**Université de Bejaia**
**Faculté de Technologie**
**Département de Technologie**

**Année Universitaire 2021-2022**
**Semestre 2**

# Informatique 2 – Série de TP N°01

## Type Tableau à une Dimension : Vecteurs

# Sommaire

---

Adapté par: Redouane OUZEGGANE

rouzeggane@gmail.com - redouane.ouzeggane@univ-bejaia.dz

# TP INFORMATIQUE 2

## SÉRIE DE TP N°01 (TABLEAUX À UNE DIMENSION – VECTEURS)

### EXERCICE N°01 : ALGORITHMES → PROGRAMME

Soit l'algorithme suivant :

```
Algorithme Exo1;
  Variables
      T : Tableau[1..100] de entier;
      N, i : entier;
      S : réel;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(T[i]);
    Fin-Pour;

    {-*-*-*- Traitement -*-*-*- }
    S ← 0;
    Pour i←1 à N  Faire
       Si T[i] mod 3 = 0 Alors
         S ← S + T[i];
    Fin-Si

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

**Questions**

1- Traduire l'algorithme en Programme PASCAL, puis compiler et exécuter le programme pour :
N = 6 et T = [15 , 7 , 6 , 9, 11, 19]

2- Dérouler l'algorithme (le programme) pour les les valeurs de N et T ci-dessus ?

3- Déduire ce que fait l'algorithme ?

4- Ré-écrire l'algorithme (Programme) en remplaçant la boucle ***Pour*** par la boucle ***Tant-que***.

5- Ré-écrire l'algorithme (Programme) en remplaçant la boucle ***Pour*** par la boucle ***Répéter***.

### EXERCICE 02 : LECTURE ET AFFICHAGE D'UN VECTEUR

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

### EXERCICE 03 : SOMME ET MOYENNE DES ÉLÉMENTS D'UN VECTEUR

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

### EXERCICE 04 : INVERSER LES ÉLÉMENTS D'UN VECTEUR

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

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

### EXERCICE 05 : LE MIN ET LE MAX DANS UN VECTEUR

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

**2-** ÉCRIRE UN PROGRAMME PASCAL QUI PERMET DE RECHERCHER LE PLUS GRAND ÉLÉMENT DANS UN VECTEUR RÉEL V AINSI QUE SA POSITION.

# Solution – Série TP N°01

## EXERCICE N°01 : ALGORITHMES → PROGRAMME

### 1- Traduire l'algorithme en Programme PASCAL puis compiler et exécuter le programme

**Algorithme**

```
Algorithme TP1_Exo1_Q1;
  Variables
      T : Tableau[1..100] de entier;
      N, i : entier;
      S : entier;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(T[i]);
    Fin-Pour;

    {-*-*-*- Traitement -*-*-*- }
    S ← 0;
    Pour i←1 à N  Faire
        Si T[i] mod 3 = 0 Alors
          S ← S + T[i];
    Fin-Si

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

**Programme PASCAL**

```
Program TP1_Exo1;
  Var
      T : Array[1..100] of integer ;
      N, i : integer ;
      S : integer ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur T :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur T  :');
    For i:=1 to N do
      Read(T[i]);

    { -*-*-*- Traitement -*-*-*- }
    S := 0;
    For  i := 1 to N do
      If T[i] mod 3 = 0 then
        S := S + T[i];

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

Sur ce lien : https://onlinegdb.com/bPw5nzC3o, vous trouverez le programme PASCAL ci-dessus, que vous pouvez exécuter en ligne.

### 2- Dérouler l'algorithme / programme Pour N=6 et T = [15 , 7 , 6 , 9 , 11 , 19 ]

Le déroulement est l'exécution manuelle des instructions et l'affichage de l'impacte de ces instructions sur les valeurs des variables de l'algorithme (ou le programme). À travers le déroulement, nous visualisons l'évolution des valeurs des différentes variables, et ça nous permet d'expliquer ou de comprendre le fonctionnement de l'algorithme :

| ***Instructions*** | ***Variables*** | | | | ***Affichage*** |
|---|---|---|---|---|---|
| | ***N*** | ***i*** | ***T*** | ***S*** | |
| Lire (N); | 6 | / | | / | |
| Pour i←1 à N faire<br>Lire(T[i]);<br>Fin-Pour | '' | 1..N<br>1..6 | 1 2 3 4 5 6<br>[15 , 7 , 6 , 9, 11, 19] | | |
| S ← 0 ; | '' | / | '' | 0 | |
| Pour i=1<br>Si T[1] mod 3=0<br>Si 15 mod 3=0 ⇒ True<br>S ← S + T[i]=0 + T[1]<br>S ← 0 + 15 = 15 | '' | 1 | I=1 2 3 4 5 6<br>[15 , 7 , 6 , 9, 11, 19] | 15 | |

| Instructions | N | i | T | S | Affichage |
|---|---|---|---|---|---|
| Pour i=2<br>Si T[2] mod 3=0<br>Si 7 mod 3=0 ⇒ False | 6 | 2 | 1 i=2 3 4 5 6<br>[15 , 7 , 6 , 9, 11, 19] | | |
| Pour i=3<br>Si T[3] mod 3=0<br>Si 6 mod 3=0 ⇒ True<br>S ← S + T[i]=15 + T[3]=<br>S ← 15+6=21 | " | 3 | 1 2 i=3 4 5 6<br>[15 , 7 , 6 , 9, 11, 19] | 21 | |
| Pour i=4<br>Si T[4] mod 3=0<br>Si 9 mod 3=0 ⇒ True<br>S ← S + T[i]=21 + T[4]=<br>S ← 21+9=30 | " | 4 | 1 2 3 i=4 5 6<br>[15 , 7 , 6 , 9, 11, 19] | 30 | |
| Pour i=5<br>Si T[5] mod 3=0<br>Si 11 mod 3=0 ⇒ False | " | 5 | 1 2 3 4 i=5 6<br>[15 , 7 , 6 , 9, 11, 19] | | |
| Pour i=6<br>Si T[6] mod 3=0<br>Si 19 mod 3=0 ⇒ False | " | 6 | 1 2 3 4 5 i=6<br>[15 , 7 , 6 , 9, 11, 19] | | |
| Écrire('La somme S = ', S); | " | " | " | | La somme S = 30 |

### 3- Déduire ce que fait l'algorithme

D'après le déroulement ci-dessus, nous déduisons que l'algorithme permet de réaliser la somme des composantes du vecteur T divisibles par 3.

### 4- Ré-écrire l'algorithme en remplaçant la boucle Pour par la boucle Tant-que

La boucle Pour possède un compteur, une valeur initiale et une valeur finale. Par contre, la boucle Tant-que possède une condition. Comme nous l'avons vus au niveau du cours (ainsi que les exercices traités pendant les séances de T.P.), la boucle pour sera transformée à la boucle Tant-que suivant le modèle suivant :

**Algorithmique**

```
Pour cpt ← vi à vf Faire
    <Bloc_Inst_Pour> ;
Fin-Pour;
```

Transformé

```
cpt ← vi;
Tantque (cpt <= vf) do
    <Bloc_Inst_Pour> ;
    cpt ← cpt + 1 ;
Fin-Tantque;
```

Telque :

ctp : compteur de la boucle *Pour* (varible entière)
vi : valeur initiale
vf : valeur finale

**P. Pascal**

```
For cpt := vi à vf do
begin
    <Bloc_Inst_For> ;
end;
```

Transformé

```
cpt := vi;
While (cpt <= vi) do
begin
    <Bloc_Inst_For> ;
    cpt := cpt + 1 ;
end;
```

En appliquant ce modèle de transformation, nous aurons l'algorithme et le programme suivant :

**Algorithme**

```
Algorithme TP1_Exo1_Q4;
  Variables
      T : Tableau[1..100] de entier;
      N, i : entier;
      S : entier;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(T[i]);
    Fin-Pour;

    {-*-*-*- Traitement -*-*-*- }
    S ← 0;
    i ← 1;
    Tant-que i <= N   Faire
        Si T[i] mod 3 = 0 Alors
          S ← S + T[i];
        Fin-si
        i ← i+1;
    Fin-Si

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

**Programme PASCAL**

```
Program TP1_Exo1_Q4;
  Var
      T : Array[1..100] of integer ;
      N, i : integer ;
      S : integer ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur T :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur T  :');
    For i:=1 to N do
      Read(T[i]);

    { -*-*-*- Traitement -*-*-*- }
    S := 0;
    i := 1;
    While  i <=  N do
    begin
      If T[i] mod 3 = 0 then
        S := S + T[i];
      i := i + 1;
    end ;

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

Un programme en ligne, que vous pouvez exécuter et avoir des résultats, est disponible sur le lien suivant : https://onlinegdb.com/OAt7D8ttc

## 5- Ré-écrire l'algorithme en remplaçant la boucle Pour par la boucle Répéter

De la même façon que la boucle Tant-que, la boucle Répéter possède une condition. Sauf que pour cette dernière (la boucle répéter), la condition représente la condition d'achèvement (terminaison) de la boucle. Voici le modèle de transformation de la boucle Pour vers la boucle Répéter :

**Algorithmique**

```
Pour cpt ← vi à vf Faire
    <Bloc_Inst_Pour> ;
Fin-Pour;
```

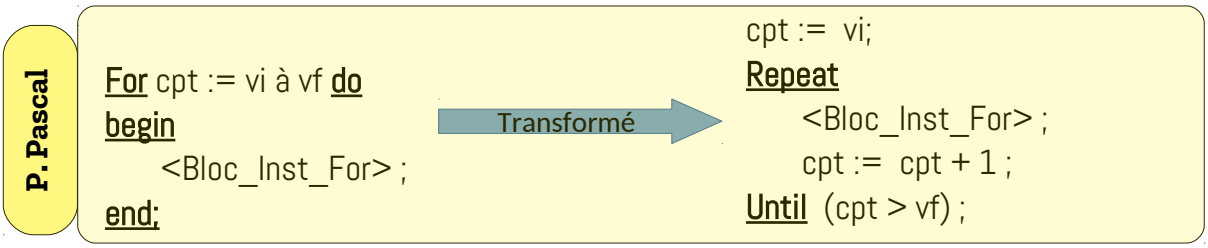
Transformé

```
cpt ← vi;
Répéter
    <Bloc_Inst_Pour> ;
    cpt ←  cpt + 1 ;
Jusqu'à  (cpt > vf) ;
```

**Telque :**

cpt : compteur(varible entière)
vi : valeur initiale
vf : valeur finale

En pascal :




En appliquant la traduction ci-dessus, nous aurons l'algorithme et le programme Pascal suivants :

**Algorithme**

```
Algorithme TP1_Exo1_Q5;
  Variables
      T : Tableau[1..100] de entier;
      N, i : entier;
      S : entier;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(T[i]);
    Fin-Pour;

