorithme Eclater_tab ;
Var
T, TN, TP : Tableau [1..50] d'entier ;
n, i, j, k : entier ;
 	
Début
{-*-*-*- Entrées -*-*-*-}
Répéter
Ecrire('Saisir un entier') ;
Lire(n) ;
Jusqu'à (n>=10) et (n<=50) ;
 	
Ecrire('Saisir les ', n, ' éléments de T') ;
Pour i ← 1 **à** n **faire**
Read (T[i]) ;
Fin-Pour;
 	
{-*-*-*- Traitement -*-*-*-}
j ← 0 ;
k ← 0 ;
Pour i ← 1 **à** n **faire**
Si T[i] < 0 **Alors**
j ← j+1 ;
TN[j] ← T[i] ;
Sinon
k ← k+1 ;
TP[k] ← T[i] ;
Fin-Pour;
 	
{-*-*-*- Sorties -*-*-*-}
Ecrire('TN : ');
Pour i ← 1 **à** j **faire**
Ecrire(TN[i]:4) ;
Fin-Pour;
	Program Eclater_tab ;
	Var
	T, TN, TP : array [1..50] of integer ;
	n, i, j, k : integer ;
	
	Begin
	{-*-*-*- Entrées -*-*-*-}
	Repeat
	Writeln('Saisir un entier') ;
	Readln(n) ;
	Until (n>=10) and (n<=50) ;
	
	Writeln ('Saisir les ', n, ' éléments de T') ;
	For i:=1 **to** n **do**
	Read (T[i]) ;
	{-*-*-*- Traitement -*-*-*-}
	j := 0 ; k := 0 ;
	For i := 1 **to** n **do**
	If T[i] < 0 **Then**
	Begin
	j := j+1 ;
	TN[j] := T[i] ;
	End
	Else
	Begin
	k := k+1 ;
	TP[k] := T[i] ;
	End ;
	{-*-*-*- Sorties -*-*-*-}
	Write('TN : ');
	For i := 1 **to** j **do**
	Write (TN[i]:4) ;
	Writeln;

Ecrire('TP : '); **Pour** i ← 1 **à** k **faire** Ecrire(TP[i]:4) ; **Fin-Pour;** **Fin.**	Write('TP : '); **For** i := 1 **to** k **do** Write (TP[i]:4) ; **End.**

```
1  Program Eclater_tab ;
2  Var
3  T, TN, TP : array [1..50] of integer ;
4  n, i, j, k : integer ;
5
6  Begin
7    {-*-*-*- Entrées -*-*-*-}
8    Repeat
9      Writeln('Saisir un entier') ;
10     Readln(n) ;
11   Until (n>=10) and (n<=50) ;
12
13   Writeln ('Saisir les ', n, ' éléments de T') ;
14   For i:=1 to n do
15     Read (T[i]) ;
16   {-*-*-*- Traitement -*-*-*-}
17   j := 0 ; k := 0 ;
18   For i := 1 to n do
19     If T[i] < 0 Then
20       Begin
21         j := j+1 ;
22         TN[j] := T[i] ;
23       End
24     Else
25       Begin
26         k := k+1 ;
27         TP[k] := T[i] ;
28       End ;
29   {-*-*-*- Sorties -*-*-*-}
30   Write('TN : ');
31   For i := 1 to j do
32     Write (TN[i]:4) ;
33     Writeln;
34   Write('TP : ');
35   For i := 1 to k do
36     Write (TP[i]:4) ;
37 End.
```

```
MyPascal V1.20.5 (Exécution)
Saisir un entier
11
Saisir les 11 éléments de T
1 3 7 3 5 -4 8 -2 8 -9 88
TN :   -4  -2  -9
TP :    1   3   7   3   5   8   8  88
```

Après l'exécution

Exercice N°11 : (Somme de deux vecteurs T1 et T2)

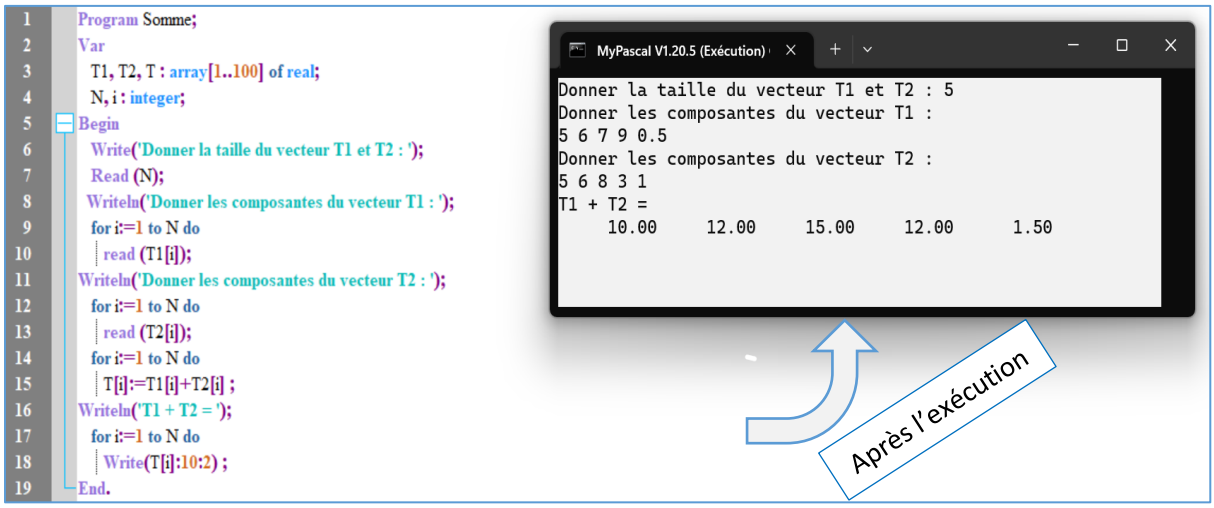
Ecrire un algorithme permettant de faire la somme de deux vecteurs T1 et T2 de N éléments des valeurs réelles. Traduire l'algorithme en Pascal.

Corrigé de l'exercice N°11 :

Algorithme	Pascal
Algorithme Somme ; **Variables**	**Program** Somme; **Var**

```
T1, T2, T : Tableau [1..100] de réel;
N, i : entier;
Début
Ecrire('Donner la taille du vecteur T1 et T2 : ');
Lire (N) ;
Ecrire('Donner les composantes du vecteur T1 : ');
pour i ← 1 à N faire
  Lire (T1[i]) ;
FinPour
Ecrire('Donner les composantes du vecteur T2 : ');
pour i ← 1 à N faire
  Lire (T2[i]) ;
FinPour
pour i=1 à N faire
  T[i] ←  T1[i] + T2[i] ;
FinPour
Ecrire('T1 + T2 = ');
pour i=1 à N faire
  Ecrire (T[i]:10:2) ;
FinPour
Fin.
```

```
T1, T2, T : array[1..100] of real;
N, i : integer;
Begin
Write('Donner la taille du vecteur T1 et T2 : ');
Read (N);
Writeln('Donner les composantes du vecteur T1 : ');
for i:=1 to N do
  read (T1[i]);
Writeln('Donner les composantes du vecteur T2 : ');
for i:=1 to N do
  read (T2[i]);
for i:=1 to N do
  T[i]:=T1[i]+T2[i] ;
Writeln('T1 + T2 = ');
for i:=1 to N do
  Write(T[i]:10:2) ;
End.
```



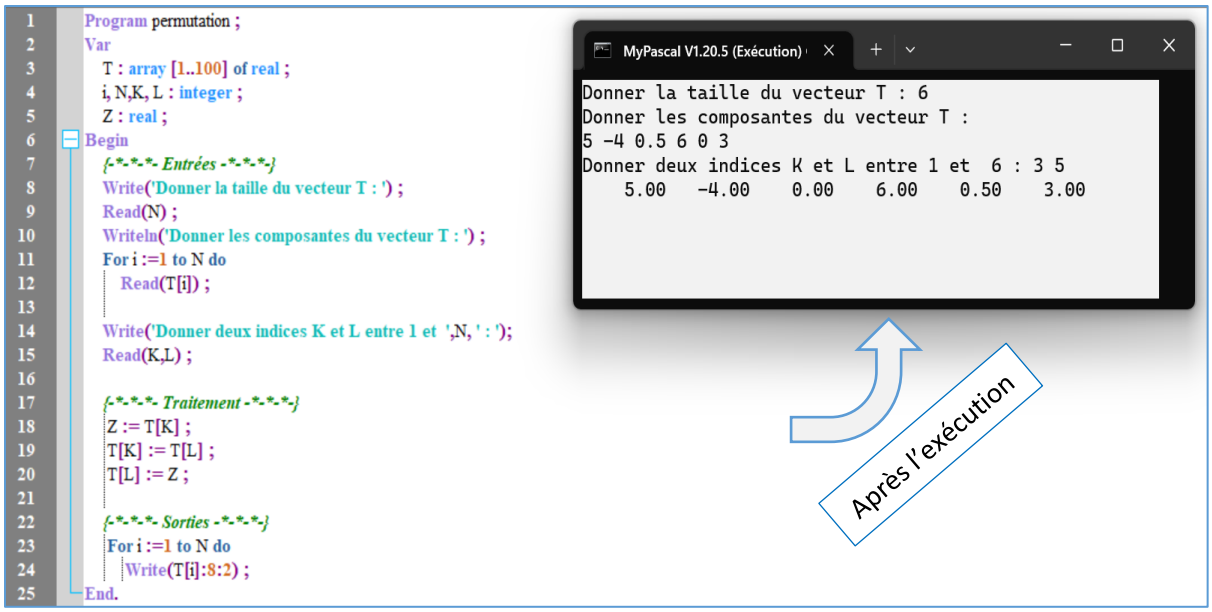
Exercice N°12 : (Permutation entre les cases d'indice K et L)

Soit T un vecteur de type réel et de taille N, et soient K et L deux positions dans le vecteur T. Écrire un algorithme/Programme PASCAL qui permet de permuter entre les deux éléments du vecteur T, d'indice K et L.

Corrigé de l'exercice N°12 :

Algorithme	Pascal

Algorithme permutation ;
Variables
T : Tableau [1..100] de réel ;
i, N, K, L : entier ;
Z : **réel** ;
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donner la taille du vecteur T : ') ;
Lire(N) ;
Ecrire('Donner les composantes du vecteur T : ') ;
Pour i←1 **à** N **faire**
Lire(T[i]) ;
FinPour ;
Ecrire('Donner deux indices K et L entre 1 et',N,' :');
Lire(K,L) ;
{-*-*-*- Traitement -*-*-*-}
Z ← T[K] ;
T[K] ← T[L] ;
T[L] ← Z ;
{-*-*-*- Sorties -*-*-*-}
Pour i←1 **à** N **faire**
Ecrire(T[i]:8:2) ;
FinPour ;
Fin.
	Program permutation ;
	Var
	T : array [1..100] of real ;
	i, N,K, L : integer ;
	Z : **real** ;
	Begin
	{-*-*-*- Entrées -*-*-*-}
	Write('Donner la taille du vecteur T : ') ;
	Read(N) ;
	Writeln('Donner les composantes du vecteur T : ') ;
	For i :=1 **to** N **do**
	Read(T[i]) ;
	Write('Donner deux indices K et L entre 1 et',N,' :');
	Read(K,L) ;
	{-*-*-*- Traitement -*-*-*-}
	Z := T[K] ;
	T[K] := T[L] ;
	T[L] := Z ;
	{-*-*-*- Sorties -*-*-*-}
	For i :=1 **to** N **do**
	Write(T[i]:8:2) ;
	End.



Exercice N°13 : (Produit scalaire de deux vecteurs T1 et T2)

Ecrire un algorithme permettant de calculer et afficher le produit scalaire de deux vecteurs T1 et T2 de N éléments des valeurs réelles. Traduire l'algorithme en Pascal.

Corrigé de l'exercice N°13 :

Algorithme	Pascal

Algorithme Produit_Scalaire ;
Variables
T1, T2 : tableau [1..100] de reel
N, i : entier
PS : reel
Début
Ecrire('Donner la taille du vecteur T1 et T2 : ');
Lire (N) ;
Writeln('Donner les composantes du vecteur T1 : ');
pour i ← 1 **à** N **faire**
Lire (T1[i]) ;
Fin-Pour
Writeln('Donner les composantes du vecteur T2 : ');
pour i ← 1 **à** N **faire**
	Program Produit_Scalaire;
	Var
	T1, T2 : array [1..100] of real;
	N, i : integer;
	PS : real;
	Begin
	Write('Donner la taille du vecteur T1 et T2 : ');
	Read (N);
	Writeln('Donner les composantes du vecteur T1 : ');
	for i:=1 **to** N **do**
	read(T1[i]);
	Writeln('Donner les composantes du vecteur T2 : ');
	for i:=1 **to** N **do**
	read(T2[i]);
	PS:=0;

Lire (T2[i]) ; **Fin-Pour** PS ← 0 ; **pour** i ← 1 **à** N faire PS ← PS + T1[i] * T2[i] ; **Fin-Pour** Ecrire('Le produit scalaire = ', PS:6:2); **Fin**	**for** i:=1 **to** N **do** PS:= PS + T1[i]*T2[i] ; Write('Le produit scalaire = ', PS:6:2); **End.**

```
1  Program Produit_Scalaire;
2  Var
3     T1, T2 : array [1..100] of real;
4     N, i : integer;
5     PS : real;
6  Begin
7     Write('Donner la taille du vecteur T1 et T2 : ');
8     Read (N);
9  Writeln('Donner les composantes du vecteur T1 : ');
10    for i:=1 to N do
11    read(T1[i]);
12 Writeln('Donner les composantes du vecteur T2 : ');
13 for i:=1 to N do
14    read(T2[i]);
15 PS:=0;
16 for i:=1 to N do
17    PS:= PS + T1[i]*T2[i] ;
18 Write('Le produit scalaire = ', PS:6:2);
19 End.
```

```
MyPascal V1.20.5 (Exécution)
Donner la taille du vecteur T1 et T2 : 5
Donner les composantes du vecteur T1 :
4 -5 0.5 5 6
Donner les composantes du vecteur T2 :
5 3 0 6.5 4
Le produit scalaire =  61.50
```

Après l'exécution

Exercice N°14 : (Somme des éléments divisible par 3 d'un vecteur T)

Ecrire un algorithme permettant de calculer et afficher la somme des éléments divisible par 3 d'un vecteur T de type réel et de taille N. Traduire l'algorithme en Pascal.