    {-*-*-*- Traitement -*-*-*- }
    S ← 0;
    i ← 1;
    Répéter
        Si T[i] mod 3 = 0 Alors
          S ← S + T[i];
        Fin-si
        i ← i+1;
    Jusqu'à i>N;

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

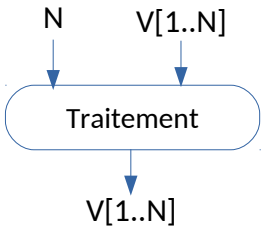
**Programme PASCAL**

```
Program TP1_Exo1_Q5;
  Var
      T : Array[1..100] of integer ;
      N, i : integer ;
      S : integer ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur T :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur T  :');
    For i:=1 to N do
      Read(T[i]);

    { -*-*-*- Traitement -*-*-*- }
    S := 0;
    i := 1;
    Repeat
      If T[i] mod 3 = 0 then
        S := S + T[i];
      i := i + 1;
    Until i>N;

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

De la même façon, le programme ci-dessus est disponible sur le lien : https://onlinegdb.com/DZ1QH80xd (Exécution en ligne).

## EXERCICE N°02 : LECTURE & AFFICHAGE D'UN VECTEUR

Un vecteur est une suite de cases mémoires de même type. Ces cases sont adjacentes et contiguës (l'une après l'autre). Dans l'algorithme à réaliser, qui contient aucun traitement, nous aurons un vecteur d'entrée de taille N et le même vecteur en sortie. Le schéma suivant illustre les variables d'entrée et de sortie :

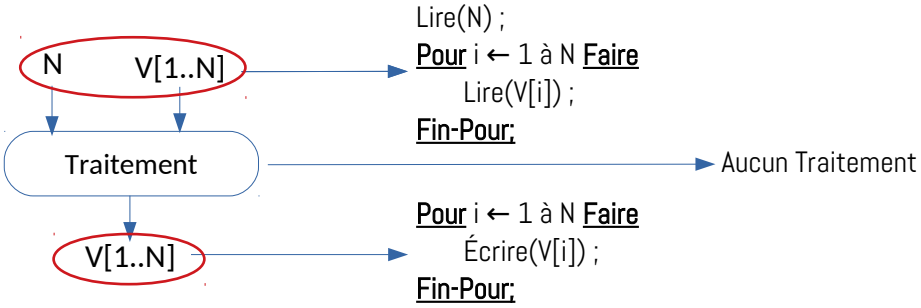



Comme vous l'avez vu au niveau du cours, la lecture d'un vecteur d'un vecteur V de taille N, nécessite :

- *Lire la taille du vecteur V : N*
- *Lire les composantes du vecteur V : V[1], V[2], ..., V[N]*

La même chose pour l'affichage d'un vecteur, nous devons afficher toutes les cases de 1 jusqu'à N.

Pour réaliser ça, nous utilisons une boucle (pour) comme suit :




(Voir le lien : https://onlinegdb.com/yc5H_FJ4S).

**Algorithme**

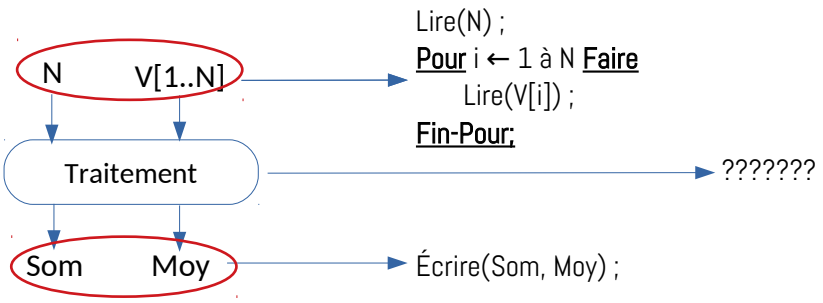
```
Algorithme TP1_Exo2_Q1;
  Variables
      V : Tableau[1..100] de réel;
      N, i : entier;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(V[i]);
    Fin-Pour;

    {-*-*-*- Traitement -*-*-*- }

    {-*-*-*- Sortie -*-*-*- }
    Pour i←1 à N faire
      Écrire(V[i]);
    Fin-Pour
Fin.
```

**Programme PASCAL**

```
Program TP1_Exo2_Q1;
  Var
      V : Array[1..100] of real;
      N, i : integer;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur V :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur V  :');
    For i:=1 to N do
      Read(V[i]);

    {-*-*-*- Traitement -*-*-*- }

    {-*-*-*- Sortie -*-*-*- }
    Writeln('Les valeur du vecteur V : ') ;
    for i:=1 to N do
      write(V[i]:10:2);
End.
```

## EXERCICE N°03 : SOMME ET MOYENNE DES ÉLÉMENTS D'UN VECTEUR

*Premièrement* nous commençons par le schéma Entrées / Traitement / Sorties, en écrivant le fragment de l'algorithme associé aux entrées et sorties :




*Deuxièmement*, après avoir déterminer les entrées et sortie, nous attaquons la partie traitement qui contient la séquence d'instructions qui calculent les sorties en fonction des entrées.

Le calcul de la somme des éléments de V – la variable Som :

Tel que Som représente la somme des éléments V[1], V[2], … V[N], donc :

$$\textbf{Som = V[1] + V[2] + V[3] +... + V[i] + .... + V[N]} = \sum_{i=1}^{N} V[i]$$

Comme nous l'avons pendant le semestre 1, cette somme devient :

```
Som ← 0 ;
Pour i ←1 à N faire
    Som ← Som + V[i] ;
Fin-Pour;
```

La moyenne des éléments du vecteur V de taille N est comme suit :

**Moy = Som / N**

Donc, ça sera une affectation : Moy ← Som / N :

(Voir le lien : https://onlinegdb.com/_RrAASKTA).

**Algorithme**

```
Algorithme TP1_Exo3_Q1;
  Variables
      V : Tableau[1..100] de réel;
      N, i : entier;
      Som, Moy : réel ;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(V[i]);
    Fin-Pour;

    {-*-*-*- Traitement -*-*-*- }
    Som ← 0 ;
    Pour i ← 1 à N faire
        Som ← Som + V[i] ;
    Fin-Pour ;
    Moy ← Som / N ;

    {-*-*-*- Sortie -*-*-*- }
    Écrire (Som, Moy);
Fin.
```

**Programme PASCAL**

```
Program TP1_Exo3_Q1;
  Var
      V : Array[1..100] of real;
      N, i : integer; Som, Moy : real ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur V :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur V  :');
    For i:=1 to N do
      Read(V[i]);

    {-*-*-*- Traitement -*-*-*- }
    Som := 0;
    For i:=1 to N do
       Som := Som + V[i];
    Moy := Som / N;

    {-*-*-*- Sortie -*-*-*- }
    Writeln('La somme du vecteur V  =  ', Som:0:3) ;
    Writeln('La moyenne du vecteur V  =  ', Moy:0:3) ;
End.
```

- Dérouler l'algorithme pour N=5 et V = [5, 2, 8, 7, 14]

| Instructions | Variables (mémoire centrale: RAM) | | | | | Affichage |
|---|---|---|---|---|---|---|
| | N | i | V | Som | Moy | |
| Lire(N) | 5 | / | / | / | / | |
| Pour i←1 à N=5faire<br>Lire(V[i]) ; | // | 1..5 | 1 2 3 4 5<br>[ 5 , 2 , 8 , 7 , 14 ] | / | / | |
| Som ← 0 ; | // | 5 | // | 0 | / | |
| Pour i=1<br>Som ← som + V[i]<br>Som ← 0 + V[1] = 0+5=5 | // | 1 | i=1 2 3 4 5<br>[ 5 , 2 , 8 , 7 , 14 ] | 5 | | |
| Pour i=2<br>Som ← som + V[i]<br>Som ← 5 + V[2]=5+2=7 | // | 2 | 1 i=2 3 4 5<br>[ 5 , 2 , 8 , 7 , 14 ] | 7 | | |
| Pour i=3<br>Som ← som + V[i]<br>Som ← 7 + V[3]=7+8=15 | // | 3 | 1 2 i=3 4 5<br>[ 5 , 2 , 8 , 7 , 14 ] | 15 | | |
| Pour i=4<br>Som ← som + V[i]<br>Som ←5+V[4]=15+7=22 | // | 4 | 1 2 3 i=4 5<br>[ 5 , 2 , 8 , 7 , 14 ] | 22 | | |
| Pour i=5<br>Som ← som + V[i]<br>Som←5+V[5]=22+14=36 | // | 5 | 1 2 3 4 i=5<br>[ 5 , 2 , 8 , 7 , 14 ] | 36 | | |
| Moy ← Som / 5<br>Moy ← 36/5 = 7.2 | // | // | // | | 7.2 | |
| Écrire (Som, Moy) | 5 | | [ 5 , 2 , 8 , 7 , 14 ] | 36 | 7.2 | 36 7.2 |

Vous remarquez que dans le tableau de déroulement ci-dessus, la colonne affichage n'est pas vraiment importante puisque les valeurs des variables sont visibles dans leurs colonnes respectifs et que l'affichage est utilisé uniquement à la dernière instruction : écrire (Som, Moy).

Aussi, c'est bien de remarquer le déroulement des deux boucles Pour :

- La première, qui concerne la lecture du vecteur V, a été déroulée d'une façon direct, puisque c'est n'est pas important de la détailler montre uniquement les valeurs de V et sa taille
- La seconde, qui est dans la partie traitement, a été exécuté d'une façon détaillée, puisque c'est important de montrer la façon dont le traitement a été réalisé.

## EXERCICE N°04 : INVERSER LES ÉLÉMENTS D'UN VECTEUR

### 1- Inverser les éléments du vecteur T dans un autre vecteur V

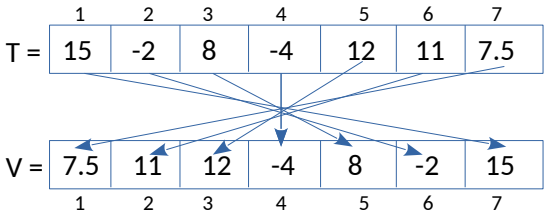
Pour illustrer le problème à résoudre nous prenons l'exemple suivant :

| T = | 15 | -2 | 8 | -4 | 12 | 11 | 7.5 |
|---|---|---|---|---|---|---|---|

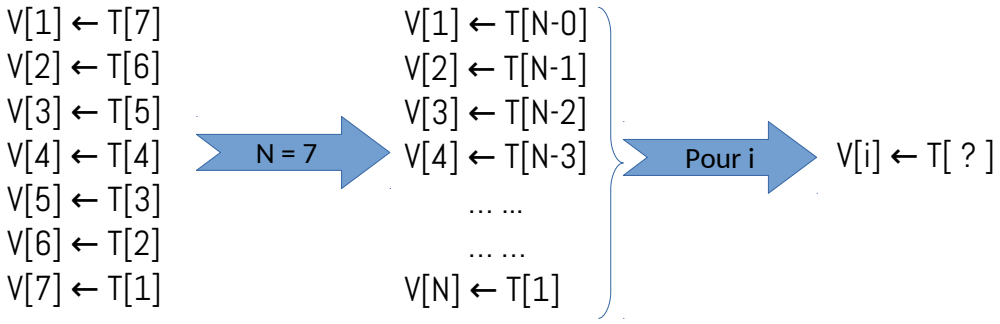
Le vecteur V sera :

| V = | 7.5 | 11 | 12 | -4 | 8 | -2 | 15 |
|---|---|---|---|---|---|---|---|

On peut schématiser ceci comme suit :




On comprend que la taille du vecteur V est la même que celle du vecteur T, et pour cet exemple N = 7. Nous pouvons écrire les affectations suivantes :




On doit faire une formule générale pour les affectations V[i] ← T[ ? ] / pour i=1,N.

D'après les affectations ci-dessus :

L'indice 1 correspond à N = N-0 = N-(1-1) = N - 1 + 1

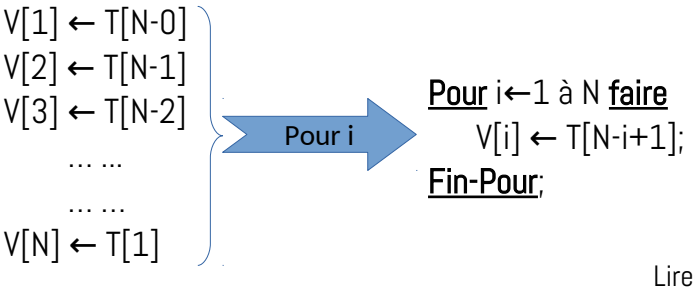
L'indice 2 correspond à N-1 = N-(2-1) = N - 2 + 1

L'indice 3 correspond à N-2 = N-(3-1) = N - 3 + 1

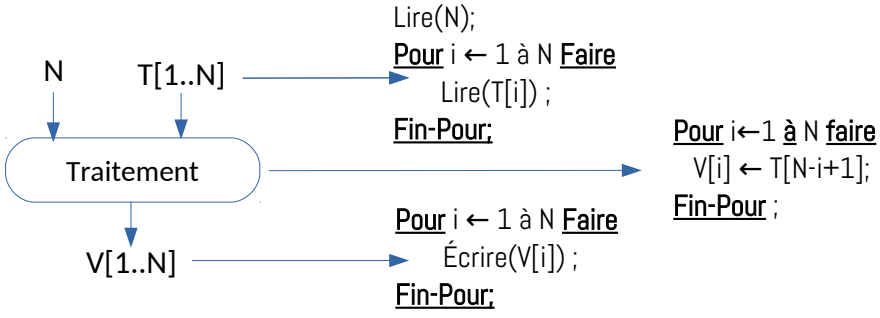
L'indice 4 correspond à N-3 = N-(4-1) = N - 4 + 1

On déduit que : L'indice i correspond à N-(i-1) = N-i+1

Donc :




Pour le schéma d'entrée/ sortie, ça sera comme suit :



Tous les éléments de l'algorithme : Entrée / Traitement / Sortie sont prêts, on peut écrire l'algorithme suivant :

**Algorithme**

```
Algorithme TP1_Exo4_Q1_methode_1;
  Variables
       T,V : Tableau[1..100] de réel;
       N, i : entier;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(T[i]);
    Fin-Pour;

    {-*-*-*- Traitement -*-*-*- }
    Pour i ← 1 à N faire
        V[i] ← T[N-i+1] ;
    Fin-Pour ;


    {-*-*-*- Sortie -*-*-*- }
    Pour i ← 1 à N faire
        Écrire( V[i] );
    Fin-Pour ;
Fin.
```

**Programme PASCAL**

```
Program TP1_Exo4_Q1_methode_1;
  Var
       T, V : Array[1..100] of real;
       N, i : integer;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur T :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur T  :');
    For i:=1 to N do
      Read(T[i]);

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


    {-*-*-*- Sortie -*-*-*- }
    Writeln('Le vecteur V est : ') ;
    For i:=1 to N do
      write(V[i]:10:2);
End.
```

Voir le lien suivant : https://onlinegdb.com/ZdcqMIlp7, pour exécuter en ligne le programme ci-dessus.

Dans l'algorithme ci-dessus, nous avons utilisé un seule indice (compteur) i et la formule (N-i+1) pour trouver la case correspondante à l'indice i. Une autre méthode consiste à utiliser deux indices i et j tel-que i varie de 1 à N avec un pas=1 et j varie de N à 1 avec un pas = -1. comme indiqué ci-dessous ( voir le lien : https://onlinegdb.com/ntvLAIxBL ):

**Algorithme**

```
Algorithme TP1_Exo4_Q1_methode_2;
  Variables
       T,V : Tableau[1..100] de réel;
       N, i, j : entier;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(T[i]);
    Fin-Pour;
    {-*-*-*- Traitement -*-*-*- }
    j ← N;
    Pour i ← 1 à N faire
        V[i] ← T[j] ;
        J ← j-1;
    Fin-Pour ;
    {-*-*-*- Sortie -*-*-*- }
    Pour i ← 1 à N faire
        Écrire( V[i] );
    Fin-Pour ;
Fin.
```

**Programme PASCAL**

```
Program TP1_Exo4_Q1_methode_2;
  Var
       T, V : Array[1..100] of real;     N, i,j : integer;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur T :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur T  :');
    For i:=1 to N do
      Read(T[i]);
    {-*-*-*- Traitement -*-*-*- }
    j ← N;
    For i:=1 to N do
    begin
      V[i] := T[j];
      j ← j-1;
    end ;
    {-*-*-*- Sortie -*-*-*- }
    Writeln('Le vecteur V est : ') ;
    For i:=1 to N do
      write(V[i]:10:2);
End.
```

## 2- Inverser les éléments du vecteur T dans lui même

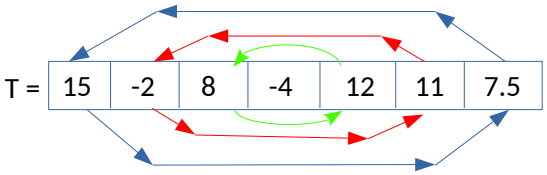
De la même façon, nous prenons l'exemple ci-dessus pour illustrer ce problème :

| T = | 15 | -2 | 8 | -4 | 12 | 11 | 7.5 |
|---|---|---|---|---|---|---|---|

Le vecteur T doit être à la fin de l'algorithme :

| T = | 7.5 | 11 | 12 | -4 | 8 | -2 | 15 |
|---|---|---|---|---|---|---|---|

Ceci sera réalisé par des permutations entre cases, comme illustré par ce schéma :



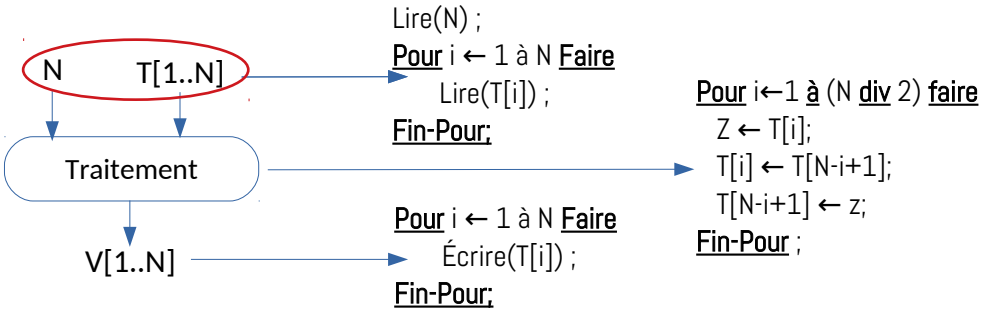

Vous remarquez qu'il 3 permutations (nombre de couleurs bleu, rouge et vert) :
T[1] <-->T[7] , T[2] <-->T[6] et T[3] <-->T[5] et T[4] est une case médiane (au milieu).

Le nombre de permutations = N div 2 (la moitié (entière) de la taille de vecteur) et chaque i (i allant de 1 à N div 2) sera permutée avec la case N-i+1, comme suit :

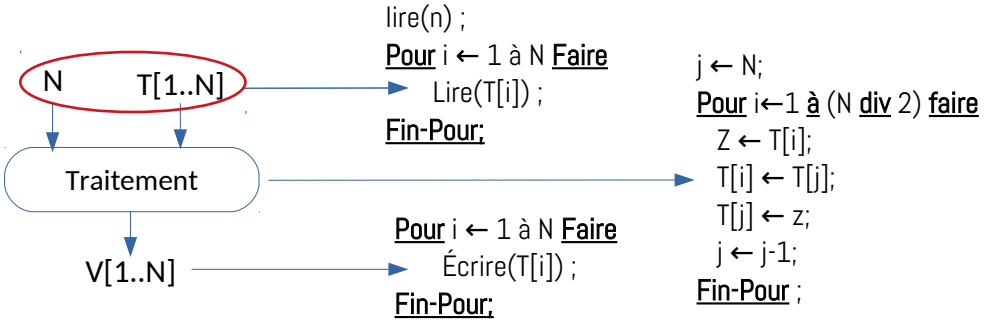
Z ← T [i]; T[i] ← T[N-i+1]; T[N-i+1] ← z;

Bien sûr, Z doit être une variable de type des cases du vecteur T (dans notre cas, T est un vecteur réel, donc Z est réel).

Donc, nous aurons le schéma d'entrée/sortie suivant :




Une deuxième méthode, comme la question 01, consiste à utiliser un autre indice j, comme suit :



L'algorithme et le programme PASCAL de la méthode 01 sont les suivants (le lien : https://onlinegdb.com/_hsjEklkt) :

**Algorithme**

```
Algorithme TP1_Exo4_Q2_methode_1;
  Variables
       T : Tableau[1..100] de réel;
       N, i : entier;  Z:réel;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(T[i]);
    Fin-Pour;
    {-*-*-*- Traitement -*-*-*- }
    Pour i ← 1 à (N div 2) faire
        Z ← T[i] ;
        T[i] ← T[N-i+1] ;
       T[N-i+1] ← Z;
    Fin-Pour ;
    {-*-*-*- Sortie -*-*-*- }
    Pour i ← 1 à N faire
        Écrire( T[i] );
    Fin-Pour ;
Fin.
```

**Programme PASCAL**

```
Program TP1_Exo4_Q2_methode_1;
  Var
       T: Array[1..100] of real;     N, i : integer;   Z:real ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur T :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur 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 ;
    {-*-*-*- Sortie -*-*-*- }
    Writeln('Le vecteur V est : ') ;
    For i:=1 to N do
       write(T[i]:10:2);
End.
```

Pour la méthode N°02, voir ci-dessous ( lien : https://onlinegdb.com/vVSt1rRRv ) :

**Algorithme**