Corrigé de l'exercice N°14 :

Algorithme	Pascal
Algorithme Vecteur; **Variables** T : Tableau [1..100] d'entier ; N, i, S : entier; **Début** {-*-*-*- Entrées -*-*-*-} Ecrire('Donner la taille du vecteur T : '); Lire(N);	**Program** Vecteur ; **Var** T : array [1..100] of integer; N, i, S : integer; **Begin** {-*-*-*- Entrées -*-*-*-} Write('Donner la taille du vecteur T : ');

```
Ecrire('Donner les composantes du vecteur
        T : ');
Pour i←1 à N faire
    Lire(T[i]);
Fin-Pour;

{-*-*-*- Traitements -*-*-*-}
S ← 0;
Pour i←1 à N faire
    Si (T[i] mod 3 = 0) alors
        S ← S+T[i];
    Fin-Si;
Fin-Pour;

{-*-*-*- Sorties -*-*-*-}
Ecrire('La somme S=', S);
Fin.
```

```
Read(N);
Writeln('Donner les composantes du vecteur
        T : ');
For i:=1 to N do
    Read(T[i]);

{-*-*-*- Traitements -*-*-*-}
S:=0;
For i:=1 to N do
  if (T[i] mod 3 = 0) then
    S:=S+T[i];

{-*-*-*- Sorties -*-*-*-}
Write('La somme S=',S);
End.
```

```
1  Program Vecteur ;
2  Var
3    T : array [1..100] of integer;
4    N, i, S : integer;
5  Begin
6
7    {-*-*-*- Entrées -*-*-*-}
8    Write('Donner la taille du vecteur T : ');
9    Read(N);
10   Writeln('Donner les composantes du vecteur T : ');
11   For i:=1 to N do
12       Read(T[i]);
13
14   {-*-*-*- Traitements -*-*-*-}
15   S:=0;
16   For i:=1 to N do
17     if (T[i] mod 3 = 0) then
18       S:=S+T[i];
19
20   {-*-*-*- Sorties -*-*-*-}
21   Write('La somme S=',S);
22 End.
```

```
MyPascal V1.20.5 (Exécuti
Donner la taille du vecteur T : 6
Donner les composantes du vecteur T :
5 9 6 3 2 3
La somme S=21
```

Après l'exécution

Exercice N°15 : (Inverser les éléments d'un vecteur)

1. Ecrire 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).

Corrigé de l'exercice N°15 :

Question 1 :

Algorithme	Pascal

Algorithme inverser_T_dans_V;
Variables
i, N : **entier**;
V, T : Tableau [1..100] **de réel**;
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donner la taille du vecteur T :');
Lire(N);
Ecrire('Donner les composantes de T :');
Pour i←1 **à** N **faire**
Lire(T[i]);
FinPour
 	
{-*-*-*- Traitement -*-*-*-}
Pour i←1 **à** N **faire**
V[i] ← T[N-i+1];
FinPour
 	
{-*-*-*- Sorties -*-*-*-}
Ecrire('L"inverse de T est V :');
Pour i←1 **à** N **faire**
Ecrire(V[i]:8:2);
FinPour
Fin.
	Program inverser_T_dans_V;
	Var
	i, N : **integer**;
	V, T : **array** [1..100] **of real**;
	Begin
	{-*-*-*- Entrées -*-*-*-}
	Writeln('Donner la taille du vecteur T :');
	Read(N);
	Writeln('Donner les composantes de T :');
	For i := 1 **to** N **do**
	Read(T[i]);
	
	{-*-*-*- Traitement -*-*-*-}
	For i := 1 **to** N **do**
	V[i] := T[N-i+1];
	
	{-*-*-*- Sorties -*-*-*-}
	Writeln('L"inverse de T est V :');
	For i := 1 **to** N **do**
	Write(V[i]:8:2);{Afficher sur 8 espaces avec 2 chiffres après la virgule}
	End.

```pascal
Program inverser_T_dans_V;
 Var
   i, N : integer;
   V, T : array [1..100] of real;
Begin
 {-*-*-*- Entrées -*-*-*-}
 Writeln('Donner la taille du vecteur T :');
 Read(N);
 Writeln('Donner les composantes de T :');
 For i := 1 to N do
    Read(T[i]);

 {-*-*-*- Traitement -*-*-*-}
 For i := 1 to N do
    V[i] := T[N-i+1];

 {-*-*-*- Sorties -*-*-*-}
 Writeln('L''inverse de T est V :');
 For i := 1 to N do
    Write(V[i]:8:2);{Afficher sur 8 espaces avec 2 chiffres après la virgule}
End.
```

```
MyPascal V1.20.5 (Exécution)
Donner la taille du vecteur T :
6
Donner les composantes de T :
4 6 5 0 7 5
L'inverse de T est V :
    5.00    7.00    0.00    5.00    6.00    4.00
```

Après l'exécution

Question 2 :

Algorithme	Pascal

Algorithme inverser_T_dans_T;
Variables
i, N : **entier**;
T : Tableau [1..100] **de réel**;
Z : **réel**;
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donner la taille du vecteur T :');
Lire(N);
Ecrire('Donner les composantes de T :');
Pour i←1 **à** N **faire**
Lire(T[i]);
FinPour ;
 	
{-*-*-*- Traitement -*-*-*-}
Pour i←1 **à** (N div 2) **faire**
Z ← T[i];
T[i] ← T[N-i+1];
T[N-i+1] ← Z;
FinPour ;
 	
{-*-*-*- Sorties -*-*-*-}
	Program inverser_T_dans_T;
	Var
	i, N : **integer**;
	T : **array** [1..100] **of real**;
	Z : **real** ;
	Begin
	{-*-*-*- Entrées -*-*-*-}
	Writeln('Donner la taille du vecteur T :');
	Read(N);
	Writeln('Donner les composantes de T :');
	For i := 1 **to** N **do**
	Read(T[i]);
	
	{-*-*-*- Traitement -*-*-*-}
	For i := 1 **to** (N div 2) **do**
	Begin
	Z := T[i];
	T[i] := T[N-i+1];
	T[N-i+1] := Z;
	End;
	
	{-*-*-*- Sorties -*-*-*-}

Ecrire('L"inverse de T est :'); **Pour** i←1 **à** N **faire** Ecrire(T[i]:8:2); **FinPour ;** **Fin.**	Writeln('L"inverse de T est :'); **For** i := 1 **to** N **do** Write(T[i]:8:2) ; **End.**

```
1  Program inverser_T_dans_T;
2  Var
3    i, N : integer;
4    T : array [1..100] of real;
5    Z : real ;
6  Begin
7    {-*-*-*- Entrées -*-*-*-}
8    Writeln('Donner la taille du vecteur T :');
9    Read(N);
10   Writeln('Donner les composantes de T :');
11   For i := 1 to N do
12     Read(T[i]);
13
14   {-*-*-*- Traitement -*-*-*-}
15   For i := 1 to (N div 2) do
16     Begin
17       Z := T[i];
18       T[i] := T[N-i+1];
19       T[N-i+1] := Z;
20     End;
21
22   {-*-*-*- Sorties -*-*-*-}
23   Writeln('L"inverse de T est :');
24   For i := 1 to N do
25     Write( T[i]:8:2) ;
26 End.
```

```
MyPascal V1.20.5 (Exécution)
Donner la taille du vecteur T :
6
Donner les composantes de T :
4 6 5 0 7 5
L'inverse de T est :
    5.00    7.00    0.00    5.00    6.00    4.00
```

Après l'exécution

Exercice N°16 :

Ecrire un algorithme/programme pascal qui permet de lire un tableau de N éléments réels, qui calcule et affiche le nombre d'éléments égaux au dernier élément lu.

Exemple : pour le tableau suivant : le nombre = 5

4	-3	4	4	6	4	12	4	-9	4

Corrigé de l'exercice N°16 :

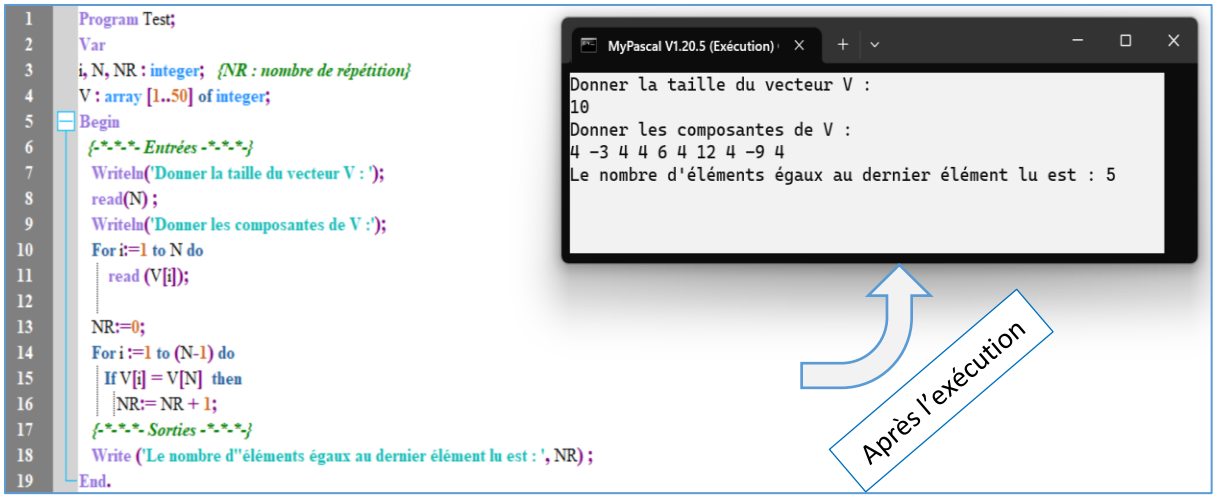
Algorithme	Pascal
Algorithme Test; **Variables** i, N, NR : entier; {NR : nombre de répétition} V : tableau [1..50] de réel; **Début** {-*-*-*- Entrées -*-*-*-} Ecrire('Donner la taille du vecteur V : ');	**Program** Test; **Var** i, N, NR : integer; {NR : nombre de répétition} V : array [1..50] of integer; **Begin** {-*-*-*- Entrées -*-*-*-} Writeln('Donner la taille du vecteur V : ');

```
Lire(N) ;
Ecrire('Donner les composantes de V :');
Pour i ← 1 à N faire
   Lire(V[i]);
FinPour

NR ← 0;
Pour i ← 1 à (N-1) faire
  Si V[i] = V[N]  alors
     NR:= NR + 1;
  FinSi
FinPour
{-*-*-*- Sorties -*-*-*-}
Ecrire ('Le nombre d''éléments égaux au
        dernier élément lu est : ', NR) ;
Fin.
```

```
read(N) ;
Writeln('Donner les composantes de V :');
For i:=1 to N do
   read (V[i]);

NR:=0;
For i :=1 to (N-1) do
  If V[i] = V[N]  then
     NR:= NR + 1;
{-*-*-*- Sorties -*-*-*-}
Write ('Le nombre d''éléments égaux au
        dernier élément lu est : ', NR) ;
End.
```



Exercice N°17 :

Ecrire un algorithme/programme pascal qui permet de lire un tableau T de N éléments réels, qui calcule et affiche le nombre d'éléments égaux au premier élément lu.

Exemple : pour le tableau suivant : le nombre = 5

4	-3	4	4	6	4	12	4	-9	4

Corrigé de l'exercice N°17 :

Algorithme	Pascal

Algorithme Test;
Variables
i, N, NR : entier; {NR : nombre de répétition}
V : tableau [1..50] de réel;
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donner la taille du vecteur V : ');
Lire(N) ;
Ecrire('Donner les composantes de V :');
Pour I ← 1 **à** N **faire**
Lire(V[i]);
FinPour
{-*-*-*- Entrées -*-*-*-}
NR ← 0;
Pour i ← 2 **à** N **faire**
Si V[i] = V[1] **alors**
NR:= NR + 1;
FinSi
FinPour
{-*-*-*- Sorties -*-*-*-}
Ecrire ('Le nombre d''éléments égaux au premier élément lu est : ', NR) ;
Fin.
	Program Test;
	Var
	i, N, NR : integer; {NR : nombre de répétition}
	V : array [1..50] of integer;
	Begin
	{-*-*-*- Entrées -*-*-*-}
	Writeln('Donner la taille du vecteur V : ');
	read(N) ;
	Writeln('Donner les composantes de V :');
	For i:=1 **to** N **do**
	read (V[i]);
	{-*-*-*- Entrées -*-*-*-}
	NR:=0;
	For i :=2 **to** N **do**
	If V[i] = V[1] **then**
	NR:= NR + 1;
	{-*-*-*- Sorties -*-*-*-}
	Write ('Le nombre d''éléments égaux au premier élément lu est : ', NR) ;
	End.

```
1  Program Test;
2  Var
3  i, N, NR : integer;  {NR : nombre de répétition}
4  V : array [1..50] of integer;
5  Begin
6   {-*-*- Entrées -*-*-}
7    Writeln('Donner la taille du vecteur V : ');
8    read(N) ;
9    Writeln('Donner les composantes de V :');
10   For i:=1 to N do
11     read (V[i]);
12   {-*-*- Entrées -*-*-}
13   NR:=0;
14   For i :=2 to N do
15     If V[i] = V[1]  then
16       NR:= NR + 1;
17   {-*-*- Sorties -*-*-}
18   Write ('Le nombre d''éléments égaux au premier élément lu est : ', NR) ;
19 End.
```

```
MyPascal V1.20.5 (Exécution)
Donner la taille du vecteur V :
10
Donner les composantes de V :
4 -3 4 4 6 4 12 4 -9 4
Le nombre d'éléments égaux au premier élément lu est : 5
```

Après l'exécution

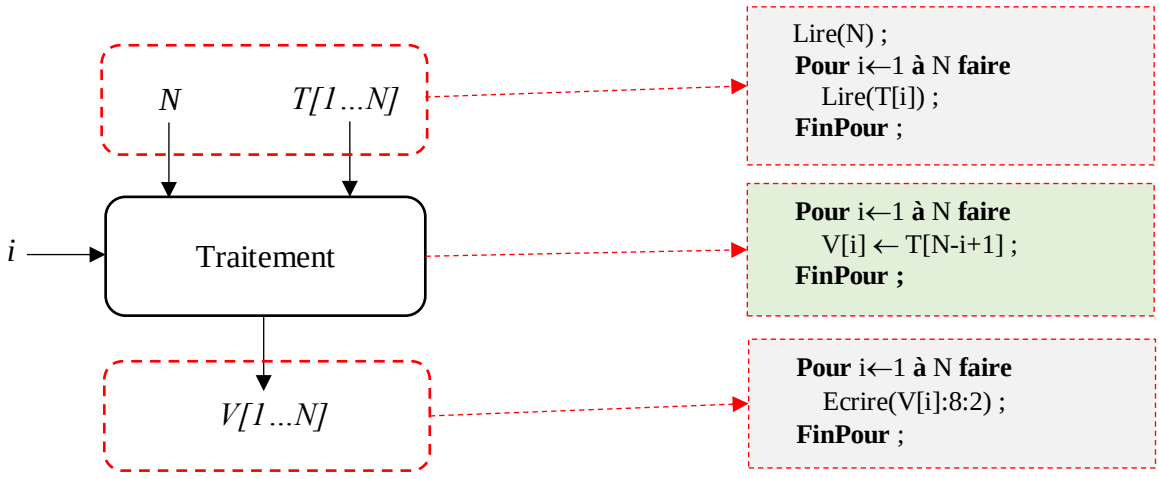
Exercice N°18 : Inverser les éléments d'un vecteur

1. Ecrire 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).

Corrigé de l'exercice N°18 :

Question 01 :

Les variables d'entrée, variable de sorite et la partie traitement sont présentées dans le schéma ci-dessous :



Pour illustrer la partie traitement, nous prenons un exemple :

N = 6, T = [11 13 -8 5 7 22]

Nous devons avoir le vecteur V : V = [22 7 5 -8 13 11]

Ce que nous remarquons : (i allant de 1 à N, avec N=6)

Pour i=1 ➔ V[1]=T[6]
Pour i=2 ➔ V[2]=T[5]
Pour i=3 ➔ V[3]=T[4]
Pour i=4 ➔ V[4]=T[3]
Pour i=5 ➔ V[5]=T[2]
Pour i=6 ➔ V[6]=T[1]

V[1]=T[6-1+1]
V[2]=T[6-2+1]
V[3]=T[6-3+1]
V[4]=T[6-4+1]
V[5]=T[6-5+1]
V[6]=T[6-6+1]

Pour toutes égalités, on peut écrire :

```
Pour i ←1 à N faire
    V[i] ← T[N-i+1] ;
FinPour ;
```

Algorithme	Pascal

Algorithme inverser_T_dans_V;
Variables
i, N : **entier**;
V, T : Tableau [1..100] **de réel**;
Début
{-*-*-*- Entrées -*-*-*-}
Ecrire('Donner la taille du vecteur T :');
Lire(N);
Ecrire('Donner les composantes de T :');
Pour i←1 **à** N **faire**
Lire(T[i]);
FinPour
 	
{-*-*-*- Traitement -*-*-*-}
Pour i←1 **à** N **faire**
V[i] ← T[N-i+1];
FinPour
 	
{-*-*-*- Sorties -*-*-*-}
Ecrire('L"inverse de T est V :');
Pour i←1 **à** N **faire**
Ecrire(V[i]:8:2);
FinPour
Fin.
	Program inverser_T_dans_V;
	Var
	i, N : **integer**;
	V, T : **array** [1..100] **of real**;
	Begin
	{-*-*-*- Entrées -*-*-*-}
	Writeln('Donner la taille du vecteur T :');
	Read(N);
	Writeln('Donner les composantes de T :');
	For i := 1 **to** N **do**
	Read(T[i]);
	
	{-*-*-*- Traitement -*-*-*-}
	For i := 1 **to** N **do**
	V[i] := T[N-i+1];
	