```
Algorithme TP1_Exo4_Q2_methode_2;
  Variables
       T : Tableau[1..100] de réel;
       N, i, j : entier;  Z:réel;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(T[i]);
    Fin-Pour;
    {-*-*-*- Traitement -*-*-*- }
    j ← N;
    Pour i ← 1 à (N div 2) faire
        Z ← T[i] ;
        T[i] ← T[j] ;
       T[j] ← Z;
        J ← j-1;
    Fin-Pour ;
    {-*-*-*- Sortie -*-*-*- }
    Pour i ← 1 à N faire
        Écrire( T[i] );
    Fin-Pour ;
Fin.
```

**Programme PASCAL**

```
Program TP1_Exo4_Q2_methode_2;
  Var
       T: Array[1..100] of real;  N, i,j : integer;   Z:real ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur T :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur T  :');
    For i:=1 to N do
      Read(T[i]);
    {-*-*-*- Traitement -*-*-*- }
    j ← N;
    For i:=1 to (N div 2)  do
    begin
       Z:= T[i] ;
       T[i] := T[j];
       T[j]:= Z ;
       j ← j-1;
    end ;
    {-*-*-*- Sortie -*-*-*- }
    Writeln('Le vecteur V est : ') ;
    For i:=1 to N do
       write(T[i]:10:2);
End.
```

## EXERCICE N°05 : RECHERCHER LE MIN ET LE MAX DANS UN VECTEUR

### 1- Rechercher la plus petite valeur dans un vecteur (le Min d'un vecteur)

Pour trouver la plus petite valeur dans un vecteur V, il suffit de connaitre son emplacement pmin (sa position : l'indice de la case) dans ce vecteur. La recherche de pmin sera faite en suivant les étapes suivantes :

1- On suppose que la plus petite valeur se trouve dans le premier emplacement : pmin ← 1

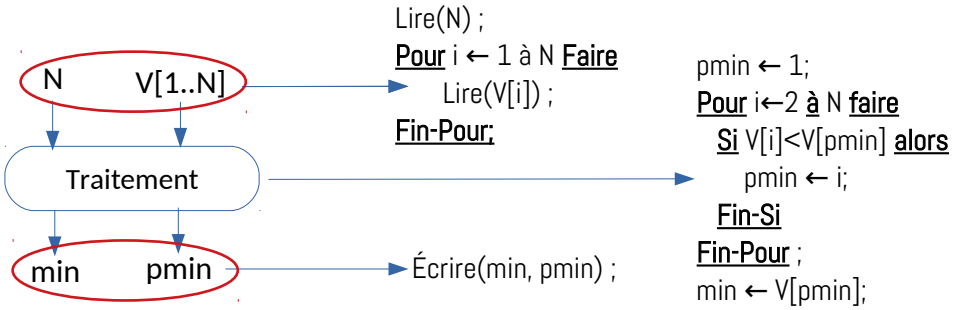
2- On parcours toute les cases i , tel-que i varie de 2 jusqu'à la taille du vecteur V : N

3- à chaque case i (tel-que i=$\overline{2,N}$), on compare entre les deux cases d'indice i et pmin, c'est à dire entre les cases : V[i] et V[pmin]. Le cas le plus pertinent est : V[i] < V[pmin], dans ce cas, pmin doit être ajuster (corriger) en affectant i à pmin (pmin ← i), ceci sera comme suit :

```
Pour i←2 à N faire
        Si V[i] < VT[pmin] alors
                pmin ← i ;
        Fin-Si ;
Fin-Pour;
```

4- Une fois la position du minimum est connue, alors le miminum min est juste l'affectation suivante : min ← V[pmin];

L'analyse ci-dessus, peut être regroupée dans le schéma suivant :




**Algorithme**

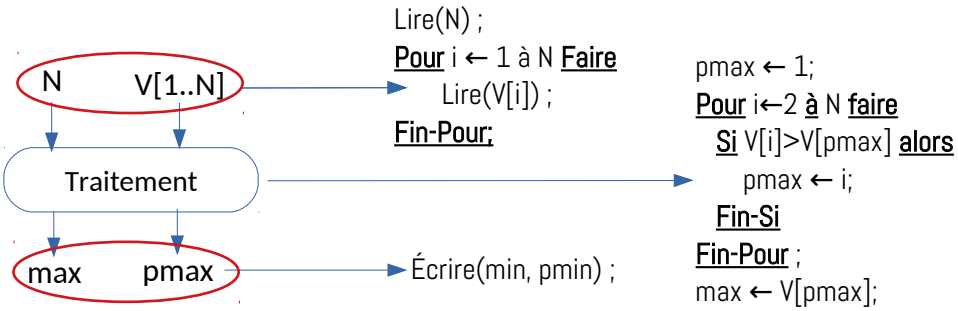
```
Algorithme TP1_Exo5_Q1_Minimum;
  Variables
    V : Tableau[1..100] de réel;
    N, i, pmin : entier;  min : réel;
Début
  { -*-*-*- Entrées -*-*-*- }
  Lire(N) ;
  Pour i←1 à N faire
    Lire(V[i]);
  Fin-Pour;
  {-*-*-*- Traitement -*-*-*- }
  Pmin ← 1 ;
  Pour i ← 2 à N faire
      Si V[i] < V[pmin] alors
          pmin ← i;
     Fin-Si ;
  Fin-Pour ;
  Min ← V[pmin];
  {-*-*-*- Sortie -*-*-*- }
  Écrire(min, pmin);
Fin.
```

**Programme PASCAL**

```
Program TP1_Exo5_Q1_Minimum;
  Var
     V: Array[1..100] of real;   N, i, pmin : integer; min:real ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur V :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur V  :');
    For i:=1 to N do
      Read(V[i]);
    {-*-*-*- Traitement -*-*-*- }
    Pmin := 1;{On suppose que le min se trouve à l'indice 1}
    For i:=2 to N  do
      If V[i]<V[pmin] then
           pmin := i ;
    Min:= V[pmin];
    {-*-*-*- Sortie -*-*-*- }
    Writeln('Le vecteur minimale du V est : ', min:0:2) ;
    Writeln('Sa position est : ', pmin) ;
End. {Voir le lien : https://onlinegdb.com/yiLT6DPT9}
```

## 2- Rechercher la plus grande valeur dans un vecteur (le Max d'un vecteur)

La même idée qui sera appliquée pour la recherche du maximum et sa position dans un vecteur. Il suffit d'inverser la comparaison : V[i]>V[pmax]. Donc, nous aurons le schéma d'entrée/sortie suivant :




Ainsi, et avec la même logique de construction d'algorithme, nous aurons ce qui suit comme algorithme et traduction en programme PASCAL :

**Algorithme**

```
Algorithme TP1_Exo5_Q2_Maximum;
  Variables
    V : Tableau[1..100] de réel;
    N, i, pmax : entier;  max : réel;
Début
   { -*-*-*- Entrées -*-*-*- }
   Lire(N) ;
   Pour i←1 à N faire
     Lire(V[i]);
   Fin-Pour;

   {-*-*-*- Traitement -*-*-*- }
   pmax ← 1 ;
   Pour i ← 2 à N faire
       Si V[i] > V[pmax] alors
            pmax ← i;
      Fin-Si ;
   Fin-Pour ;
   max ← V[pmax];


   {-*-*-*- Sortie -*-*-*- }
   Écrire(max, pmax);
Fin.
```

**Programme PASCAL**

```
Program TP1_Exo5_Q2_Maximum;
  Var
      V: Array[1..100] of real;
      N, i, pmax : integer;
      max : real ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur V :') ;
    Read(N) ;
    Writeln('Donner les valeurs du vecteur V  :');
    For i:=1 to N do
      Read(V[i]);

    {-*-*-*- Traitement -*-*-*- }
    pmax := 1;{Supposer que le max se trouve à l'indice 1}
    For i:=2 to N  do
      If V[i] > V[pmax] then
            pmax := i ; { pmax reçoit la valeur de i }
    max:= V[pmax];

    {-*-*-*- Sortie -*-*-*- }
    Writeln('Le vecteur maximale du V est : ', max:0:2) ;
    Writeln('Sa position est : ', pmax) ;
End.
```

Voir le lien suivant : https://onlinegdb.com/MimIBk79Q , pour exécuter une version du programme précédent.

**Remarque :**

Dans la recherche du minimum et du maximum dans un vecteur quelconque, le plus important est leur position et non leur valeur (si on connais la position on connaîtra la valeur).

# Série d'Exercices Supplémentaires sur les Vecteurs (Tableaux à une Dimension)

# TP INFORMATIQUE – SEMESTRE 2

## EXERCICES SUPPLÉMENTAIRES SUR LES VECTEURS

### EXERCICE SUP.-01 : LA RECHERCHE D'UNE VALEUR DANS UN VECTEUR

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

### EXERCICE SUP.-02 : PERMUTATION ENTRE LES CASES D'INDICE K ET L

Soit V un vecteur de type réel et de taille N, et soient K et L deux positions dans le vecteur V. Écrire un algorithme / Programme PASCAL qui permet de permuter entre les deux éléments du vecteur V, d'indice K et L.

### EXERCICE SUP.-03 : TRIER UN VECTEUR AVEC UN ORDRE CROISSANT

Soit V un vecteur de type réel de taille N. Écrire un algorithme/programme PASCAL qui permet de trier (ordonner) les éléments du vecteur V avec un ordre croissant.

### EXERCICE SUP.-04 : FRAGMENTER UN VECTEUR

Soit V un vecteur de type réel de taille N. Écrire un algorithme/programme PASCAL qui permet de fragmenter (diviser) sur deux vecteurs T1 et T2, telque, T1 contiendra les éléments strictement négatif (<0) de V et et T2 contiendra les éléments positifs (>=0) de V.

### EXERCICE SUP.-05 : SOMME ET PRODUIT SCALAIRE DE DEUX VECTEURS

Soit V1 et V2 deux vecteurs de type réel de taille N. Écrire un algorithme/programme PASCAL qui permet de calculer la somme et le produit scalaire de V1 et V2.

Remarque : – La somme de deux vecteurs est un vecteur – Le produit scalaire de deux vecteurs est une valeur scalaire (réelle).

### EXERCICE SUP.-06 : 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 produits 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 SUP.-07 : CONVERTIR UN NOMBRE DE BASE 10 VERS BASE 2

Soit *Nb* un nombre entier positifs écrit en base 10. Écrire 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.

# Solution – Série d'Exercices Supplémentaires

## EXERCICE SUP. N°01 : RECHERCHE D'UNE VALEUR DANS UN VECTEUR

La recherche d'une valeur X dans un vecteur V donne deux résultats :

- Soit X ne se trouve pas dans V : V ne contient pas la valeur de X
- Soit X se trouve dans V dans telle position

On prend un exemple, comme nous avons l'habitue de le faire, afin de comprendre le problème à résoudre. Soit les deux scénarios d'exécution suivants :

*Scénario 1 :*

Donner la taille du vecteur T : 6
Donner les valeurs du vecteur T : 15 3 12 66 33 -9
Donner la valeur de X à rechercher : 68
---
La valeur de X = 68.00 ne se trouve pas dans le vecteur V

*Scénario 2 :*

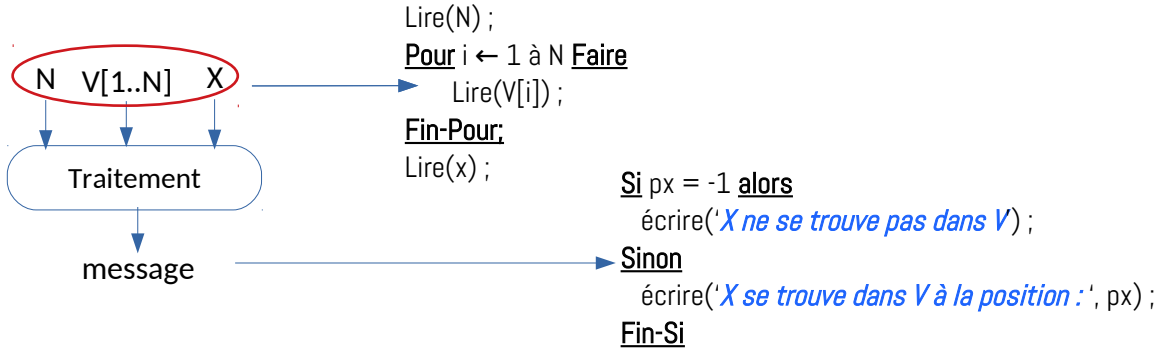
Donner la taille du vecteur T : 6
Donner les valeurs du vecteur T : 15 3 12 66 33 -9
Donner la valeur de X à rechercher : 66
---
La valeur de X = 66.00 se trouve dans le vecteur V à la position : 4

Donc, on déduit que :

- Les variables d'entrées sont :
  - *) N : la taille du vecteur V
  - *) V[1..N] : les valeurs des composantes du vecteur V
  - *) X : la valeur à rechercher dans V
- Les variables de sortie sont :
  - Message qui peut être soit :
    - *) X ne se trouve pas dans le vecteur V (ou V ne contient pas la valeur de X)
    - *) X se trouve dans le vecteur V à la position : px

Soit la variable px qui représente la position de x dans le vecteur V. On met px = -1 pour supposer (initialement) que X ne se trouve pas dans V. après la recherche de la valeur de X, nous aurons deux cas :

- Soit px = -1 , donc, la valeur de X ne se trouve pas dans V
- Soit px <> -1, donc, la valeur de X se trouve dans V à la position px



Ce qui reste à réaliser, est la partie Traitement. L'idée est d'utiliser une boucle Tant-que avec une condition composée : on recherche la valeur de X dans le vecteur V tant-que nous n'avons pas atteint la fin du vecteur V et on n'a pas encore trouvé la valeur de X.

C'est-à-dire, l'indice i ne dépasse pas la taille du vecteur (la valeur N) *ET* px est toujours égale à -1. Ceci peut être écrit comme suit :

Tant-que (i <= N) ET (px = -1) Faire

i<=N : on n'as pas encore atteint la fin du vecteur V

px = -1 : on n'as pas encore trouvé la valeur de x dans le vecteur V

Bien évidemment, il faut initialiser i à 1 et px à -1. Et à chaque itération de la boucle Tant-que, on compare la valeur de V[i] à celle de x

Si (V[i] = X) Alros

Et dans le cas d'égalité, on indique la position : px ← i ; et automatiquement, la condition de la boucle Tant-que deviendra fausse à cause que px sera différent de -1.

En regroupant tout ce qui a été analysé ci-dessus, nous aurons la partie traitement suivante :

```
px ← -1 ; {On suppose initialement que X ne se trouve pas dans le vecteur V}
i ← 1 ; {On commence la recherche par la première case}
Tant-que (i <= N) ET (px = -1) Faire { i<= N : on n'as teriminé le vecteur ET px=-1 : on n'aps encore trouvé la vleur de X}
        Si V[i] = X Alors {Si la valeur de la case V[i] est égale à la valeur de X}
                px ← i; {On a trouver la valeur de X à la position i dans vecteur V}
        Fin-Si;
        i ← i+1; {Aller à la case suivante}
Fin-Tant-que;
```

Avec tout ça nous aurons l'algorithme et le programme PASCAL (voir le lien : https://onlinegdb.com/dsvznCiOF ) suivant :

**Algorithme**

```
Algorithme TP1_Supp_Exo1_Recherche;
  Variables
    V : Tableau[1..100] de réel;
    N, i, px : entier;  X : réel;
Début
   { -*-*-*- Entrées -*-*-*- }
   Lire(N) ;
   Pour i←1 à N faire
     Lire(V[i]);
   Fin-Pour;
   Lire(x);
   {-*-*-*- Traitement -*-*-*- }
   px ← -1 ;  i ← 1;
   Tant-que (i <= N) ET (px = -1) faire
      Si V[i] = X alors
           px ← i;
      Fin-Si ;
      i ← i +1;
   Fin-Tant-que ;
   {-*-*-*- Sortie -*-*-*- }
   Si px = -1 alors
     écrire('X ne se trouve pas dans V')
   Sinon
     écrire('X se trouve dans le vecteur V à : ', px);
   Fin-Si;
Fin.
```

**Programme PASCAL**

```
Program TP1_Supp_Exo1_Recherche;
  Var
      V: Array[1..100] of real;
      N, i, px : integer;   x : real ;
Begin
   { -*-*-*- Entrées -*-*-*- }
   Write('Donner la taille du vecteur V : ') ;
   Read(N) ;
   Write('Donner les valeurs du vecteur V  : ');
   For i:=1 to N do
     Read(V[i]);
   Write('Donner la valeur de X  : ') ; Read(X);
   {-*-*-*- Traitement -*-*-*- }
   px := -1;  i:= 1 ;
   while (i <= N) and (px = -1) do
   begin
     If V[i] = X then
           px := i ;
     i := i + 1;
   end;
   {-*-*-*- Sortie -*-*-*- }
   if px = -1 then
      Write('X ne se trouve pas dans le vecteur V')
   else
      Write('X se trouve dans V à la position : ', px) ;
End.
```

Une autre méthode consiste à utiliser un booléen *Trouve*, qui vaut *False* si la valeur ne se trouve pas dans V, et vaut *True* si la valeur de x se trouve dans le vecteur V, comme suit (Voir aussi le lien suivant : https://onlinegdb.com/RyyZGlOUb ) :

**Algorithme**

```
Algorithme TP1_Supp_Exo1_Recherche_methode_2;
   Variables
      V : Tableau[1..100] de réel;
      N, i, px : entier;  X : réel;
      Trouve : booléen;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(V[i]);
    Fin-Pour;
    Lire(x);
    {-*-*-*- Traitement -*-*-*- }

    i ← 1;
    Trouve ← false;
    Tant-que (i <= N) ET  Not(Trouve)  faire
        Si V[i] = X alors
             Trouve ← true;
        Sinon
             i ← i +1;
        Fin-Si ;
    Fin-Tant-que ;
    px ← i;

    {-*-*-*- Sortie -*-*-*- }
    Si Not(Trouve) alors
      écrire('X ne se trouve pas dans V')
    Sinon
      écrire('X se trouve dans le vecteur V à : ', px);
    Fin-Si;
Fin.
```

**Programme PASCAL**

```
Program TP1_Supp_Exo1_Recherche_methode_2;
  Var
      V: Array[1..100] of real;
      N, i, px : integer;    x : real ;
      Trouve : boolean;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur V : ') ;
    Read(N) ;
    Write('Donner les valeurs du vecteur V  : ');
    For i:=1 to N do
      Read(V[i]);
    Write('Donner la valeur de X  : ') ;
    Read(X);

    {-*-*-*- Traitement -*-*-*- }
    i:= 1 ;
    Trouve := false;
    while (i <= N) and Not(Trouve) do
      If V[i] = X then
            Trouve := True;
      else
            i := i + 1;
    px ← i;

    {-*-*-*- Sortie -*-*-*- }
    if Not(Trouve) then
       Write('X ne se trouve pas dans le vecteur V')
    else
       Write('X se trouve dans V à la position : ', px) ;
End.
```

Vous remarquez, pendant l'exécution de ce type de programme, on souhaite réaliser plusieurs recherches, avec différentes valeurs de x et les même valeurs du Vecteur V, sans quitter le programme. Puisque c'est fatiguant de ré-exécuter le programme et de refaire la ressaisie du vecteur lui même. Il suffit d'ajouter une boucle **Répéter** … **Jusqu'à**, comme indiquer sur le programme PASCAL sur le lien suivant : https://onlinegdb.com/7hs4p18oK

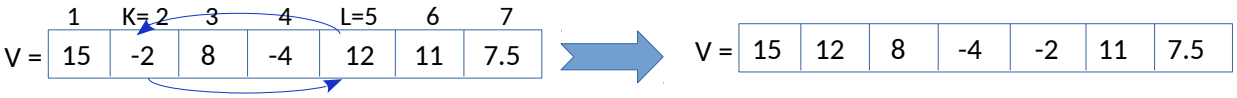
J'espère que cet exercice est clair pour vous, si vous avez des questions n'hésitez pas à les poser sur e-learning ou sur le mail : redouane.ouzeggane@gmail.com (😊).

## EXERCICE SUP. N°02 : PERMUTATION ENTRE LES CASES D'INDICE K ET L

Nous prenons l'exemple suivant pour illustrer le problème de permutation entre deux cases d'indice K et L :

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| V = | 15 | -2 | 8 | -4 | 12 | 11 | 7.5 |

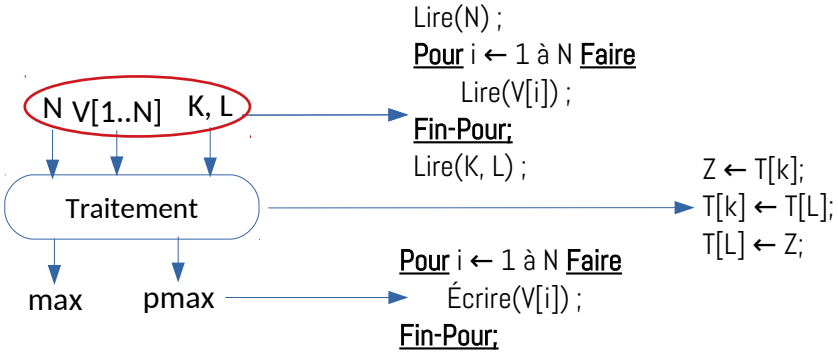
K = 2 et L =5, donc on veut permuter entre la case d'indice 2 et la case d'indice 5 :




Donc, nous devons permuter les valeurs des cases V[K] et V[L] :

Z ← T[k] ;
T[K] ← T[L] ;
T[L] ← Z ;

Nous aurons le schéma suivant :




### Algorithme