	{-*-*-*- Sorties -*-*-*-}
	Writeln('L"inverse de T est V :');
	For i := 1 **to** N **do**
	Write(V[i]:8:2);{Afficher sur 8 espaces avec 2 chiffres après la virgule}
	End.

```
1  Program inverser_T_dans_V;
2   Var
3     i,N : integer;
4     V,T : Array [1..100] of real;
5  Begin
6   {-*-*-*- Entrées -*-*-*-}
7   Writeln('Donner la taille du vecteur T :');
8   Read(N);
9   Writeln('Donner les composantes de T :');
10   For i := 1 to N do
11     Read( T[i] );
12
13   {-*-*-*- Traitement -*-*-*-}
14   For i := 1 to N do
15     V[i] := T[N-i+1];
16
17   {-*-*-*- Sorties -*-*-*-}
18   Writeln('L''inverse de T est V :');
19   For i := 1 to N do
20     Write(V[i]:8:2);{Afficher sur 8 espaces avec 2 chiffres après la virgule}
21  End.
```

```
MyPascal V1.20.5 (Exécution) C:\Users\Ahmed\Des...
Donner la taille du vecteur T :
6
Donner les composantes de T :
11  13  -8  5  7 22
L'inverse de T est V :
   22.00    7.00    5.00   -8.00   13.00   11.00
```

Après l'exécution

Question 02 : Inverser le vecteur T dans lui même

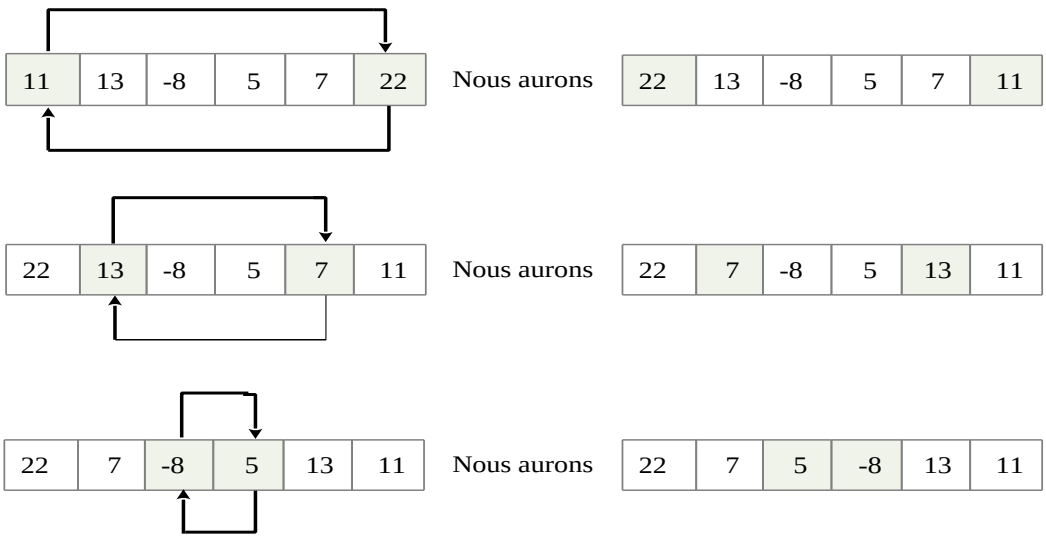
Le même problème que la question 01, sauf que, dans cette deuxième question, nous voulons inverser le vecteur T dans lui-même.

Explication :

On reprend le même exemple précédent :

N = 6, T = [11 13 -8 5 7 22]

Pour inverser les éléments de T, dans le même vecteur, nous allons faire les permutations suivantes :



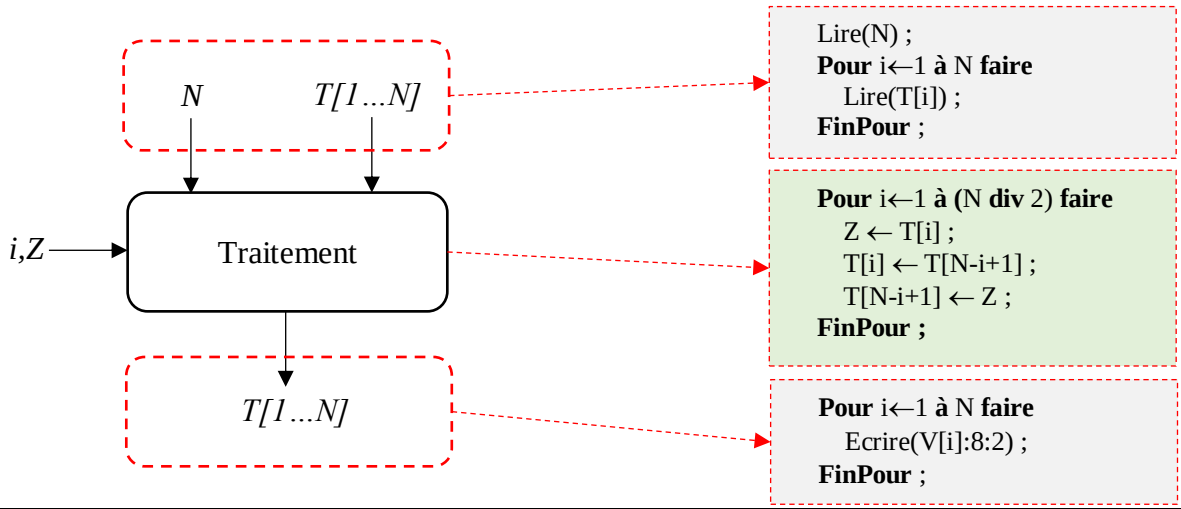
Après la troisième permutation, nous aurons le résultat demandé (inverser le vecteur T dans lui-même).

Remarques à retenir :

- Inverser un vecteur dans lui-même en permutant des cases.
- Le nombre de permutations est la moitié du vecteur.
- Les permutations : (1 avec N), (2 avec N-1), … Jusqu'à (N **div** 2).
- De la question 1, nous déduisons que : la case N° i sera permutée avec la case N° (N-i+1), tel-que i = 1… (N **div** 2).
- Pour permuter entre les cases *i* et (N-i+1), nous utilisons une troisième variable Z, comme suit:

```
Z ←T[i]
T[i] ← T[N-i+1]
T[N-i+1] ← Z
Pour chaque valeur de i allant de 1 à (N div 2)
```

Les variables d'entrée, variable de sorite et la partie traitement sont présentées dans le schéma ci-dessous :



Algorithme	Programme PASCAL
Algorithme inverser_T_dans_T; **Variables** i, N : **entier**; T : Tableau [1..100] **de réel**; Z : **réel**; **Début** {-*-*-*- Entrées -*-*-*-} Ecrire('Donner la taille du vecteur T :'); Lire(N);	**Program** inverser_T_dans_T; **Var** i, N : **integer**; T : **array** [1..100] **of real**; Z : **real** ; **Begin** {-*-*-*- Entrées -*-*-*-} Writeln('Donner la taille du vecteur T :'); Read(N);

Ecrire('Donner les composantes de T :');
Pour i←1 **à** N **faire**
Lire(T[i]);
FinPour ;
 	
{-*-*-*- Traitement -*-*-*-}
Pour i←1 **à** (N div 2) **faire**
Z ← T[i];
T[i] ← T[N-i+1];
T[N-i+1] ← Z;
FinPour ;
 	
{-*-*-*- Sorties -*-*-*-}
Ecrire('L"inverse de T est :');
Pour i←1 **à** N **faire**
Ecrire(T[i]:8:2);
FinPour ;
Fin.
	Writeln('Donner les composantes de T :');
	For i := 1 **to** N **do**
	Read(T[i]);
	
	{-*-*-*- Traitement -*-*-*-}
	For i := 1 **to** (N div 2) **do**
	Begin
	Z := T[i];
	T[i] := T[N-i+1];
	T[N-i+1] := Z;
	End;
	
	{-*-*-*- Sorties -*-*-*-}
	Writeln('L"inverse de T est :');
	For i := 1 **to** N **do**
	Write(T[i]:8:2) ;
	End.

```
1  Program inverser_T_dans_T;
2  Var
3    i,N : integer;
4    T : Array [1..100] of real;
5    Z : real ;
6  Begin
7    {-*-*-*- Entrées -*-*-*-}
8    Writeln('Donner la taille du vecteur T :');
9    Read(N);
10   Writeln('Donner les composantes de T :');
11   For i := 1 to N do
12     Read( T[i] );
13
14   {-*-*-*- Traitement -*-*-*-}
15   For i := 1 to (N div 2) do
16     Begin
17       Z := T[i];
18       T[i] := T[N-i+1];
19       T[N-i+1] := Z;
20     End;
21
22   {-*-*-*- Sorties -*-*-*-}
23   Writeln('L"inverse de T est :');
24   For i := 1 to N do
25     Write( T[i]:8:2) ;
26 End.
```

```
MyPascal V1.20.5 (Exécution) C:\Users\Ahmed\Deskt...
Donner la taille du vecteur T :
6
Donner les composantes de T :
11 13 -8 5 7 22
L'inverse de T est :
   22.00    7.00    5.00   -8.00   13.00   11.00
```

Après l'exécution

I.6. Exercices supplémentaires

Exercice supplémentaire 01 : La recherche d'une valeur dans un vecteur.

Soit V un vecteur de type réel de taille N.
Ecrire 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.

Exercice supplémentaire 02 : Somme, Produit et compteur d'éléments

Soit V un vecteur de type réel et de taille N.
Écrire un algorithme / Programme PASCAL qui permet de :
– Réaliser la somme des éléments divisibles par 3 et non divisible par 4.
– Réaliser le produit des éléments divisible par 4 et non divisible par 3.
– Compter le nombre d'éléments non-divisibles par 3 et non-divisibles par 4.

Exercice supplémentaire 03 : Convertir un nombre de base 10 vers base 2

Soit Nb un nombre entier positif écrit en base 10.
Ecrire un algorithme / programme PASCAL qui permet de convertir la valeur de Nb en base 2 et d'enregistrer les chiffres binaires de Nb dans un vecteur T.

Exercice supplémentaire 04 :

Ecrire un algorithme permettant de supprimer une case dont la position est lue à partir du clavier dans un tableau de dix entiers. La case supprimée sera substituée par un zéro ajouté à la fin du tableau.

Par exemple, après la suppression de la troisième case du tableau : 3, 15, 8, 18, 34, 1, 0, 4, 5, 21, le tableau devient : 3, 15, 18, 34, 1, 0, 4, 5, 21, 0.
Traduire l'algorithme en Pascal.

Exercice supplémentaire 05 :

Ecrire un algorithme permettant de résoudre le problème suivant :
– Données : un tableau contenant 100 entiers
– Résultat : "vrai" si le tableau est trié du plus petit au plus grand et "faux" sinon

Exercice supplémentaire 06 :

Ecrire un algorithme permettant de saisir 100 valeurs et qui les range au fur et à mesure dans un tableau.

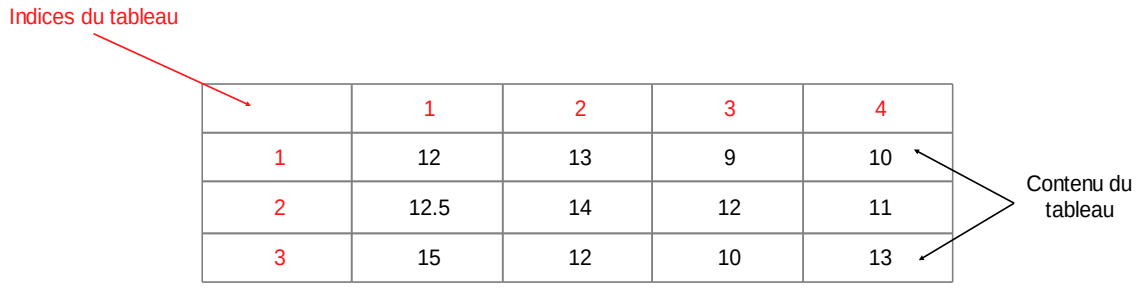
I.7. Tableaux à deux dimensions

Reprenons l'exemple des notes en considérant cette fois qu'un étudiant a plusieurs notes (une note pour chaque matière). Pour quatre étudiants, nous aurions le tableau de relevés des notes suivant :

	Etudiant 1	Etudiant 2	Etudiant 3	Etudiant 4
Informatique	12	13	9	10
Physique	12.5	14	12	11
Mathématiques	15	12	10	13

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

Nous pouvons représenter schématiquement un tableau de 3 lignes et de 4 colonnes comme suit :



I.7.1. Déclaration

Syntaxe :

Algorithme

```
Variable
Variable identificateur : tableau [1..nb_lignes, 1..nb_colonnes] de type ;
Ou bien
Variable
Variable identificateur : tableau [nb_lignes, nb_colonnes] de type ;
```

Pascal

```
Var
Variable identificateur : array [1..nb_lignes, 1..nb_colonnes] of type ;
Ou bien
Var
Variable identificateur : array [nb_lignes, nb_colonnes] of type ;
```

Exemple :

L'instruction suivante déclare un tableau Note de type réel à deux dimensions composé de 3 lignes et de 4 colonnes :

Algorithme	Pascal
Variable Note : tableau [1..3, 1..4] de réels ;	**Var** Note : array [1..3, 1..4] of real ;

I.7.2. Utilisation

Pour accéder à un élément de la matrice (tableau à deux dimensions), il suffit de préciser, entre crochets, les indices de la case contenant cet élément.

Les éléments de la matrice peuvent être utilisés comme n'importe quelle variable.

Exemple :

L'instruction suivante affecte à la variable X la valeur du premier élément du tableau Note :

Algorithme	Pascal
X ← Note [1,1] ;	X := Note [1,1] ;

I.7.3. Lecture et affichage d'une matrice

L'algorithme (respectivement, le programme) suivant permet de lire et d'écrire une matrice de *n* ligne et *m* colonnes :

Algorithme	Pascal

Algorithme LectureAffichageMatrice
Variables
mat : tableau [1..10, 1..10] de reel
N,M, i,j:entier
Début
Lire (N, M);
pour i=1 **à** N **faire**
Pour j=1 **à** M **faire**
Lire (mat[i, j])
FinPour
FinPour
pour i=1 **à** N **faire**
Pour j=1 **à** M **faire**
Ecrire (mat[i, j])
	Program LectureAffichageMatrice;
	var
	mat : **array** [1..10, 1..10] **of** real;
	N,M, i,j : integer;
	Begin
	Read(N, M);
	For i:=1 **to** N **do**
	For j:=1 **to** M **do**
	Read(mat[i, j]);
	For i:=1 **to** N **do**
	Begin
	For j:=1 **to** M **do**
	Begin
	Write(mat[i,j],' ');

FinPour **FinPour** **Fin.**	**End**; writeln; **End**; **End.**

I.8. Exercices avec corrigés

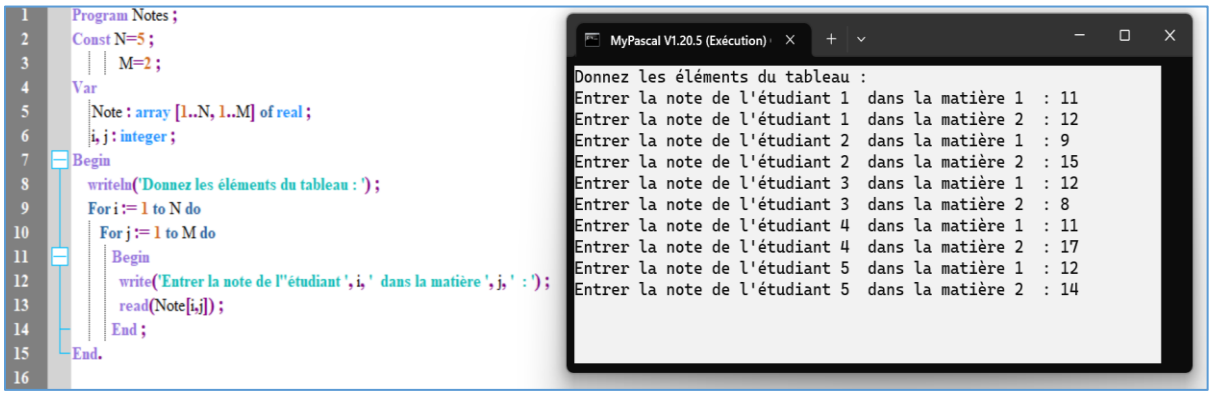
Exercice N°19 :

Ecrire un algorithme permettant la saisie des notes d'une classe de 5 étudiants en 2 matières.

Corrigé de l'exercice N°19 :

Algorithme	Pascal

Algorithme Notes ;
Constante N=5 ;
M=2 ;
Variables
Note : tableau [1..N, 1..M] de réels ;
i, j : entier ;
Début
Ecrire('Donnez les éléments du tableau : ') ;
Pour i ← 1 **à** N **faire**
Pour j ← 1 **à** M **faire**
Ecrire('Entrer la note de l''étudiant ', i, '
dans la matière', j, ' : ') ;
Lire(Note[i,j]) ;
Fin-Pour ;
Fin-Pour ;
Fin.
	Program Notes ;
	Const N=5 ;
	M=2 ;
	Var
	Note : array [1..N, 1..M] of real ;
	i, j : integer ;
	Begin
	writeln('Donnez les éléments du tableau : ') ;
	For i := 1 **to** N **do**
	For j := 1 **to** M **do**
	Begin
	write('Entrer la note de l''étudiant ', i, '
	dans la matière ', j, ' : ') ;
	read(Note[i,j]) ;
	End ;
	End.



Exercice N°20 :

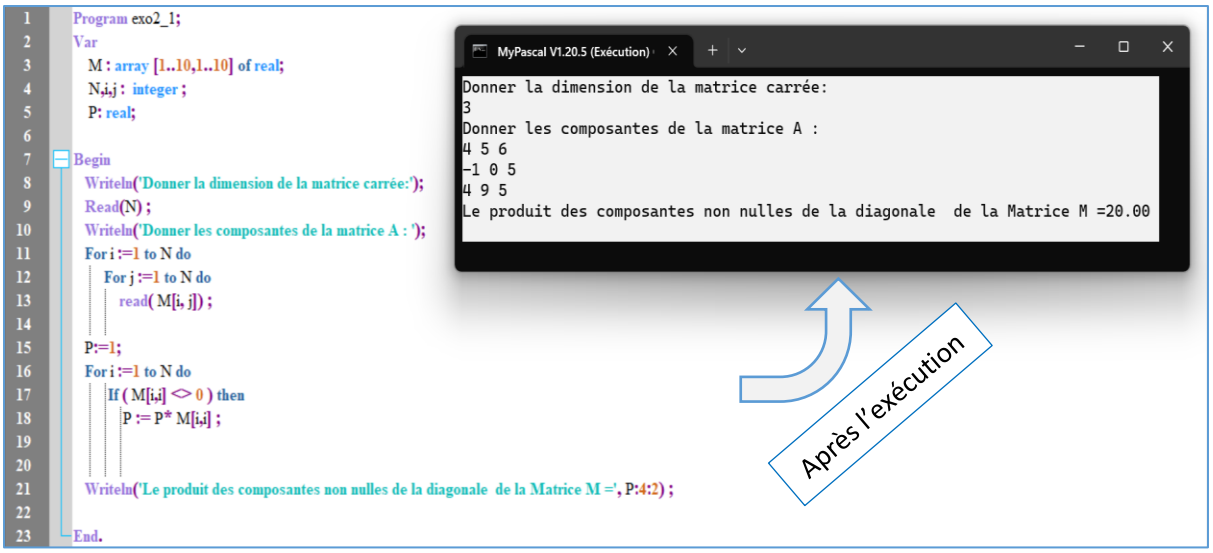
Soit M une matrice carrée de taille N x N et de type réel.

Écrire un programme pascal qui permet de calculer le produit des composantes non nulles de la diagonale principale de la matrice M.

Corrigé de l'exercice N°20 :

Algorithme	Pascal

Algorithme exo2_1;
Variables
M : tableau [1..10,1..10] de réel;
N,i,j : entier ;
P: réel;
 	
Début
Ecrire('Donner la dimension de la matrice carrée:');
Lire(N) ;
Ecrire('Donner les composantes de la matrice A : ');
Pour i ←1 **à** N **faire**
Pour j ←1 **à** N **faire**
Lire(M[i, j]) ;
Fin-Pour;
Fin-Pour;
 	
P ← 1 ;
Pour i←1 **à** N **faire**
Si (M[i,i] <> 0) **alors**
P ← P* M[i,i] ;
Fin-si;
Fin-Pour;
 	
Écrire ('Le produit des composantes non nulles de la diagonale de la Matrice M = ', P:4:2) ;
Fin.
	Program exo2_1;
	Var
	M : array [1..10,1..10] of real;
	N,i,j : integer ;
	P: real;
	
	Begin
	Writeln('Donner la dimension de la matrice carrée:');
	Read(N) ;
	Writeln('Donner les composantes de la matrice A : ');
	For i :=1 **to** N **do**
	For j :=1 **to** N **do**
	read(M[i, j]) ;
	
	P:=1;
	For i :=1 **to** N **do**
	If (M[i,i] <> 0) **then**
	P := P* M[i,i] ;
	
	Writeln('Le produit des composantes non nulles de la diagonale de la Matrice M =', P:4:2) ;
	
	End.



Exercice N°21 :

Soit A une matrice réelle d'ordre N x M.

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

Corrigé de l'exercice N°21 :

Algorithme	Pascal

Algorithme exo2_1
Variables
A : Tableau [1..10, 1..10] **de** Réel
i, j, N, M, imin, jmin : entier
Min : Réel
Début
Ecrire('Donner les dimensions de la Matrice A : ');
Lire(N, M)
Ecrire('Donner les composantes de la matrice A : ');
Pour i ←1 **à** N **faire**
Pour j ←1 **à** M **faire**
Lire(A[i, j])
Fin-Pour
Fin-Pour
min ← A[1, 1]
	Program exo2_1;
	Var
	A : array[1..10, 1..10] **of** Real;
	i, j, N, M, imin, jmin : integer;
	min : real;
	Begin
	Write('Donner les dimensions de la Matrice A : ');
	Read(N, M);
	Writeln('Donner les composantes de la matrice A : ');
	For i:=1 **to** N do
	For j:=1 **to** M **do**
	Read(A[i, j]);
	
	min := A[1, 1]; {On suppose que la première case est le min}

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
Ecrire('min=', min:3:2, 'et sa position est : ',
imin,' ',jmin);
Fin
	imin:= 1; {Donc sa position est la ligne 1}
	jmin := 1; {et la colonne 1}
	For i:=1 **to** N **do** {Pour toute ligne i}
	For j:=1 **to** M **do** {Pour toute colonne j}
	if A[i, j] < min **Then**
	Begin
	min := A[i,j]; {On actualise le Min}
	imin := i; {On actualise sa ligne imin}
	jmin := j; {On actualise sa colonne jmin}
	end;
	{*-* Sorties *-*}
	Write('min=', min:3:2, 'et sa position est : ',
	imin,' ',jmin);
	End.

```
1  Program exo2_1;
2  Var
3    A : array[1..10, 1..10] of Real;
4    i, j, N, M, imin, jmin : integer;
5    min : real;
6  Begin
7   Write('Donner les dimensions de la Matrice A : ');
8   Read(N, M);
9   Writeln('Donner les composantes de la matrice A : ');
10  For i:=1 to N do
11   For j:=1 to M do
12    Read( A[i, j] );
13  min := A[1, 1]; {On suppose que la première case est le min}
14  imin:= 1; {Donc sa position est la ligne 1}
15  jmin := 1; {et la colonne 1}
16  For i:=1 to N do {Pour toute ligne i}
17   For j:=1 to M do {Pour toute colonne j}
18    if A[i, j] < min Then
19     Begin
20      min := A[i,j]; {On actualise le Min}
21      imin := i; {On actualise sa ligne imin}
22      jmin := j; {On actualise sa colonne jmin}
23     end;
24  Write('min=', min:3:2, 'et sa position est : ', imin,'  ',jmin);
25  End.
```

```
MyPascal V1.20.5 (Exécution)
Donner les dimensions de la Matrice A : 3 4
Donner les composantes de la matrice A :
2 3 5 6
-4 5 9 7
1 2 3 11
min=-4.00et sa position est : 2    1
```

Après l'exécution

Exercice N°22 :

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

Corrigé de l'exercice N°22 :

Algorithme	Pascal

Algorithme exol;
Variables
A : Tableau [1..100,1..100] de réels;
N,M,i,j : Entier;
 	
S:Tableau[1..100] de réels; {On déclare S comme vecteur qui contient la somme de chaque ligne parce que on ne connait pas la taille de la matrice A}
 	
P:Tableau[1..100] de réels; {On déclare P comme vecteur qui contient le produite chaque colonne parce que on ne connait pas la taille de la matrice A}
Début {Début du programme}
{*-* Entrées *-*}
Ecrire('Donner la taille de la matrice A');
Lire(N,M);
Ecrire('Donner les composantes de la matrice A');
Pour i ←1 **à** N **faire**
Pour j ←1 **à** M **faire**
Lire(A[i, j])
Fin-Pour
Fin-Pour
{*-* Traitement *-*}
 	
Pour i ←1 **à** N **faire**
S[i] ←0;
Pour j ←1 **à** M **faire**
	Program exol;
	Var
	A : array [1..100,1..100] of real;
	N,M,i,j : integer;
	
	S:array[1..100] of real; {On déclare S comme vecteur qui contient la somme de chaque ligne parce que on ne connait pas la taille de la matrice A}
	
	P:array[1..100] of real; {On déclare P comme vecteur qui contient le produite chaque colonne parce que on ne connait pas la taille de la matrice A}
	Begin {Début du programme}
	{*-* Entrées *-*}
	Writeln('Donner la taille de la matrice A');
	Read(N,M);
	Writeln('Donner les composantes de la matrice A');
	For i:=1 **to** N **do**
	For j:=1 **to** M **do**
	Read(A[i,j]);
	
	{*-* Traitement *-*}
	For i:=1 **to** N **do**
	begin
	S[i]:=0;
	For j:=1 **to** M **do**
	S[i]:=S[i]+A[i,j];

```
            S[i] ← S[i]+A[i,j];
        Fin-Pour
    Fin-Pour

    Pour j ←1 à M faire
        P[j] ← 1;
        For i ←1 to N do
            P[j] ←P[j]*A[i,j];
        Fin-Pour
    Fin-Pour
    {*-* Sorties *-*}
    Ecrire('La somme de chaque ligne est : ');
    Pour i ←1 à N faire
      Ecrire(S[i]:6:2);
    FinPour


    Ecrire('Le produit de chaque colonne est : ');
     Pour j ←1 à M faire
        Ecrire(P[j]:6:2);
Fin.
```

```
    End;

    For j:=1 to M do
    begin
        P[j]:=1;
        For i:=1 to N do
          P[j]:=P[j]*A[i,j];
    End;


  {*-* Sorties *-*}
  Writeln('La somme de chaque ligne est : ');
  For i:=1 to N do
  Write(S[i]:6:2);

  Writeln; {J'ai ajouté cette instruction juste
            pour sauter la ligne (ni pas
            obligatoire)}
  Writeln('Le produit de chaque colonne est : ');
    For j:=1 to M do
      Write(P[j]:6:2);
End. {Fin du programme}
```

Exercice N°23 :

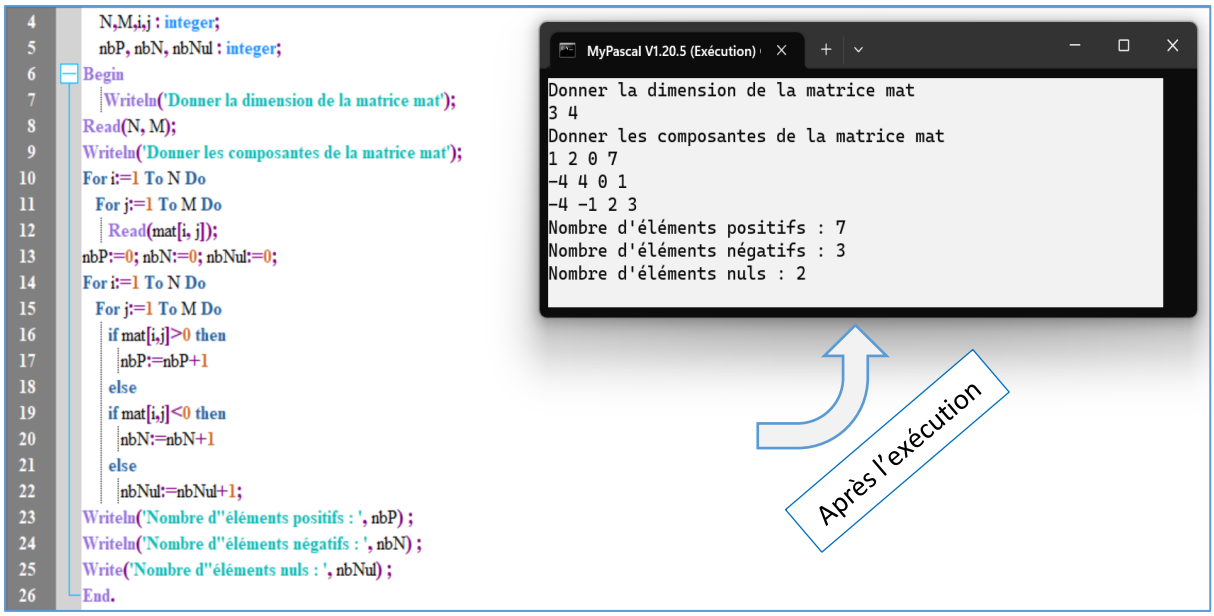
Écrire un algorithme/programme PASCAL qui permet de compter le nombre d'éléments négatifs, positifs et nuls dans une matrice.

Corrigé de l'exercice N°23 :

Algorithme	Pascal
Algorithme elements_P_N_N ;	**Program** elements_P_N_N ;
Variables	**var**
mat : tableau [1..10, 1..10] de réel ;	mat :**array** [1..10, 1..10] **of** real;
N,M,i,j:entier ;	N,M,i,j : integer;
nbP, nbN, nbNul : entier ;	nbP, nbN, nbNul : integer;
Début	**Begin**

```
Ecrire('Donner la dimension de la matrice
        mat');
Lire (N, M);
Ecrire('Donner les composantes de la
        matrice mat');
pour i ← 1 à N faire
  Pour j ← 1 à M faire
    Lire (mat[i, j])
  FinPour
FinPour
nbP ← 0; nbN ← 0; nbNul ← 0;
pour i ← 1 à N faire
  Pour j ← 1 à M faire
    Si mat[i, j] > 0 Alors
      nbP ← nbP + 1;
    Sinon
    Si mat[i,j] < 0 Alors
      nbN ← nbN + 1;
    Sinon
      nbNul ← nbNul + 1;
    FinSi
    FinSi
  FinPour
 FinPour
 Ecrire('Nombre d''éléments positifs : ', nbP) ;
 Ecrire('Nombre d''éléments négatifs : ',
nbN) ;
 Ecrire('Nombre d''éléments nuls : ', nbNul) ;
Fin.
```

```
Writeln('Donner la dimension de la matrice
        mat');
Read(N, M);
Writeln('Donner les composantes de la
        matrice mat');
For i:=1 To N Do
  For j:=1 To M Do
    Read(mat[i, j]);
nbP:=0; nbN:=0; nbNul:=0;
For i:=1 To N Do
  For j:=1 To M Do
    if mat[i,j]>0 then
      nbP:=nbP+1
    else
    if mat[i,j]<0 then
      nbN:=nbN+1
    else
      nbNul:=nbNul+1;
 Writeln('Nombre d''éléments positifs : ', nbP) ;
 Writeln('Nombre d''éléments négatifs : ',
nbN) ;
 Write('Nombre d''éléments nuls : ', nbNul) ;
End.
```



Exercice N°24 :

Dans le cadre d'étude du comportement thermomécanique d'un alliage métallique, l'étudiant a besoin de déterminer la dureté de chacune de ces **5** éprouvettes. Pour chaque éprouvette, il mesure la dureté **3** fois. Il note ses résultats dans un tableau (T_b) en brouillon puis il dessine sur son compte rendu un nouveau tableau (T_f) en une seule ligne contenant la moyenne des **3** duretés mesurées pour chacune des éprouvettes.

Ecrire un algorithme qui permet de saisir, traiter et afficher ces deux tableaux $(T_b \text{ et } T_f)$ de mesure de dureté.