```
Algorithme TP1_Supp_Exo2_Permutations;
   Variables
      V : Tableau[1..100] de réel;
      N, i, K, L : entier;  Z : réel;
Début
   { -*-*-*- Entrées -*-*-*- }
   Lire(N) ;
   Pour i←1 à N faire
     Lire(V[i]);
   Fin-Pour;
   Lire(K, L) ;
   {-*-*-*- Traitement -*-*-*- }
   Z ← V[K];
   V[K]←V[L] ;
   V[L]←Z ;

   {-*-*-*- Sortie -*-*-*- }
   Pour i←1 à N faire
     Écrire(V[i]);
   Fin-Pour;
Fin.
```

### Programme PASCAL

```
Program TP1_Supp_Exo2_Permutations;
  Var
      V: Array[1..100] of real;
      N, i, K, L : integer;   Z : real ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur V : ') ;
    Read(N) ;
    Write('Donner les valeurs du vecteur V  : ');
    For i:=1 to N do
      Read(V[i]);

    Write('Donner la valeur de l'indice K  : ') ;
    Repeat
       Read(K) ;
    Until (K>=1)AND(K<=N);

    Write('Donner la valeur de l'indice L  : ') ;
    Repeat
       Read(L) ;
    Until (K>=1)AND(L<=N);

    {-*-*-*- Traitement -*-*-*- }
    Z := V[K] ;
    V[K] :=V[L] ;
    V[L] := Z ;

    {-*-*-*- Sortie -*-*-*- }
    Writeln('Les valeurs des cases du vecteur V  : ');
    For i:=1 to N do
      Write(V[i]:10:2);
End.
```

Le programme ci-coté, est disponible sur le lien : https://onlinegdb.com/vQ2E4HL-N .

## EXERCICE SUP. N°03 : TRIER UN VECTEUR AVEC UN ORDRE CROISSANT

Nous allons voir deux méthodes pour le tri d'un vecteur :

- Tri par sélection : recherches multiples d'indice de minimums et permutation. Voir le lien suivant : https://fr.wikipedia.org/wiki/Tri_par_s%C3%A9lection
- Tri par propagation (Tri à bulles) : comparaisons d'éléments consécutives et permutations : https://fr.wikipedia.org/wiki/Tri_%C3%A0_bulles

### a- Méthode 01 : Tri par sélection

Soit V un vecteur de Taille N. Le tri du vecteur V en utilisant la méthode de sélection consiste à réaliser ceci :

- *Chercher* **pmin** l'indice du minimum dans le tableau de l'indice 1 jusqu'à N, et le permuter avec la position 1.

- *Chercher* **pmin** l'indice du minimum dans le tableau à partir de l'indic 2 jusqu'à N, et le permuter avec la position 2.

- *Chercher* **pmin** l'indice du minimum dans le tableau à partir de l'indic 3 jusqu'à N, et le permuter avec la position 3.

...

...

- *Chercher* **pmin** l'indice du minimum dans le tableau à partir de l'indic N-1 jusqu'à N, et le permuter avec la position N-1.

Vous remarquez que nous effectuons (N-1) recherches de minimums et les permuter avec la première case de début de chaque recherche. Donc, nous (N-1) étapes.

Les schémas suivants illustrent les étapes ci-dessus avec V =[-8 -2 12 -4 120 13 -7.5 ] :

**Étape 01 : Tout le Vecteur V de l'indice 1 jusqu'à N**

Rechercher pmin du 1 jusqu'à N=7 => pmin = 3

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| V = | 12 | -2 | -8 | -4 | 120 | 13 | -7.5 |

Permuter entre 3 et 1

Après permutation, le minimum du V sera mis dans la cas 1.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| V = | -8 | -2 | 12 | -4 | 120 | 13 | -7.5 |

**Étape 02 : Une partie du Vecteur V de l'indice 2 jusqu'à N**

Rechercher pmin de 2 jusqu'à N=7 => pmin = 7

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| V = | -8 | -2 | 12 | -4 | 120 | 13 | -7.5 |

Permuter entre 7 et 2

Après permutation, le deuxième minimum du V sera mis dans la cas 2.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | 12 | -4 | 120 | 13 | -2 |

**Étape 03 : Une partie du Vecteur V de l'indice 3 jusqu'à N**

Rechercher pmin de 3 jusqu'à N=7 => pmin = 4

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | 12 | -4 | 120 | 13 | -2 |

Permuter entre 4 et 3

Après permutation, le deuxième minimum du V sera mis dans la cas 2.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | -4 | 12 | 120 | 13 | -2 |

**Étape 04 : Une partie du Vecteur V de l'indice 4 jusqu'à N**

Rechercher pmin de 4 jusqu'à N=7 => pmin = 7

| | 1 | 2 | 3 | **4** | 5 | 6 | **7** |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | -4 | 12 | 120 | 13 | -2 |

Permuter entre 7 et 4

Après permutation, le deuxième minimum du V sera mis dans la cas 2.

| | 1 | 2 | 3 | **4** | 5 | 6 | **7** |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | -4 | -2 | 120 | 13 | 12 |

**Étape 05 : Une partie du Vecteur V de l'indice 5 jusqu'à N**

Rechercher pmin de 5 jusqu'à N=7 => pmin = 7

| | 1 | 2 | 3 | 4 | **5** | 6 | **7** |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | -4 | -2 | 120 | 13 | 12 |

Permuter entre 7 et 5

Après permutation, le deuxième minimum du V sera mis dans la cas 2.

| | 1 | 2 | 3 | 4 | **5** | 6 | **7** |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | -4 | -2 | 12 | 13 | 120 |

**Étape 06 : Une partie du Vecteur V de l'indice 6 jusqu'à N**

Rechercher pmin de 6 jusqu'à N=7 => pmin = 6

| | 1 | 2 | 3 | 4 | 5 | **6** | 7 |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | -4 | -2 | 12 | 13 | 120 |

Permuter entre 6 et 6

Pmin = 6 est la même que la première case de recherche, donc, pas de permutation

| | 1 | 2 | 3 | 4 | 5 | **6** | 7 |
|---|---|---|---|---|---|---|---|
| V = | -8 | -7.5 | -4 | -2 | 12 | 13 | 120 |

Vous remarquez qu'il y 6=N-1 étapes, puisque le vecteur contient 7 composantes (Nombre d'étapes = Taille du vecteur -1). Chacune d'elle est composé de deux opérations.

Soit l'étape i (*i* allant de 1 à N-1 / N est la taille d'un vecteur V).

L'étape *i* contient les deux opérations suivantes :

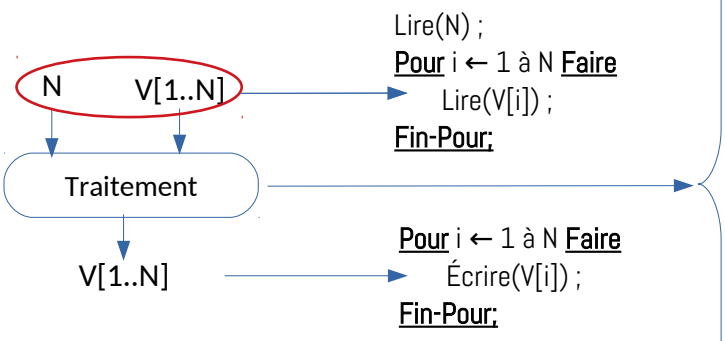
1) Recherche de la position du minimum (*pmin*) à partir de l'indice *i* jusqu'à l'indice N

```
pmin ← i ;
Pour j←(i+1) à N faire
   Si V[j] < V[pmin] alors
      pmin ← j;
   Fin-Si ;
Fin-Pour ;
```

2) Permuter les cases d'indices *pmin* et *i* (si bien sûr *pmin* est différent de *i*)

```
Si pmin <> i alors
   Z ← V[pmin] ;
   V[pmin] ← V[i];
   V[i] ← Z;
Fin-Si;
```

Nous allons regrouper toutes cette analyse dans le schéma des entrées/traitement/sorties, et par la suite nous écrivons l'algorithme et le programme PASCAL correspondant ( Voir le lien : https://onlinegdb.com/CXBIANob2 ) :




```
{Pour chaque étape i allant de 1 à (N-1)}
Pour i←1 à (N-1) faire
    {Opération 1 : chercher pmin : position du min}
    Pmin ← i;
    Pour j←(i+1) à N faire
        Si V[j] < V[pmin] alors
            pmin ← j;
        Fin-Si;
    Fin-Pour ;

    {Opération 2 : permuter entre les cases pmin et i}
    Si pmin <> i alors
        Z ← V[pmin];
        V[pmin] ← V[i];
        V[i] ← V[Z];
    Fin-Si;
Fin-Pour;
```

## Algorithme

```
Algorithme TP1_Supp_Exo3_Par_Selection;
  Variables
    V : Tableau[1..100] de réel;
    N, i, j, pmin : entier;  Z : réel;
Début
  { -*-*-*- Entrées -*-*-*- }
  Lire(N) ;
  Pour i←1 à N faire
    Lire(V[i]);
  Fin-Pour;

  {-*-*-*- Traitement -*-*-*- }
  Pour i← 1 à (N-1) faire
      pmin ← i;
      Pour j←(i+1) à N faire
          Si V[j]<V[pmin] alors
              pmin ← j;
          Fin-si;
      Fin-Pour;

      Si pmin<>i alors
          Z ← V[pmin] ;
          V[pmin] ← V[i] ;
          V[i]←Z;
      Fin-Si;
  Fin-Pour;

  {-*-*-*- Sortie -*-*-*- }
  Pour i←1 à N faire
    Écrire(V[i]);
  Fin-Pour;
Fin.
```

## Programme PASCAL

```
Program TP1_Supp_Exo3_Par_Selection;
  Var
      V: Array[1..100] of real;
      N, i, j, pmin : integer;    Z : real ;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur V : ') ;
    Read(N) ;
    Write('Donner les valeurs du vecteur V  : ');
    For i:=1 to N do
      Read(V[i]);

    {-*-*-*- Traitement -*-*-*- }
    For i:=1 to (N-1) do {Pour chaque étape i allant de 1 à N-1 }
    begin
      {Opération 1 : rechercher pmin : position du minimum }
      pmin ← i;
      For j←(i+1) to N do
          If V[j] < V[pmin] then
              Pmin ← j;
      {Opération 2 : permutation si elle existe }
      if pmin<>i then
      begin
          Z := V[pmin];
          V[pmin] := V[i] ;
          V[i] := Z;
      endi;
    end;

    {-*-*-*- Sortie -*-*-*- }
    Writeln('Le vecteur V Trié avec ordre croissant  : ');
    For i:=1 to N do
      Write(V[i]:10:2);
End.
```

L'exécution du programme sur le lien : https://onlinegdb.com/CXBIANob2 donne le résultat suivant :

```
Donner la taille du vecteur V : 7
Donner les valeurs du vecteur V : 15 -3 6 55 32 -8 -7.5
-----------------------------------
L'affichage du vecteur V après le tri (ordre croissant) :
      -8.00      -7.50      -3.00       6.00      15.00      32.00      55.00
```

Un autre programme PASCAL, sur le lien : https://onlinegdb.com/uL38o_zcD, qui permet de réaliser le tri par sélection en affichant les détailles : la position du minimum *pmin*, les permutations réalisée et les valeurs du vecteur V (après chaque étape) :

```
Donner la taille du vecteur V : 7
Donner les valeurs du vecteur V : 15 -3 6 55 32 -8 -7.5

--------------------
L'étape i=1 :
   01 - La position du min  entre 1 et 7 : pmin = 6 ==>  minimum=-8.00
   02 - Permutation entre pmin=6 et i=1 :
   La vecteur V =     -8.00    -3.00     6.00    55.00    32.00    15.00    -7.50
--------------------
L'étape i=2 :
   01 - La position du min  entre 2 et 7 : pmin = 7 ==>  minimum=-7.50
   02 - Permutation entre pmin=7 et i=2 :
   La vecteur V =     -8.00    -7.50     6.00    55.00    32.00    15.00    -3.00
--------------------
L'étape i=3 :
   01 - La position du min  entre 3 et 7 : pmin = 7 ==>  minimum=-3.00
   02 - Permutation entre pmin=7 et i=3 :
   La vecteur V =     -8.00    -7.50    -3.00    55.00    32.00    15.00     6.00
--------------------
L'étape i=4 :
   01 - La position du min  entre 4 et 7 : pmin = 7 ==>  minimum=6.00
   02 - Permutation entre pmin=7 et i=4 :
   La vecteur V =     -8.00    -7.50    -3.00     6.00    32.00    15.00    55.00
--------------------
L'étape i=5 :
   01 - La position du min  entre 5 et 7 : pmin = 6 ==>  minimum=15.00
   02 - Permutation entre pmin=6 et i=5 :
   La vecteur V =     -8.00    -7.50    -3.00     6.00    15.00    32.00    55.00
--------------------
L'étape i=6 :
   01 - La position du min  entre 6 et 7 : pmin = 6 ==>  minimum=32.00
   02 - Pas de Permutation pmin=6 et i=6 :
   La vecteur V =     -8.00    -7.50    -3.00     6.00    15.00    32.00    55.00
-----------------------------------
L'affichage du vecteur V après le tri (ordre croissant) :
      -8.00      -7.50      -3.00       6.00      15.00      32.00      55.00
```

Espérant que le tri par sélection est bien compris, et nous passons au tri *par bulles*.

### b- Méthode 02 : Tri par bulles (Tri par propagation)

Soit V un vecteur de Taille N. Le tri (ordre croissant) du vecteur V en utilisant la méthode par bulles (par propagation) consiste à réaliser des comparaisons successives (et éventuellement échange (permutation)) entre deux éléments adjacents (voisins *i* et *i+1*) :

- *Étape 1* : comparaison entre i et (i+1) / i allant de 1 à N-1
  Si V[i] > V[i+1] alors échanger (permuter) entre V[i] et V[i+1]
  Après l'étape 1 : le maximum sera mis dans V[N]
- *Étape 2* : comparaison entre i et (i+1) / i allant de 1 à N-2
  Si V[i] > V[i+1] alors échanger (permuter) entre V[i] et V[i+1]
  Après l'étape 2 : le deuxième maximum sera mis dans V[N-1]
- *Étape 3* : comparaison entre i et (i+1) / i allant de 1 à N-3
  Si V[i] > V[i+1] alors échanger (permuter) entre V[i] et V[i+1]
  Après l'étape 2 : le troisième maximum sera mis dans V[N-2]

.....

.....

- *Étape N-1* : comparaison entre i et (i+1) / i allant de 1 à 1
  Si V[i] > V[i+1] alors échanger (permuter) entre V[i] et V[i+1]
  Après l'étape 2 : le $(n-1)^{ième}$ maximum sera mis dans V[2]

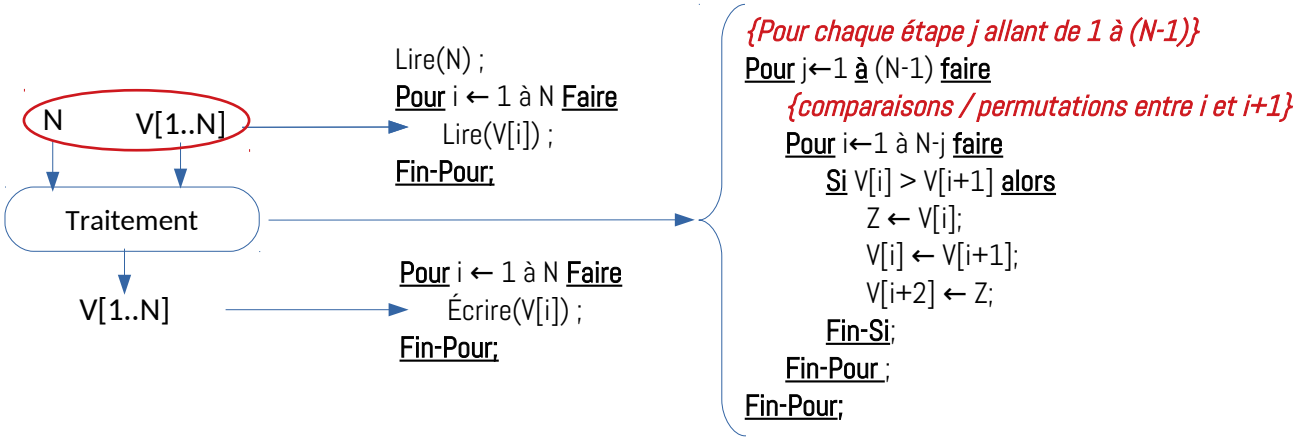
Et de cette façon, nous allons avoir le tableau trié, avec ordre croissant. Et si vous remarquez, à chaque étape i, le *ième maximum* est mis à la dernière case V[N-i+1] : on place toute d'abord les maximums successives à la dernière case de l'étape i.

Pour généraliser les étapes ci-dessus, on écrit en fonction de j / j=1 à (N-1) (N-1 étapes) :
- *Étape j* : comparaison entre i et (i+1) / i allant de 1 à N-j
  Si V[i] > V[i+1] alors échanger (permuter) entre V[i] et V[i+1]
  Après l'étape 2 : le maximum sera mis dans V[N-j+1]

Nous pouvons écrire le fragment d'algorithme suivant :

```
Pour j ← 1 à (N-1) faire { Pour chaque j allant de 1 à (N-1) : Il y a (N-1) étapes }
  Pour i←1 à (N-j) faire
    Si V[i] > V[i+1] alors { Comparaison successives entre i et i+1 : i=1,(N-j)}
      Z ← V[i];
      V[i] ← V[i+1] ;        { Permutation entre les cases i et i+1 si V[i] > V[i+1] }
      V[i+1] ← Z ;
    fin-si;
  Fin-Pour ;
Fin-Pour;
```

En regroupant tous ce qui a été ci-dessus dressé, nous obtenu les différentes parties de l'algorithme sur le schéma suivant :




Enfin, voici l'algorithme et le programme Pascal correspondant (voir le lien : https://onlinegdb.com/ASVWSw_r ) :

## Algorithme

```
Algorithme TP1_Supp_Exo3_Par_Bulles;
   Variables
      V : Tableau[1..100] de réel;
      N, i, j : entier;  Z : réel;
Début
    { -*-*-*- Entrées -*-*-*- }
    Lire(N) ;
    Pour i←1 à N faire
      Lire(V[i]);
    Fin-Pour;

    {-*-*-*- Traitement -*-*-*- }
    Pour j← 1 à (N-1) faire
        Pour i←1 à N-j faire
            Si V[i]>V[i+1] alors
                Z ← V[i] ;
                V[i] ← V[i+1] ;
                V[i+1] ← Z ;
            Fin-si;
        Fin-Pour;
    Fin-Pour;

    {-*-*-*- Sortie -*-*-*- }
    Pour i←1 à N faire
      Écrire(V[i]);
    Fin-Pour;
Fin.
```

## Programme PASCAL

```
Program TP1_Supp_Exo3_Par_Bulles;
  Var
       V: Array[1..100] of real;
       N, i, j : integer;    Z : real ;
Begin
     { -*-*-*- Entrées -*-*-*- }
     Write('Donner la taille du vecteur V : ') ;
     Read(N) ;
     Write('Donner les valeurs du vecteur V  : ');
     For i:=1 to N do
       Read(V[i]);

     {-*-*-*- Traitement -*-*-*- }
     For j:=1 to (N-1) do {Pour chaque étape j allant de 1 à N-1 }
        For i←1 to (N-j) do
            if V[i] > V[i+1] then
            begin
                Z := V[i];
                V[i] := V[i+1] ;
                V[i+1] := Z;
            endi;

     {-*-*-*- Sortie -*-*-*- }
     Writeln('Le vecteur V Trié avec ordre croissant  : ');
     For i:=1 to N do
       Write(V[i]:10:2);
End.
```

L'exécution du programme en ligne (https://onlinegdb.com/ASVWSw_r) donne le résultat :

```
Donner la taille du vecteur V : 7
Donner les valeurs du vecteur V : 15 -3 6 55 32 -8 -7.5
------------------------------------
L'affichage du vecteur V après le tri (ordre croissant) :
      -8.00      -7.50      -3.00       6.00      15.00      32.00      55.00
```

De le même façon que le tri par sélection, un autre programme PASCAL, sur le lien : https://onlinegdb.com/ja2ExrLeqe, qui permet de réaliser le tri par bulles en affichant les détaille (les valeur du vecteur V après chaque étape) :

```
Donner la taille du vecteur V : 7
Donner les valeurs du vecteur V : 15 -3 6 55 32 -8 -7.5

--------------------
Étape j=1 - État initial du vecteur V est :
   15.00    -3.00     6.00    55.00    32.00    -8.00    -7.50
       Permutation entre i=1 et i+1=2
       Permutation entre i=2 et i+1=3
       Permutation entre i=4 et i+1=5
       Permutation entre i=5 et i+1=6
       Permutation entre i=6 et i+1=7
L'état finial, à la fin de l'étape j=1, du vecteur V est :
   -3.00     6.00    15.00    32.00    -8.00    -7.50    55.00