Corrigé de l'exercice N°24 :

Algorithme	Pascal
Algorithme mesure_durete ;	**program** mesure_durete ;
Variables	**Var**
Tb : tableau [1..5,1..3] de réels ;	Tb : array [1..5,1..3] of real ;
Tf : tableau [1..5] de réels ;	Tf : array [1..5] of real ;
S : réel ;	S : real ;
i, j :entier ;	i, j :integer ;
Début	**Begin**
Pour i **de** 1 **à** 5 **faire**	**For** i := 1 **to** 5 **do**
S ← 0 ;	**Begin**
Pour j **de** 1 **à** 3 **faire**	S := 0 ;
Ecrire('Introduisez la',j, 'ème mesure de	**For** j :=1 **to** 3 **do**

```
                la',i, 'ème éprouvette') ;
        Lire(Tb[i,j]);
        S← S+Tb[i,j];
      Fin pour
      Tf[i] ← S/3 ;
   Fin pour
   Ecrire('Le tableau des résultats bruts est le
            suivant : ') ;
   Pour i de 1 à 5 faire
     Pour j de 1 à 3 faire
       Écrire(Tb[i,j]:8:2) ;
     Fin pour
   Fin pour
   Ecrire('Le tableau final des résultats est le
            suivant : ') ;
   Pour i de 1 à 5 faire
     Écrire(Tf[i]:8:2) ;
   Fin pour
Fin.
```

```
Begin
    write('Introduisez la ',j, ' ème mesure de
          la',i, ' ème éprouvette :') ;
    read(Tb[i,j]);
    S := S+Tb[i,j];
  End;
  Tf[i] := S/3 ;
End ;
writeln('Le tableau des résultats bruts est le
          suivant : ') ;
For i :=1 to 5 do
  For j := 1 to 3 do
    write(Tb[i,j]:8:2) ;
writeln ; {Pour sauter la ligne}
writeln('Le tableau final des résultats est le
          suivant : ') ;
For i :=1 to 5 do
  write(Tf[i]:8:2) ;
End.
```

Exercice N°25 :

Ecrire un programme pascal qui permet de calculer et afficher la somme des éléments situés au-dessus de la diagonale d'une matrice carrée.

Corrigé de l'exercice N°25 :

Pascal (méthode 1)

```
Program Somme_elements_dessus_diagonale;
Var
A : array [1..50,1..50] of integer;
Somme, i, j, N  : integer;
Begin
  {-*-*-*- Entrées -*-*-*-}
  read(N) ;
  For i:=1 to N do
    For j:=1 to N do
      read (A[i, j]);
```

Pascal (méthode 2)

```
Program Somme_elements_dessus_diagonale;
Var
A : array [1..50,1..50] of integer;
Somme, i, j, N  : integer;
Begin
  {-*-*-*- Entrées -*-*-*-}
  read(N) ;
  For i:=1 to N do
    For j:=1 to N do
      read (A[i, j]);
```

```
{-*-*-*- Entrées -*-*-*-}
Somme:=0;
For i :=1 to N-1 do
  For j :=i+1 to N do
     Somme:= Somme + A[i,j];
{-*-*-*- Sorties -*-*-*-}
Write ('La somme des éléments situés au-
        dessus de la diagonale est : ', Somme)
        ;
End.
```

```
{-*-*-*- Entrées -*-*-*-}
Somme:=0;
For i :=1 to N do
  For j :=1 to N do
    If (i<j)  then
      Somme:= Somme + A[i,j];
{-*-*-*- Sorties -*-*-*-}
Write ('La somme des éléments situés au-
        dessus de la diagonale est : ', Somme)
        ;
End.
```

Exercice N°26 :

Ecrire un programme pascal qui permet de calculer et afficher la somme des éléments situés en dessous de la diagonale d'une matrice carrée.

Corrigé de l'exercice N°26 :

Pascal (méthode 1)

```
Program Somme_elements_dessus_diagonale;
Var
A : array [1..50,1..50] of integer;
Somme, i, j, N  : integer;
Begin
  {-*-*-*- Entrées -*-*-*-}
  read(N) ;
  For i:=1 to N do
    For j:=1 to N do
      read (A[i, j]);
  {-*-*-*- Entrées -*-*-*-}
  Somme:=0;
  For i :=2 to N do
    For j :=1 to i-1 do
       Somme:= Somme + A[i,j];
  {-*-*-*- Sorties -*-*-*-}
  Write ('La somme des éléments situés au-
          dessus de la diagonale est : ', Somme);
```

Pascal (méthode 2)

```
Program Somme_elements_dessous_diagonale;
Var
A : array [1..50,1..50] of integer;
Somme, i, j, N  : integer;
Begin
  {-*-*-*- Entrées -*-*-*-}
  read(N) ;
  For i:=1 to N do
    For j:=1 to N do
      read (A[i, j]);
  {-*-*-*- Entrées -*-*-*-}
  Somme:=0;
  For i :=1 to N do
    For j :=1 to N do
      If (i>j)  then
        Somme:= Somme + A[i,j];
  {-*-*-*- Sorties -*-*-*-}
```

End.	Write ('La somme des éléments situés au-dessus de la diagonale est : ', Somme) ; **End.**

I.9. Exercices supplémentaires

Exercice supplémentaire 01 : Somme de deux matrices

Ecrire un algorithme/programme PASCAL qui permet de réaliser la somme de deux matrices réelles A et B d'ordre N x M.

Exercice supplémentaire 02 : Somme, Moyenne et Produit des éléments d'une matrice

Soit une matrice A réelle d'ordre N x M.

1. Ecrire un algorithme/programme PASCAL qui calcule la somme et la moyenne des éléments de la matrice A.
2. Ecrire un algorithme/programme PASCAL qui permet de calculer la somme de chaque ligne et le produit de chaque colonne.

Exercice supplémentaire 03 : Produit de deux matrices

Soit A et B deux matrices carrées d'ordre N.

Ecrire un algorithme/programme PASCAL qui permet de calculer le produit de A et B.

Exercice supplémentaire 04 : La recherche d'une valeur dans une matrice

Soit M une matrice de type réel de taille N x M.

Ecrire un algorithme/programme PASCAL qui permet de rechercher si une valeur réelle X existe ou non dans la matrice M. Dans le cas où X existe dans M, on affiche aussi sa position (numéro de ligne et de colonne).

Exercice supplémentaire 05 : Le Min et le Max dans une matrice et leurs positions

Soit A une matrice réelle d'ordre N x M.

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

Exercice supplémentaire 06 :

Partie A : les vecteurs

Écrire un seul programme en Pascal qui permet de traiter les notes d'une classe d'étudiants. Le programme doit effectuer les opérations suivantes :

1) Saisir le nombre d'étudiants N (en supposant que $N \leq 30$)
2) Saisir les N notes des étudiants (les notes sont des réels compris entre 0 et 20).
3) Afficher toutes les notes saisies.
4) Calculer et afficher :
   - La moyenne générale de la classe.
   - Le nombre d'étudiants ayant réussi (note ≥ 10)
   - Le nombre d'étudiants ayant échoué (note < 10)

Partie B : Les matrices

Ecrire un seul programme en pascal qui permet de :

- Demander à l'utilisateur de saisir une matrice de dimension $\boldsymbol{N \times M}$
- Calculer le produit des éléments de chaque ligne de la matrice
- Afficher le produit de chaque ligne

CHAPITRE II

Les fonctions et procédures

Chapitre II : Les fonctions et procédures

II.1. Objectif de ce chapitre

A l'issu de ce chapitre, l'apprenant sera capable de :

- Comprendre le rôle des sous-programmes pour structurer un programme en Pascal.
- Différencier procédures et fonctions en expliquant leurs particularités et usages.
- Déclarer, définir et appeler des sous-programmes en respectant la syntaxe de Pascal.
- Gérer le passage de paramètres (par valeur et par référence) pour améliorer la modularité du code.

II.2. 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 d'un seul tenant 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 : les procédures et les fonctions.

Exemple :

calculer le nombre de combinaisons de k éléments à partir de n élément (n et k deux nombres naturels) :

$$C_n^k = \frac{n!}{k! \times (n-k)!} \quad \text{si } k \leq n$$

On remarque qu'il y a trois factoriels à calculer. Si on n'utilise pas de sous programmes (fonctions ou procédure), on risque de répéter dans le programme des séquences semblables d'instructions (redondance) ce qui nuit à la clarté du programme, sa lisibilité et contiendra plusieurs séquences semblables d'instructions (dans l'exemple, le nombre d'instructions est multiplié par 3).

Afin de rendre le programme plus lisible et réduire le nombre d'instruction (moins d'espace mémoire), on fait appel à des fonctions et procédures (sous programmes).

Dans le cas précédent, du moment que les trois factoriels se calculent de la même façon, on écrit une seule fonction FACT qui permet de calculer le factoriel d'une valeur naturelle quelconque n (n!=1×2×3×... ×n).

Pour calculer le nombre de combinaisons C_n^k, il suffira d'appeler la fonction FACT trois fois en transmettant n, k et (n-k) comme paramètre à la fonction.

II.3. Les fonctions

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

II.3.1. Déclaration d'une fonction

Syntaxe :

```
Fonction nom_Fonct(liste de paramètres) : type
Variables identificateurs : type
Début
   Instruction(s)
   Retourner Expression
FinFonct
```

La syntaxe de la déclaration d'une fonction est assez proche de celle d'une procédure à laquelle on ajoute un type qui représente le type de la valeur retournée par la fonction et une instruction retourner Expression. Cette dernière instruction renvoie au programme appelant le résultat de l'expression placée à la suite du mot clé Retourner.

Note :

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

Exemple :

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

Solution :

```
{Déclaration de la fonction Max}
Fonction Max(X :réel, Y :réel) :réel
Début
   Si X>Y alors
      Retourner X
   Sinon
      Retourner Y
   FinSi
FinFonction
```

II.3.2. L'appel d'une fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivi des paramètres effectifs. C'est la même syntaxe qu'une procédure.

A la différence d'une procédure, la fonction retourne une valeur. L'appel d'une fonction pourra donc être utilisé dans une instruction (affichage, affectation,…) qui utilise sa valeur.

Syntaxe :

```
Nom_Fonction(liste de paramètres)
```

Exemple :

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

Solution :

```
Algorithme Appel_fonction_Max
Variables
A, B, M : réel
Début
   Si X>Y alors
      Retourner X
   Sinon
      Retourner Y
   FinSi
FinFonction
   {Algorithme principal}
Début
   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
```

II.3.3. Portée des variables

La portée d'une variable désigne le domaine de visibilité de cette variable.

Une variable peur être déclarée dans deux emplacements distincts.

Une variable déclarée dans la partie déclaration de l'algorithme principal 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 est 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 la procédure.

Exemple :

```
Algorithme Portée
Variables
X, Y : Entier
Procédure P1()
Variables A :Entier
Début
  …
FinProc
{Algorithme principale}
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.

Remarque :

Les variables globales sont à éviter pour la maintenance des programmes.

II.4. Les procédures

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

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

II.4.1. Déclaration d'une procédure

Syntaxe :

```
Procédure nom_proc (liste de paramètres)
Variable identificateurs : type
Début
    Instruction(s)
Fin Proc
```

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

Exemple :

Ecrire une procédure qui affiche à l'écran une ligne de 15 étoiles puis passe à la ligne suivante.

Solution :

```
Procédure Etoiles()
  Variables
  i : entier
Début
   Pour i ← 1 à 15 faire
      Ecrire(' * ')
   FinPour
FinProc
```

II.4.2. 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)
```

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

Exemple :

En utilisant la procédure *Etoiles* déclarée dans l'exemple précédent, écrire un algorithme permettant de dessiner un carré d'étoiles de 15 lignes et de 15 colonnes.

Solution :

```
Algorithme carré_étoiles
  Variables
  J : entier
{Déclaration de la procédure Etoiles()}
Procédure Etoiles()
  Variables
  i : entier
Début
  Pour i ← 1 à 15 faire
    Ecrire(' * ')
  FinPour
FinProc
{Algorithme principal (Partie principale)}
Début
  Pour j ← 1 à 15 faire
    {Appel de la procédure Etoiles}
    Etoiles()
  FinPour
Fin
```

Remarque 1 :

Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de la partie principale (algorithme principal).

Remarque 2 :

Lors de la conception d'un algorithme deux aspects apparaissent :

- La définition (déclaration) de la procédure ou fonction ;
- L'appel de la procédure ou fonction au sein de l'algorithme principal.

II.4.3. Passage de paramètres

Les échanges d'informations entre une procédure et le sous algorithme appelant se font par l'intermédiaire de paramètres.

Il existe deux principaux types de passages de paramètres qui permettent des usages différents :

II.4.3.1. 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	Pascal

Algorithme Passage_par_valeur
Variables
N : entier
{Déclaration de la procédure P1}
Procédure P1(A : entier)
Début
A ← A*2
Ecrire (A)
FinProc
{Algorithme principal}
Début
N ← 5
P1(N)
	Program Passage_par_valeur;
	Var
	N : integer;
	{Déclaration de la procédure P1}
	Procedure P1(A : integer);
	begin
	A:=A*2;
	writeln(A);
	end;
	{Algorithme principal}
	begin
	N:=5;
	P1(N);

Ecrire(N) **Fin.**	writeln(N); **end.**

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 recopié dans le paramètre formel A. La procédure effectue alors le traitement et affiche la valeur de la variable A, dans ce cas 10.

Après l'appel de la procédure, l'algorithme affiche la valeur de la variable N dans ce cas 5.
La procédure ne modifie pas le paramètre qui est passé par valeur.

II.4.3.2. Passage par adresse ou par variable

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 procédés du mot clé **VAR**.

Exemple :

Reprenons l'exemple précédent :

Algorithme	Pascal

Algorithme Passage_par_reference
Variables
N : entier
{Déclaration de la procédure P1}
Procédure P1(**Var** A : entier)
Début
A ← A*2
Ecrire(A)
FinProc
{Algorithme principal}
Début
N ← 5
P1(N)
	Program Passage_par_reference;
	Var
	N : integer;
	{Déclaration de la procédure P1}
	Procedure P1(**Var** A : integer);
	Begin
	A := A*2;
	writeln(A);
	End;
	{Algorithme principal}
	Begin
	N := 5;
	P1(N);

Ecrire(N) **Fin**	writeln(N); **End.**

A l'exécution de la procédure, l'instruction ***Ecrire(A)*** permet d'afficher à l'écran 10. Au retour dans l'algorithme principal, l'instruction ***Ecrire(A)*** 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 ← 1*2

Remarque :

Lorsqu'il y a plusieurs paramètres dans la définition d'une procédure, il faut absolument qu'il y en ait le même nombre à l'appel et que l'ordre soit respecté.

Algorithme Vs sous algorithme

Problème :

- Ecrire un algorithme qui permet de calculer la somme de deux nombres entiers.
- Ecrire une fonction qui permet de calculer la somme de deux nombres entiers

Algorithme	Fonction
Algorithme Somme ; **Variables** a,b,S : entier ; **Début** Ecrire('Donner 2 nombres entiers : ') ; Lire(a,b) S ← a+b ; Ecrire('S =', S); **Fin.**	Function Somme(a:entier, b:entier):entier; **Variable** S : entier ; **Début** S ← a+b ; Somme ← S ; **Fin.**
Explication : L'algorithme doit être complet, c'est-à-dire il comporte généralement les trois étapes : Lecture des données, calcul des résultats, et	**Explication :** Plus les sous algorithmes sont élémentaires, mieux ils seront exploités. Donc il faut faire un seul traitement parmi les trois : lecture des

affichage des résultats. Donc on doit écrire tout même si ce n'est pas demandé.	données, calcul des résultats, et affichage des résultats. Donc on doit se contenter de répondre à la question seulement ni plus ni moins.

II.5. Exercices avec corrigés

Exercice N°01 :

Essayez d'implémenter votre propre fonction **ABS()**. Nommez la Absolue. La fonction Absolue doit retourner la valeur absolue d'un paramètre entier.

Corrigé de l'exercice N°01 :

Algorithme	Pascal

Algorithme Val_ABS ;
Variables
x : entier ;
fonction Absolue (a : entier) : entier ;
Début
Si (a>=0) **alors**
Absolue ← a
Sinon
Absolue ← -a ;
Fin ;
Début
Ecrire('Donnez une valeur : ');
Lire(x) ;
Ecrire('La valeur absolue de ', x, ' est : ',
Absolue(x)) ;
Fin.
	Program Val_ABS ;
	Var
	x : integer;
	function Absolue (a : integer) : integer ;
	Begin
	If (a>=0) **then**
	Absolue:=a
	else
	Absolue:=-a;
	End;
	Begin
	Writeln('Donnez une valeur : ');
	Readln(x) ;
	Writeln('La valeur absolue de ', x, ' est : ',
	Absolue(x)) ;
	End.

Exercice N°02 :

Ecrire un programme Pascal permettant de calculer la surface d'un rectangle en utilisant une fonction ayant comme paramètres la longueur et la largeur de ce rectangle.

Corrigé de l'exercice N°02 :

Algorithme	Pascal

Algorithme surface_rec;
Var
Long, larg : réel ;
Fonction surface (a, b : réel) : réel ;
Début
Surface ← a*b ;
Fin ;
Début
Ecrire('Donnez la longueur et la largeur d''un rectangle : ') ;
Lire(long, larg) ;
Ecrire('La surface =', surface(long, larg)) ;
Fin.
	Program surface_rec;
	Var
	Long, larg : real ;
	Function surface (a, b : real) : real ;
	Begin
	Surface :=a*b ;
	End ;
	Begin
	Writeln('Donnez la longueur et la largeur d''un rectangle : ') ;
	Readln(long, larg) ;
	Writeln('La surface =', surface(long, larg)) ;
	End.

Exercice N°03 :

Soit le programme pascal suivant :

```
Program Exo_1;
Var x, y : integer;                    {variables globales}
Procedure Proc1(A: integer; Var B: integer) ;      {SP1}
Begin
    A := A + 1;  B := 22;
End;                                          {Fin SP1}
Procedure Proc2(A: integer; B : integer) ;      {SP2}
Begin
    A := A + 1;  B := 22;
End;                                          {Fin SP2}
Begin                                          {PP}
  x := 3;  y := 7;
  Proc1(x, y);                          {Appel au SP1}
```

```
  Writeln('x= ', x, ' y = ', y);
  x := 3;  y := 7;
  Proc2 (x, y);                    {Appel au SP2}
  Writeln('x= ', x, ' y = ', y);
End.                               {Fin PP}
```

Questions :

1) Exécuter le programme.
2) Quelle est la différence entre les deux procédures Proc1 et Proc2 ?
3) Quels sont les paramètres à passage par valeur et ceux à passage par variable ?
4) Quels sont les paramètres formels des deux procédures ? les paramètres effectifs ?
5) Les appels Proc1(3,7) et Proc2(3,7) sont-ils corrects, expliquer pourquoi.
6) Dérouler le programme.
7) Exécuter le programme en donnant le type « réel » à la variable y. Que se passe-t-il ? Pourquoi ?

Remarque :

On définit les acronymes SP et PP comme suit :

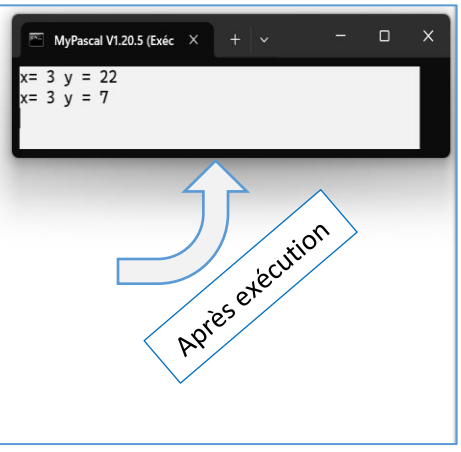
SP : **S**ous-**P**rogramme,

PP : **P**rogramme **P**rincipal

Corrigé de l'exercice N°03 :

1) Exécuter le programme :

```
1  Program Exo_1;
2  Var x, y : integer;                {variables globales}
3  Procedure Proc1(A: integer; Var B: integer) ;  {SP1}
4  Begin
5      A := A + 1; B := 22;
6  End;                               {Fin SP1}
7  Procedure Proc2(A: integer; B : integer) ;     {SP2}
8  Begin
9      A := A + 1; B := 22;
10 End;                               {Fin SP2}
11 Begin                              {PP}
12   x := 3; y := 7;
13   Proc1(x, y);                     {Appel au SP1}
14   Writeln('x= ', x, ' y = ', y);
15   x := 3; y := 7;
16   Proc2 (x, y);                    {Appel au SP2}
17   Writeln('x= ', x, ' y = ', y);
18 End.                               {Fin PP}
```



2) Quelle est la différence entre les deux procédures Proc1 et Proc2?

La différence est la présence du mot clé « **var** » dans Proc1 mais pas dans Proc2.

3) Quels sont les paramètres à passage par valeur et ceux à passage par variable ?

Les paramètres à passage par valeur et ceux à passage par variable :

Paramètre/Procédure	Proc1	Proc2
Passage de paramètre par valeur	**A**	**A et B**
Passage de paramètre par variable	**B**	**/**

4) Quels sont les paramètres formels des deux procédures ?

Les paramètres formels des deux procédures :

Type de paramètres/Procédure	Proc1	Proc2
Paramètres formels	**A et B**	**A et B**

Quels sont les paramètres effectifs ?

Les paramètres effectifs des deux procédures :

Type de paramètres/Procédure	Proc1	Proc2
Paramètres effectifs	**x et y**	**x et y**

Rappel :

- Un paramètre à passage par variable (ou par adresse) est un paramètre précédé par le mot clé «**var** ».
- Les paramètres formels sont les paramètres utilisés dans la déclaration des procédures et fonctions. Par contre, les paramètres effectifs sont les paramètres utilisés lors de l'appel aux procédures et fonctions. (Les paramètres formels sont séparés par des points-virgules).

5) Les appels Proc1(3,7) et Proc2(3,7) sont-ils corrects, expliquer pourquoi ?

- Pour Proc1(3,7) : L'appel est incorrect car le paramètre formel B est en entrée / Sortie : il doit lui correspondre une variable, et non pas une constante.
- Pour Proc2(3,7) : L'appel est correct car les paramètres formels A et B sont des paramètres d'entrée seulement. Par conséquent, ils peuvent leur correspondre des variables ou des constantes.

6) Dérouler le programme.

Instructions	Variables programme principal		Variables procédure Proc1		Variables procédure Proc2		Affichage
	x	y	A	B	A	B	
x:=3	3	/					
y :=7	3	7					
Proc1(x, y) *Appel à la procédure Proc1 et transmission des paramètres*	3	7	3	7			
A := A + 1 A := 3 + 1 A := 4			4	7			
B := 22 *Proc1 a calculé la valeur de A et B, la valeur de B est retournée à la variable globale y* *(car B est passé par variable)*		22		22			
Writeln('x = ', x, ' y = ', y);	3	22	4	22			x = 3 y = 22
x:=3	3	22					
y :=7	3	7					
Proc2(x, y) *Appel à la procédure Proc2 et transmission des paramètres*	3	7			3	7	
A := A + 1 A := 3 + 1 A := 4					4	7	
B := 22 *Proc2 a calculé la valeur de A et B, mais contrairement à Proc1, la valeur de B n'est pas retournée à la variable globale y* *(car B est passé par valeur)*						22	
Writeln('x = ', x, ' y = ', y);	3	7			4	22	x = 3 y = 7

7) Exécuter le programme en donnant le type « réel » à la variable y. Que se passe-t-il ? Pourquoi ?

Nous obtenons **une erreur de compilation** indiquant que « un appel avec paramètre par variable doivent être de type exact de la déclaration ». Ceci revient à l'incompatibilité de types entre le paramètre effectif « y » et son paramètre formel correspondant « B ».

Remarque :

Afin d'effectuer la transmission des paramètres effectifs vers les paramètres formels, la correspondance de type et de nombre entre ces paramètres est **obligatoire.** i.e,

- Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels.
- Le paramètre effectif et le paramètre formel correspondant doivent avoir le même type.

Exercice N°04 :

Soit le programme pascal suivant :

```
Program  exo_2;
Var   N: integer; Fact: integer;      {Variables globales}
  Function  Factoriel (m: integer): integer;        {SP}
  Var j, f: integer;              {Variables locales au SP}
  Begin
     f := 1;
     For j := 1 to m do
       f := f*j;
     Factoriel := f;                    {Résultat retourné}
  End;                                          {Fin SP}
Begin                                         {Début PP}
  Write('Introduire N : ');
  Read(N);
  Fact := Factoriel(N);                    {Appel au SP}
  Write('Le factoriel de ', N,  ' = ', Fact);
End.                                          {Fin PP}
```

Questions :

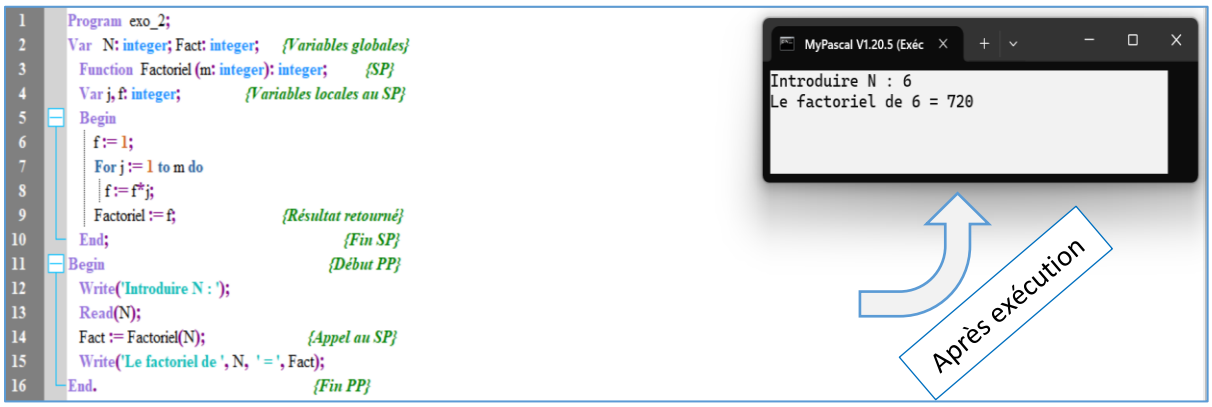
1) Exécuter le programme pour N = 6.
2) Réécrire le programme pour calculer la somme S suivante :

$$S = \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{N!} \qquad N \geq 1$$

3) Exécuter le programme pour N = 3.
4) Dérouler le programme pour N = 3.
5) Réécrire le programme modifié en remplaçant la fonction par une procédure de même nom.

Corrigé de l'exercice N°04 :

6) Exécuter le programme pour N = 6.



7) Réécrire le programme pour calculer la somme S suivante :

$$S = \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{N!} \quad N \geq 1 \quad \rightarrow \quad \text{Le terme général est : } S = \sum_{i=1}^{N} \frac{1}{i!}$$

Le programme Pascal – Solution 1	Le programme Pascal – Solution 2

Program exo_2_Q2;
Var i,N: integer; Fact: integer; S: real;
Function Factoriel (m: integer): integer;
Var j, f: integer;
Begin
f := 1;
For j := 1 to m do
f := f*j;
Factoriel := f;
End;
Begin
Write('Introduire N : ');
Read(N);
S := 0;
For i := 1 to N do
Begin
Fact := ***Factoriel(i)***;
S := S + 1/Fact;
End;
Write(' La somme S = ', S:3:1);
End.
	Program exo_2_Q2;
	Var i,N: integer; S: real;
	Function Factoriel (m: integer): integer;
	Var j, f: integer;
	Begin
	f := 1;
	For j := 1 to m do
	f := f*j;
	Factoriel := f;
	End;
	Begin
	Write('Introduire N : ');
	Read(N);
	S := 0;
	For i := 1 to N do
	Begin
	S := S + 1/Factoriel(i);
	End;
	Write(' La somme S = ', S:3:1);
	End.

8) Exécuter le programme pour N = 3

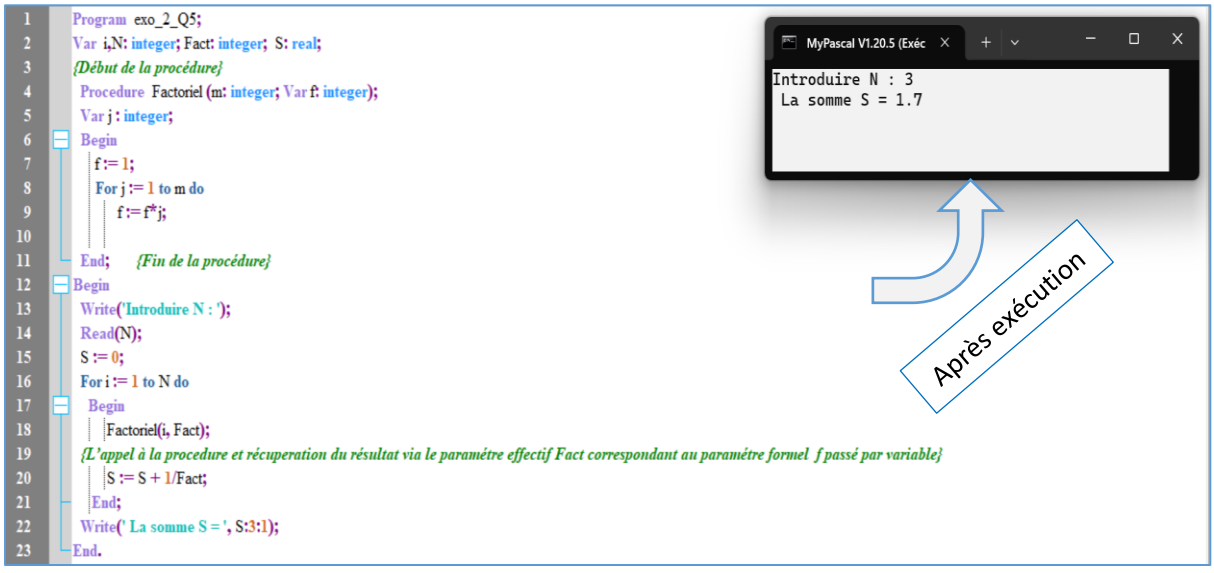
9) Dérouler le programme pour N = 3.

Instructions	Programme principal				La fonction Factoriel				Affichage
	N	i	Fact	S	m	j	f	Factoriel	
Write('Introduire N : ');	/	/	/	/					Introduire N :
Read(N)	3	/	/	/					
S := 0	3	/	/	0					
For i :=1	3	1	/	0					
Fact:=Factoriel(i) Fact := Factoriel(1) *(l'appel à Factoriel avec le paramètre i=1)*	3	1	**Fact := ?**	0					
=> La transmission des paramètres f:=1 for j:=1 f:=f*j → f :=1*1 = 1 Factoriel := f → factoriel := 1 *=> Le retour du résultat dans le PP*					**1**	1	1 1	**1**	
Fact := Factoriel(1) Fact := 1	3	1	1	**0**					
S := S + 1/Fact S := 0 + 1/1 S := 1	3	1	1	**1**					
For i := 2	3	2	1	1					
Fact:=Factoriel(i) Fact:=Factoriel(2) *(l'appel à Factoriel avec le paramètre i=2)*	3	2	**Fact := ?**	1					
=> La transmission des paramètres f:=1 for j:=1 f:=f*j → f := 1*1 = 1 for j:=2 f:=f*j → f := 1*2 = 2 Factoriel := f → factoriel := 2 *=> Le retour du résultat dans le PP*					**2**	1 2	1 1 2	**2**	
Fact := Factoriel(2) Fact := 2	3	2	2	**1**					
S := S + 1/Fact → S := 1 + 1/2 S := 1.5	3	2	2	**1.5**					
For i :=3	3	3	2	**1.5**					
Fact:=Factoriel(i) Fact := Factoriel(3) *(l'appel à Factoriel avec le paramètre i=3)*	3	3	**Fact := ?**	**1.5**					
=> La transmission des paramètres f:=1 for j:=1 f:=f*j → f := 1*1 = 1 for j:=2 f:=f*j → f := 1*2 = 2 for j:=3 f:=f*j → f := 2*3 = 6 Factoriel := f → factoriel := 6 *=> Le retour du résultat dans le PP*					**3**	1 2 3	1 1 2 6	**6**	
Fact := Factoriel(3) Fact := 6	3	3	6	**1.5**					
S := S + 1/Fact → S := 1.5 + 1/6 S := 1.666666666666667	3	3	6	**1.6666666666666667**					
Write('La somme S = ',S:3:1);	3	3	6	≠	3	3	6	6	La somme S = 1.7

10) Réécrire le programme modifié en remplaçant la fonction par une procédure de même nom.

Le programme avec fonction	Le programme avec procédure

Program exo_2_Q5;
Var i,N: integer; Fact: integer; S: real;
{Début de la fonction}
Function Factoriel (m: integer)~~: integer~~;
Var j,~~f~~: integer;
Begin
f := 1;
For j := 1 to m do
f := f*j;
~~Factoriel := f;~~
End; **{Fin de la fonction}**
 	
Begin
Write('Introduire N : ');
Read(N);
S := 0;
For i := 1 to N do
Begin
~~Fact :=~~ Factoriel(i);
{L'appel à la fonction et affectation du résultat retourné à la variable globale Fact}
S := S + 1/Fact;
End;
Write(' La somme S = ', S:3:1);
End.
	Program exo_2_Q5;
	Var i,N: integer; Fact: integer; S: real;
	{Début de la procédure}
	Procedure Factoriel (m: integer; **Var f: integer**);
	Var j : integer;
	Begin
	f := 1;
	For j := 1 to m do
	f := f*j;
	
	End; **{Fin de la procédure}**
	Begin
	Write('Introduire N : ');
	Read(N);
	S := 0;
	For i := 1 to N do
	Begin
	Factoriel(i, **Fact**);
	{L'appel à la procedure et récuperation du résultat via le paramétre effectif Fact correspondant au paramétre formel f passé par variable}
	S := S + 1/Fact;
	End;
	Write(' La somme S = ', S:3:1);
	End.



Exercice N°05 :

Un nombre parfait est un nombre entier positif supérieur à 1 qui est égal à la somme de ses diviseurs excepté lui-même (on ne compte pas comme diviseur le nombre lui-même).

1) Écrire une fonction **« Parfait »** qui vérifie si un nombre entier A est parfait ou non.

2) En utilisant la fonction, écrire un programme Pascal pour chercher les nombres parfaits entre 1 et 100.

Exemple : 6 et 28 sont des nombres parfaits car : 6 = 1 + 2 + 3, 28 = 1 + 2 + 4 + 7 + 14

Corrigé de l'exercice N°05 :

Le programme Pascal – Solution 1

```
Program  exo_3;
Var  i: integer;

  Function  Parfait(A: integer): boolean ;
  Var j, S: integer;
  Begin
     S:=0;
     For j:=1 to (A div 2) do
         If (A mod j = 0) then
            S := S + j;
     If (S = A) then
       Parfait := True
     Else
        Parfait := False;
  End;

Begin
  For i := 1 to 100 do
      If Parfait(i) then
         Writeln(i, ' est un nombre parfait ');
End.
```

Le programme Pascal – Solution 2

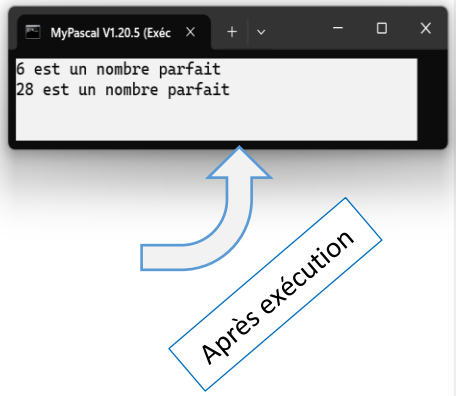
```
Program  exo_3;
Var  i: integer;

  Function  Parfait(A: integer): boolean ;
  Var j, S: integer; P: boolean;
  Begin
     S:=0;
     For j:=1 to (A div 2) do
         If (A mod j = 0) then
            S := S + j;
     If (S = A) then
       P:= True
     Else
        P:= False;

    Parfait := P;
  End;

Begin
  For i := 1 to 100 do
      If Parfait(i) then
         Writeln(i, ' est un nombre parfait ');
End.
```

```
1  Program exo_3;
2  Var i: integer;
3
4   Function Parfait(A: integer): boolean ;
5   Var j, S: integer;
6   Begin
7     S:=0;
8     For j:=1 to (A div 2) do
9       If (A mod j = 0) then
10        S := S + j;
11    If (S = A) then
12      Parfait := True
13    Else
14      Parfait := False;
15  End;
16
17 Begin
18  For i := 1 to 100 do
19    If Parfait(i) then
20      Writeln(i, ' est un nombre parfait ');
21 End.
```



II.6. Exercices supplémentaires

Exercice supplémentaire 01 :

Écrire un programme Pascal qui permet de vérifier, à l'aide de sous-programmes « **oppose** » et « **inverse** », si deux nombres sont opposés ou non et s'ils sont inverses ou non, respectivement.

Deux nombres sont opposés si leur somme est égale à 0.

Deux nombres sont inverses si leur produit est égal à 1.

Exercice supplémentaire 02 :

1) Écrire une fonction « **Puiss** » qui calcule la puissance énième de X (X^N) :

$$\text{Puiss}(X,N) = \begin{cases} 0 & \text{Si } X = 0 \\ 1 & \text{Si } N = 0 \\ X \times X \times \ldots \times X \text{ (N fois)} & \text{Si } N > 1 \end{cases}$$

2) En utilisant la fonction « Puiss », écrire un programme Pascal qui permet de calculer la somme S :

$$S = X^1 + X^3 + X^5 + \ldots + X^{2N+1}$$

Exercice supplémentaire 03 :

1) *Écrire une procédure « **occurences** » qui calcule le nombre d'occurrences d'un chiffre (0 ≤ chiffre ≤ 9) dans un tableau T de N éléments entiers.*

Exemples:

Le nombre d'occurrences du chiffre 7 dans T = | 7 | 7 | 8 | est 2.

Le nombre d'occurrences du chiffre 1 dans T = | 7 | 7 | 8 | est 0.

2) Écrire la procédure, l'insérer dans le programme et afficher les résultats dans le programme principal.

Exercice supplémentaire 04 :

1) Écrire une procédure « **Max_vecteur**» qui calcule le plus grand élément du tableau T de K composantes réelles.
2) En utilisant la procédure « **Max_vecteur**», écrire un programme Pascal qui permet de calculer et afficher le plus grand élément de chaque colonne d'une matrice M carrée d'ordre N.

Exercice supplémentaire 05 :

1) Écrire une procédure « **Décaler** » qui permet de décaler de manière circulaire à droite les éléments d'un vecteur T de M composantes entières.

2) En utilisant la procédure « **Décaler** », écrire un programme Pascal qui permet de lire les éléments d'une matrice A de taille N×M, décaler de manière circulaire à droite les éléments de chaque ligne et afficher la matrice résultat.

Exercice supplémentaire 06 :

Soit le programme suivant :

```
program Examen;
Var
    dist, tim, Vit: REAL;
…….. calcul(A: real; B: real): real;
……
V:real;
begin
  if B <> 0 then
    V:= A / B
 else
    V:= 0;
  ……..;
end;
begin
  write('Entrez dist : ');
  readln(dist);
 write('Entrez tim : ');
 read(tim);
 Vit:= calcul(dist, tim);
  if tim <> 0 then
    writeln('Vit est : ', Vit:0:2)
else
   writeln('Le tim ne peut pas être
zéro.');
```

Question :

1. Compléter le programme Pascal
2. Quel est le type de sous-programme utilisé
3. Dérouler le programme pour dis=200km et Tim=3h
4. Déduire ce que fait le programme
5. Réécrire le programme, en transformant le sous-programme utilisé dans cet exercice à l'autre type de S/programme

Exercice supplémentaire 07 :

Soit le programme PASCAL suivant :

```
Program Exo2;

var
  v, t, d : real;

….. calcule(vitesse, temps: real): real;
begin
  calcule := vitesse * temps;
end;
begin
  read(v, t);
  d := calcule(v, t);
  writeln(' d = ', d:10:2, ' m');
end.
```

Questions :

1. Quel type de sous-programme est utilisé dans ce programme.
2. Dérouler le programme pour **v= 12.5** et **t = 4**.
3. Que fait ce programme ?
4. Réécrire le sous-programme en complétant avec ce qui manque.
5. Réécrire le programme en transformant le sous-programme utilisé à l'autre type de sous-programme.

Exercice supplémentaire 08 :

Soit le programme Pascal suivant :

```
Program EXO2;
 …… Surface (b, h : Real) : …….;
 Var S :real;
  Begin
     S := 1 / 2 * b * h;
     …….. := S ;
  end;
Var
 ……: Real;
Begin
   Write ('Entrez la base et la hauteur : ');
   Read (base, haut);
   …….:= Surface (……..);
  Writeln ('La surface est : ', surf : 0 : 2);
 end.
```

Questions :

1. Réécrire le programme en complétant les parties manquantes.
2. Quel type de sous-programme est utilisé ?
3. Quels sont les paramétrés formels et effectifs utilisés ?
4. Dérouler le programme pour : base = 5 et haut = 8
5. Déduire ce que fait le programme ?
6. Réécrire le programme en transformant le sous-programme utilisé en l'autre type de sous-programme.

Exercice supplémentaire 09 :

Soit le programme Pascal suivant :

Program somme;
Var S, A,B : Integer;
…...somme (x,y : Integer, **var** T: integer);
Begin
…:= X+Y ;
End;
Begin {Début du programme principal}
Read(A,B);omme(**…,…,…**);
Write(**…**);
End. {Fin du programme principal}
	Questions :
	
	1) Compléter le programme Pascal.
	2) Quels sont les paramètres formels et effectifs ?
	3) Dérouler le programme pour A= 10, B=15.
	4) Déduire ce que fait le programme.
	5) Réécrire le programme en transformant la procedure en unefonction.

CHAPITRE III

Les enregistrements et fichiers

Chapitre III : Les enregistrements et fichiers

III.1. Objectif de ce chapitre

A l'issu de ce chapitre, l'apprenant sera capable de :

- Comprendre et utiliser les enregistrements pour structurer des données complexes en Pascal.
- Créer, ouvrir et manipuler des fichiers en mode lecture, écriture et mise à jour.
- Mettre en œuvre des procédures de sauvegarde et de récupération de données à l'aide de fichiers d'enregistrements.
- Déboguer et optimiser les opérations de lecture et d'écriture sur fichiers pour garantir l'intégrité des données.

III.2. Déclaration d'enregistrement

Un enregistrement est un type complexe de données qui permet de regrouper plusieurs données dans une seule structure.

C'est mieux de déclarer les enregistrements comme des types, et ceci en suivant la syntaxe ci-dessous :

Algorithme :

```
Type <id_Enreg> = Enregistrement
    <id_ch1> : <type1> ;
    <id_ch2> : <type2> ;
    ...
    <id_chN> : <typeN> ;
Fin
```

Pascal :

```
Type <id_Enreg> = RECORD
    <id_ch1> : <type1> ;
    <id_ch1> : <type2> ;
    ...
    <id_chN> : <typeN> ;
End;
```

Tel-que :

<id_ch1>, <id_ch2> …, <id_chN> sont des champs et <type1>, …, <typeN> sont leur types respectifs.

Exemple :

On peut déclarer un type Produit comme suit :

En algorithme	En Pascal
Type *Produit* = ***Enregistrement***	***Type*** *Produit* = ***RECORD***
Designation : Chaîne ;	Designation : String[30];
Reference : Chaîne ;	Reference : String[13];
Quantite : Entier ;	Quantite : Integer;
Prix_U : Réel ;	Prix_U : Real;
Fin	**End;**

III.3. Affectation, lecture et affichage des enregistrements

Pour référencer ou accéder à un champ quelconque d'un enregistrement, on utilise la notation pointée. La notation pointée utilise la variable enregistrement (identificateur) suivie d'un point puis du nom du champ auquel on veut accéder. Soit :

```
<Variable_Enregistrement>.<Nom du champ>
```

Exemple :

Un nombre complexe peut être décrit par un enregistrement . Soit :

En algorithme	En Pascal
Type *Complexe* = ***Enregistrement***	***Type*** *Complexe* = ***RECORD***
X, Y : réel	X, Y : real;
Fin;	**End;**

Un nombre complexe peut alors être décrit (représenté) par un enregistrement de deux champs : X est la partie réelle et Y est la partie imaginaire. Par exemple :

En algorithme	Programme Pascal
Algorithme Exemple01;	**Program** Exemple01;
Type	**Type**

Complexe = **Enregistrement**
x, y : réel;
Fin;
Variables
nb1, nb2, nb : Complexe;
Début
écrire ('Donne le premier nombre nb1 : ') ;
Lire(nb1.x, nb1.y) ;
écrire ('Donne le deuxième nombre nb2 : ');
Lire(nb2.x, nb2.y) ;
nb.x ← nb1.x + nb2.x;
nb.y ← nb1.y + nb2.y;
écrire ('Nb = ', nb.x, ' + ', nb.y, ' * i ') ;
Fin.
	Complexe = **Record**
	x , y : real;
	End;
	Var
	nb1, nb2, nb : Complexe;
	Begin
	Write ('Donne le premier nombre nb1 : ') ;
	Read(nb1.x, nb1.y) ;
	Write ('Donne le deuxième nombre nb2 : ');
	Read(nb2.x, nb2.y) ;
	nb.x:= nb1.x + nb2.x;
	nb.y := nb1.y + nb2.y;
	write ('Nb = ', nb.x, ' + ', nb.y, ' * i ') ;
	End.

Ce qui correspond au nombre complexe *Nb = x - yi*

III.4. L'instruction WITH (Avec)

L'instruction PASCAL **WITH**, permet d'utiliser les champs de l'enregistrement sans répéter la variable d'enregistrement.

En algorithme	En Pascal

Avec <*var_enreg*> ***Faire***
<instructions_champs_enreg>;
Fin-Avec;
	With <*var_enreg*> ***Do***
	Begin
	<instructions_champs_enreg>;
	End;

Dans le cas des deux affectations précédentes, on peut écrire :

En algorithme	Programme Pascal
Algorithme Exemple01; **Type** Complexe = **Enregistrement** x, y : réel; **Fin;**	**Program** Exemple01; **Type** Complexe = **Record** x , y : real; **End;**

Variables
nb1, nb2, nb : Complexe;
Début
écrire ('Donne le premier nombre nb1 : ') ;
Lire(nb1.x, nb1.y) ;
écrire ('Donne le deuxième nombre nb2 : ');
Lire(nb2.x, nb2.y) ;
Avec nb **faire**
x ← nb1.x + nb2.x;
y ← nb1.y + nb2.y;
écrire ('Nb = ', x, ' + ', y, ' * i ') ;
Fin-Avec;
Fin.
	Var
	nb1, nb2, nb : Complexe;
	Begin
	Write ('Donne le premier nombre nb1 : ') ;
	Read(nb1.x, nb1.y) ;
	Write ('Donne le deuxième nombre nb2 : ');
	Read(nb2.x, nb2.y) ;
	with nb **do**
	begin
	x:= nb1.x + nb2.x;
	y := nb1.y + nb2.y;
	write ('Nb = ', nb.x, ' + ', nb.y, ' * i ') ;
	end;
	End.

Avec l'instruction **WITH**, on évite de répéter la variable d'enregistrement pour chaque champs.

Un deuxième exemple, avec le type enregistrement Produit :

En algorithme	Programme Pascal

Algorithme Exemple03;
Type
Produit = **Enregistrement**
Designation : Chaîne[40];
Reference : Chaîne[20];
Quantite : entier;
Prix_U : réel;
Fin;
Variables
p : Produit;
Début
Avec p **faire**
Designation ← 'Ordinateur';
Reference ← 'HP-Z00365';
	Program Exemple03;
	Type
	Produit = **Record**
	Designation : String[40];
	Reference : String[20];
	Quantite : integer;
	Prix_U : real;
	End;
	Var
	p: Produit ;
	Begin
	With p **do**
	begin
	Designation := 'Ordinateur';
	Reference := 'HP-Z00365';

Quantite ← 6;	Quantite := 6;
Prix_U ← 57000.00;	Prix_U := 57000.00;
écrire ('Produit : ') ;	writeln ('Produit : ') ;
écrire(' Désignation : ', Designation);	writeln(' Désignation : ', Designation);
écrire(' Référence : ', Reference);	writeln(' Référence : ', Reference);
écrire(' Quantité : ', Quantite);	writeln(' Quantité : ', Quantite);
écrire(' Prix Unitaire : ', Prix_U);	writeln(' Prix Unitaire : ', Prix_U:0:2);
Fin-Avec;	**end;**
Fin.	**End.**

III.5. Les fichiers

Un fichier est une collection de données sauvegardée dans un support de stockage secondaire : Disque dur, flash-disk, CD, … . En langage de programmation, un fichier est une structure de données permettant de manipuler les données sauvegardées en dehors de la mémoire centrale. En langage PASCAL, nous pouvons manipuler 3 types de fichiers.

- Les Fichiers à 'accès direct' (*structurés*) de type **FILE OF**
- Les Fichiers à accès séquentiel (*Textuel*) de type **TEXT**
- Les Fichiers 'sans type' (*Binaire*) de type **FILE**.

Dans ce qui suit, nous aborderons uniquement les fichiers à accès direct : FILE OF … Ce type de fichier est constitué de blocs homogènes (de même type) dits : cellules, pouvant être de type simple : entier, réel, caractères, … ou de type enregistrements.

III.5.1. Éléments d'un fichier

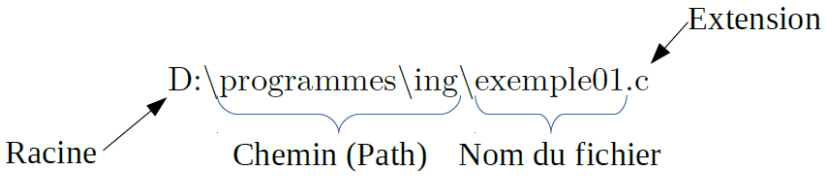
Parmi les éléments qui sont liés au fichier, nous allons voir :

- *Nom externe (nom physique)*
- *Nom interne (nom logique)*
- *Tampon (Buffer)*
- *FDF : Fin De Fichier (EOF : End Of File)*
- *Indice (pointeur interne)*

a) Nom externe (nom physique)

Une chaîne de caractère contenant l'identifiant du support (Racine), le chemin vers le fichier (path), le nom propre du fichier ainsi que son extension.

Exemples :

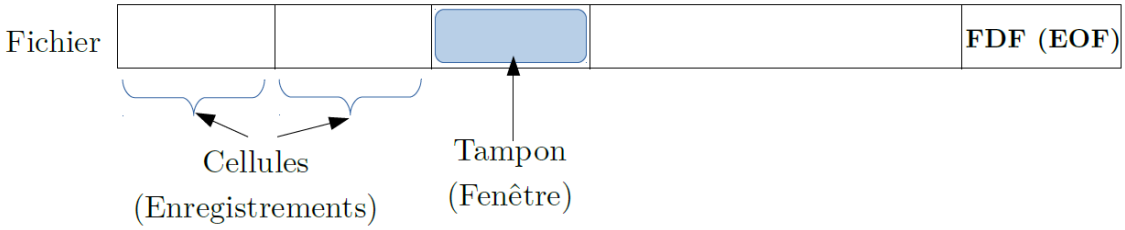


b) Nom interne (nom logique)

Nom interne est le nom avec lequel le fichier est identifié dans un programme (algorithme). C'est le nom logique qui doit être associé avec le nom externe (le nom physique). Plus précisément, un nom interne (logique) est un identificateur d'une variable pour accéder aux données physiques du fichier.

c) Tampon (Buffer)

On appelle tampon (buffer) d'un fichier, une zone de la mémoire central (RAM) pouvant contenir une cellule (éventuellement un enregistrement) de ce fichier. C'est à travers ce tampon qu'on rends les cases du fichier visible : pour cela, on appelle le tampon ***fenêtre*** à travers laquelle on *voit* le fichier.



d) FDF (EOF)

Chaque fichier se termine par une cellule spéciale dite FDF (pour Fin de Fichier), en anglais c'est : EOF (**End Of File**). Ça permet de savoir la fin de fichier lors du parcours des données du fichiers (voir les algorithmes et programmes dans la suite du cours).

e) Indice (pointeur)

Pour chaque fichier, un indice (pointeur interne) caché, qui est géré automatiquement par le type fichier, est utilisé pour lire ou écrire la cellule pointée par cet indice.

III.5.2. Manipulation des fichiers de type FILE OF

Dans ce qui suit, nous allons voir comment :

- *Déclarer un fichier de type **File Of***
- *Créer un nouveau fichier*
- *Lire et Afficher le contenu d'un fichier*

Pour illustrer les opérations ci-dessus, nous prenons un exemple de fichier d'étudiants, où chaque étudiant est caractérisé par : *Matricule, Nom, Prénom, date de naissance, filière, niveau et sa moyenne.*

a) Déclaration de fichier

L'exemple ci-dessous, explique comment déclarer un fichier de type **File Of** contenant des enregistrements d'étudiant :

En algorithme	Programme Pascal

Algorithme Exemple06_Fichier;
Type
Etudiant = **Enregistrement**
Matricule : Chaîne[15] ;
Nom, Prenom : Chaîne[30];
Date_N : Chaîne[10];
Filiere : Chaîne[50];
Niveau : entier ;
Moyenne : réel;
Fin;
Variables
F : Fichier de Etudiant ;
Début
Fin.
	Program Exemple06_Fichier;
	Type
	Etudiant = **Record**
	Matricule : String[15] ;
	Nom, Prenom : String[30];
	Date_N : String[10];
	Filiere : String[50];
	Niveau : integer ;
	Moyenne : real;
	End;
	Var
	F : File of Etudiant;
	Begin
	End.

Remarques :

- Un fichier peut être vu comme un tableau à une dimension (vecteur) illimité (pratiquement illimité) sauvegardé sur une mémoire secondaire
- Les cases (cellules) du fichier sont accessible par un pointeur invisible.

b) Création d'un nouveau fichier

Pour créer un nouveau fichier, on doit :

1. Déclarer la structure de fichier : on continue sur l'exemple 6 (fichier d'étudiants)
2. Assigner le nom logique du fichier avec le nom physique (chemin et nom complets du fichier)
3. Ouvrir le fichier en écriture : Réécrire / Rewrite

4. Ecrire les données dans un fichier
5. Fermer le fichier

L'exemple illustre les étapes ci-dessus pour créer un nouveau fichier :

En algorithme	Programme Pascal
Algorithme Exemple08_Fichier_Creer;	**Program** Exemple08_Fichier_Creer;
Type	**Type**
Etudiant = Enregistrement	Etudiant = Record
Matricule : Chaîne[15] ;	Matricule : String[15];
Nom, Prenom : Chaîne[30];	Nom, Prenom : String[30];
Date_N : Chaîne[10];	Date_N : String[10];
Filiere : Chaîne[50];	Filiere : String[50];
Niveau : entier ;	Niveau : integer ;
Moyenne : réel;	Moyenne : real;
Fin;	**End;**
Variables	**Var**
F : **Fichier de** Etudiant ;	F : **File of** Etudiant;
N, i : entier ;	N, i : integer ;
E : Etudiant ;	E : Etudiant ;
Début	**Begin**
Assigner (F, 'etudiants.dat ') ;	Assign(F, 'etudiants.dat ') ;
Réécrire(F) ; //Ouvrir le fichier en écriture	Rewrite(F); //Ouvrir le fichier en écriture
écrire ('Données le nombre d'étudiants : ');	write ('Données le nombre d'étudiants : ');
Lire(N);	Read(N);
écrire ('Données les informations : ') ;	write ('Données les informations : ') ;
Pour i←1 **à** N **faire**	**for** i:=1 **to** N **do**
Avec e faire	**begin**
Lire(Matricule, Nom, Prenom) ;	**wi**th e **do**
Lire(Date_N);	**begin**
Lire(Filiere) ;	Readln(Matricule, Nom, Prenom) ;
Lire(Niveau) ;	Readln(Date_N);
Lire(Moyenne);	Readln(Filiere) ;
Fin-Avec;	Readln(Niveau) ;
écrire(F, e); //Ajouter e au fichier F	Readln(Moyenne);

Fin-Pour;
Fermer(F) ;
Fin.
	end;
	write(F, e); //Ajouter e au fichier F
	end;
	Close(F) ;
	End.

Remarques :

- L'instruction Réécrire(F); / Rewrite(F); permet de créer un nouveau fichier si le fichier n'existe pas, sinon (le fichier existe), cette instruction écrasera le fichier existant : ***IL FAUT FAIRE ATTENTION POUR NE PAR PERDRE DES DONNÉES.***
- L'instruction Écrire(F, e), en mode écriture, permet d'ajouter l'enregistrement e à la fin du fichier F.
- À la fin du programme, il ne faut pas oublier de fermer le fichier F : Fermer(F) ; / Close(F);

c) Lire et afficher le contenu d'un fichier

Une fois un fichier est créé, on peut lire son contenu en l'ouvrant en mode lecture.

L'exemple ci-dessous, montre comment lire le contenu du fichier ***etudiants.dat*** affiche son contenu sur l'écran :

En algorithme	Programme Pascal
Algorithme Exemple09_Fichier_Afficher; **Type** Etudiant = **Enregistrement** Matricule : Chaîne[15] ; Nom, Prenom : Chaîne[30]; Date_N : Chaîne[10]; Filiere : Chaîne[50]; Niveau : entier ; Moyenne : réel; **Fin;** **Variables** F : **Fichier de** Etudiant ; N, i : entier ; E : Etudiant ;	**Program** Exemple09_Fichier_Afficher; **Type** Etudiant = **Record** Matricule : String[15] ; Nom, Prenom : String[30]; Date_N : String[10]; Filiere : String[50]; Niveau : integer ; Moyenne : real; **End;** **Var** F : **File of** Etudiant; i : integer ; E : Etudiant ;

Début	**Begin**
Assigner (F, 'etudiants.dat ') ;	Assign(F, 'etudiants.dat ') ;
Reouvrir(F) ; //Ouvrir le fichier en lecture/écriture	Reset(F);//Ouvrir le fichier en écriture
écrire ('Le contenu du fichier etudiants : ') ;	write ('Le contenu du fichier etudiants : ') ;
Tantque non(Eof(F)) **faire**	**While** not(Eof(f)) **do**
Lire(F, e); //Lire e à partir du fichier F	**begin**
Avec e **faire**	Read(F, e); //Lire e à partir du fichier F
Écrire(Matricule, Nom, Prenom, Date_N);	**With** e **do**
Écrire(Filiere, Niveau, Moyenne);	**begin**
Fin-Avec;	Writeln(Matricule, Nom, Prenom, Date_N);
Fin-Tantque;	Writeln(Filiere, Niveau, Moyenne);
Fermer(F) ;	**end ;**
Fin.	**end ;**
	Close(F) ;
	End.

Remarque 1 :

- L'instruction Reouvrir(F); / Reset(F) permet d'ouvrir un fichier existant pour lire son contenu ou le modifier. Si le fichier n'existe pas, il y aura une erreur pendant l'exécution: Runtime Error (Exception). Le pointeur de fichier est automatiquement au début du fichier (sur le premier enregistrement).
- L'instruction Lire(F, e) / Read(F, e) ;, en mode lecture, permet de lire l'enregistrement en cours et passe le pointeur à l'enregistrement suivant. À la fin du fichier, en aura le résultat de : *eof(F)* est *True*.

III.5.3. Quelques fonctions/procédures sur les fichiers

Dans cette section nous énumérons quelques fonctions et procédures prédéfinies pour qui sont utilisé pour manipuler les fichiers structurés (fichiers FILE OF). Quelques fonctions / procédures ont été utilisées dans les exemples précédents.

Le tableau suivant présente les principales fonctions / procédures sur les fichiers :

Fonction / Procédure	Signification
ASSIGN(Fichier, 'Nom_Fichier') ;	Procédure permettant d'associer le nom logique du fichier 'Fichier' au nom physique du fichier.
REWRITE (Fichier);	Procédure qui ouvre un fichier en mode écriture. Si le fichier n'existe pas, il sera créé, sinon, il sera écrasé (vider le fichier).
RESET (Fichier);	Procédure qui ouvre un fichier déjà existant et le prépare pour une lecture ou une écriture. Le pointeur de fichier est positionné sur le premier enregistrement du fichier. (ouverture en lecture/écriture).
READ(Fichier, v1, v2, …, vn) ;	Procédure de *lecture* des variables (tampons) v1, v2, …, vn à partir de fichier (lecture par fichier et non par clavier).
WRITE(Fichier, v1, v2, …, vn) ;	Procédure *d'écriture* des variables (tampons) v1, v2, …, vn dans le fichier (écriture sur le fichier et non sur l'écran).
FILESIZE(Fichier) ;	Fonction qui retourne le nombre d'enregistrements dans le fichier.
FILEPOS(Fichier) :	Fonction qui retourne la position actuelle du pointeur de fichier (entre 0 et Filesize(File)).
SEEK(Fichier, indice) ;	Procédure qui permet de positionner le pointeur de fichier sur l'indice ($0 \leq$ indice $\leq$ FileSize(File)-1). indice $\leq$ indice $\leq$ FileSize(File)-1). FileSize(File)-1).
EOF(Fichier) ;	Fonction booléenne qui retourne TRUE si on est à la fin du fichier, ou bien FALSE dans le cas contraire.
CLOSE(Fichier) ;	Fermer le fichier après son utilisation.
RENAME(Fichier, 'nom_fichier2');	Procédure pour renommer le fichier logique *Fichier* au nom : '*nom_fichier2*' . Le fichier doit être fermé avant RENAME.
ERASE(Fichier);	Procédure pour supprimer le fichier logique *Fichier*. Le fichier doit être fermé avant ERASE.

III.6. Exercices supplémentaires

Exercice supplémentaire 01 : Déclaration d'un enregistrement

Déclarez un enregistrement Etudiant contenant les champs suivants :

- Matricule : entier
- Nom : chaîne de 20 caractères
- Prenom : chaîne de 20 caractères
- Note : réel

Créez ensuite une variable de type Etudiant et écrivez un programme pour saisir ses informations via le clavier.

Exercice supplémentaire 02 : Fichier d'enregistrement

Écrivez un programme Pascal qui permet :

1. De créer un fichier de type **file of Etudiant**
2. De saisir les informations de plusieurs étudiants (via une boucle)
3. De les stocker dans un fichier nommé **etudiants.dat**

Exercice supplémentaire 03 : Affichage du contenu d'un fichier

Reprenez le fichier **etudiants.dat** créé précédemment et :

- Affichez à l'écran le contenu complet du fichier (tous les étudiants).

Exercice supplémentaire 04 : Recherche dans un fichier

Écrivez un programme qui permet de :

- Rechercher un étudiant dans le fichier **etudiants.dat** à partir de son matricule.
- Afficher ses informations si l'étudiant existe.

Exercice supplémentaire 05 : Moyenne et tri

Ajoutez une fonction qui calcule la moyenne des notes de tous les étudiants présents dans le fichier, puis :

- Affichez uniquement les étudiants ayant une note supérieure à cette moyenne.

Exercice supplémentaire 06 : Mise à jour d'un fichier

Écrivez un programme qui permet de modifier la note d'un étudiant dans le fichier à partir de son matricule.

Exercice supplémentaire 07 : Suppression logique

Écrivez un programme pour supprimer logiquement un étudiant (par exemple, en mettant la note à -1 ou en ajoutant un champ booléen **actif**).

Conclusion générale

Conclusion générale

En conclusion, ce support de cours constitue une ressource essentielle pour l'apprentissage et la maîtrise des concepts fondamentaux de la programmation. En explorant les variables indicées, les sous-programmes ainsi que la gestion des enregistrements et fichiers, il permet aux étudiants d'acquérir des compétences solides en structuration et en manipulation des données. Grâce à une approche équilibrée entre théorie et pratique, ce cours les prépare à concevoir des solutions informatiques efficaces, modulaires et adaptées aux exigences du développement logiciel. Il leur servira ainsi de base pour approfondir leurs connaissances et aborder des problématiques plus complexes dans la suite de leur formation.

Bibliographie

Bibliographie

[1] H. Halim, A. Khadraoui**,** *Algorithmique et programmation en Pascal*, Éditions Ellipses, 2019.

[2] J.-P. Braquelaire**,** *Algorithmique et programmation : De l'algorithme au programme*, Éditions Dunod, 2018.

[3] Yves Deville**,** *Algorithmique : Résolution de problèmes et programmation en Pascal*, Dunod, 2012.

[4] Jean-Michel André**,** *Algorithmique et programmation structurée en Pascal*, Éditions Ellipses, 2014.

[5] Jacques Arsac**,** *Initiation à la programmation et à l'algorithmique*, Dunod, 2009.

[6] Jean-Bernard Condat**,** *Algorithmique : Du pseudo-code au langage Pascal*, Éditions Eyrolles, 2008.

[7] Gilles Dowek**,** *Introduction à la programmation*, Éditions Eyrolles, 2015.

[8] Donald E. Knuth**,** *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, 1997.

[9] G. Brassard, P. Bratley**,** *Algorithmique – Concepts fondamentaux et applications*, Pearson, 2008.

[10] M. Charlier, J. Vanderdonckt**,** *Programmation structurée en Pascal*, De Boeck Université, 2003.

[11] Algorithmique : cours avec 957 exercices et 158 problèmes Livre de Charles E. Leiserson, Clifford Stein et Thomas H. Cormen 2017

[12] Algorithmes : Notions de base Livre de Thomas H. Cormen 2013.

[13] Ouzeggane R. Cours informatique 1, Université de Béjaia.