--------------------
Étape j=2 - État initial du vecteur V est :
   -3.00     6.00    15.00    32.00    -8.00    -7.50    55.00
       Permutation entre i=4 et i+1=5
       Permutation entre i=5 et i+1=6
L'état finial, à la fin de l'étape j=2, du vecteur V est :
   -3.00     6.00    15.00    -8.00    -7.50    32.00    55.00
--------------------
Étape j=3 - État initial du vecteur V est :
   -3.00     6.00    15.00    -8.00    -7.50    32.00    55.00
       Permutation entre i=3 et i+1=4
       Permutation entre i=4 et i+1=5
L'état finial, à la fin de l'étape j=3, du vecteur V est :
   -3.00     6.00    -8.00    -7.50    15.00    32.00    55.00

--------------------
Étape j=4 - État initial du vecteur V est :
   -3.00     6.00    -8.00    -7.50    15.00    32.00    55.00
       Permutation entre i=2 et i+1=3
       Permutation entre i=3 et i+1=4
L'état finial, à la fin de l'étape j=4, du vecteur V est :
   -3.00    -8.00    -7.50     6.00    15.00    32.00    55.00

--------------------
```

```
Étape j=5 - État initial du vecteur V est :
   -3.00    -8.00    -7.50     6.00    15.00    32.00    55.00
        Permutation entre i=1 et i+1=2
        Permutation entre i=2 et i+1=3
L'état finial, à la fin de l'étape j=5, du vecteur V est :
   -8.00    -7.50    -3.00     6.00    15.00    32.00    55.00

--------------------
Étape j=6 - État initial du vecteur V est :
   -8.00    -7.50    -3.00     6.00    15.00    32.00    55.00
L'état finial, à la fin de l'étape j=6, du vecteur V est :
   -8.00    -7.50    -3.00     6.00    15.00    32.00    55.00
-------------------------------------
L'affichage du vecteur V après le tri (ordre croissant) :
      -8.00      -7.50      -3.00       6.00      15.00      32.00      55.00
```

Il y a d'autres algorithme de tri comme par exemple :

- Le tri par insertion
- Le tri rapide ou Quicksort
- Le tri par fusion
- ....

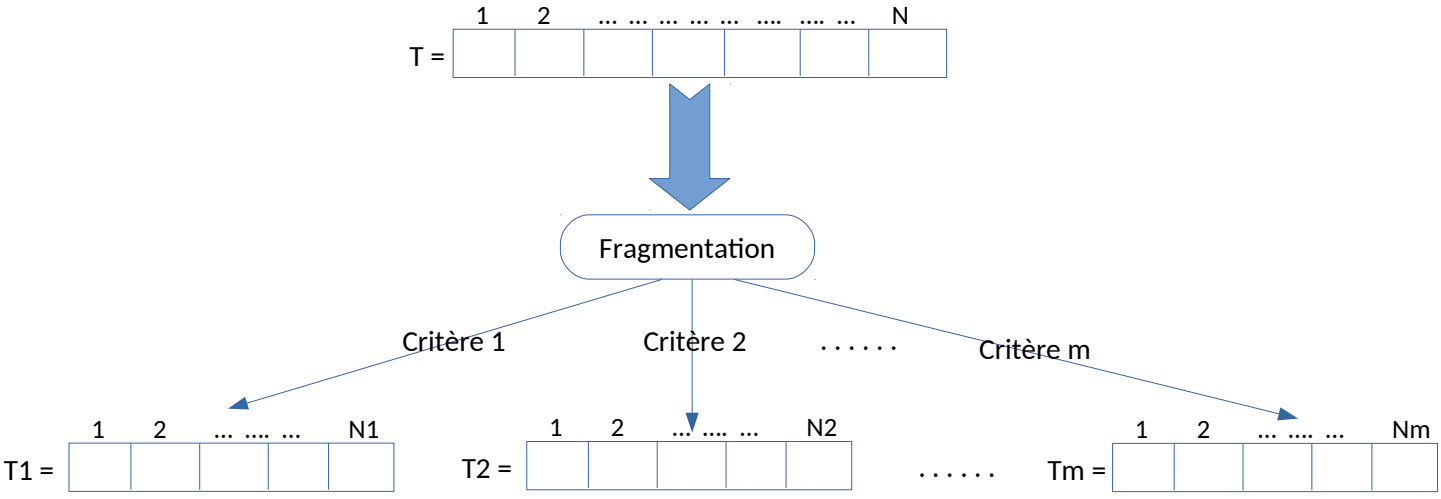
Pour plus d'information, voir les liens suivants :

- https://fr.wikipedia.org/wiki/Cat%C3%A9gorie:Algorithme_de_tri
- https://interstices.info/les-algorithmes-de-tri
- https://en.wikipedia.org/wiki/Sorting_algorithm

Les algorithmes de tri sont très importants dans l'algorithmique, et ils préparent une liste de données pour d'autre algorithmes, par exemple, la recherche par dichotomie nécessite que le vecteur soit déjà trié.

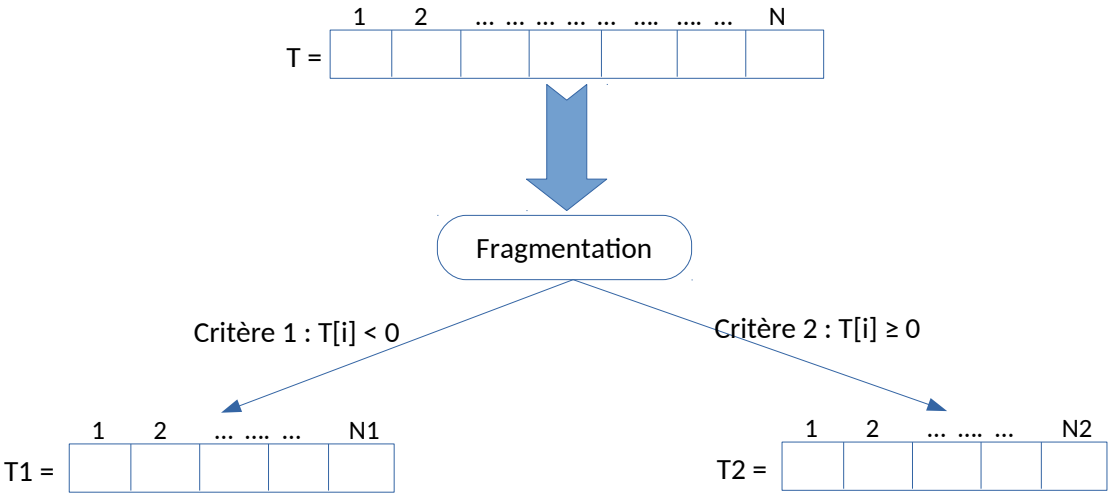
## EXERCICE SUP. N°04 : FRAGMENTER UN VECTEUR

Dans cet exercice, nous allons voir le problème de fragmentation un vecteur sur plusieurs vecteurs selon des critères donnés. D'une façon générale, ce problème peut être schématisé comme suit :



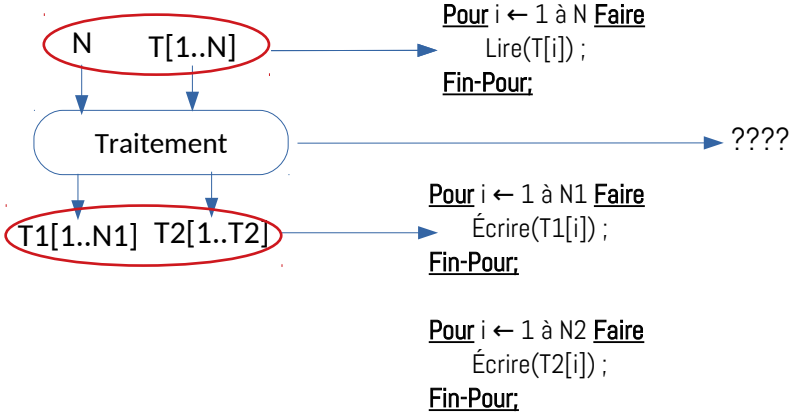

Les critères sont des expression booléennes qui s'applique sur les composantes du vecteur T (c'est à dire : T[i] / $i=\overline{1,N}$). Aussi, d'une façon générale, les critères sont disjoints, c'est à dire si une composante T[i] (tel-que i=1,N) vérifier un critère, donc T[i] ne vérifie pas les autres, dans ce cas : N1 + N2 + … + Nm = N (Somme des tailles des vecteurs fragments sera égale à la taille du vecteur initial T).

Dans l'exercice supplémentaire N°04 (l'exercice en cours ), il y a deux critères : T[i] < 0 (composantes strictement négatives) et T[i] ≥ 0 (composantes positives), selon le schéma suivant :




La première étape consiste à schématiser le problème, sous forme de variables d'entrée/sortie, par la suite, comme seconde étape, de trouver la partie traitement (les instructions qui transforment les variables d'entrée vers les variable de sortie).

Les variables d'entrées sont N, et T[1..N], et les variables de sorties sont T1[1..N1] et T2[1..N2] commet illustré par le schéma suivant :




L'idée de traitement est la suivante :

1- Initialiser N1 et N2 à 0

2- Parcourir les éléments du vecteur T : T[i] pour i allant de 1 à N

3- Pour chaque T[i], comparer sa valeur à zéro

3.1- Si T[i] < 0 alors : N1 ← N1+1 et T1[N1] ← T[i]

3.2- Si T[i] >= 0 alors : N2 ← N2+1 et T2[N2] ← T[i]

En regroupant toutes les briques de ce qui a été ci-dessus dressé, nous aurons l'algorithme et le programme PASCAL suivants (Voir le lien https://onlinegdb.com/DjJ4ZbXSL) :

## Algorithme

```
Algorithme TP1_Supp_Exo4_Fragmentation;
  Variables
    T, T1, T2 : Tableau[1..100] de réel;
    N, i, N1, N2 : entier;
Début
  { -*-*-*- Entrées -*-*-*- }
  Lire(N) ;
  Pour i←1 à N faire
    Lire(T[i]);
  Fin-Pour;

  {-*-*-*- Traitement -*-*-*- }
  N1 ← 0;
  N2 ← 0;
  Pour i← 1 à N faire
     Si T[i] < 0  alors
        N1 ← N1 + 1 ;
        T1[N1] ← T[i] ;
     Sinon
        N2 ← N2 + 1 ;
        T2[N2] ← T[i] ;
     Fin-Si;
  Fin-Pour;

  {-*-*-*- Sortie -*-*-*- }
  Pour i←1 à N1 faire
    Écrire(T1[i]);
  Fin-Pour;
  Pour i←1 à N2 faire
    Écrire(T2[i]);
  Fin-Pour;
Fin.
```

## Programme PASCAL

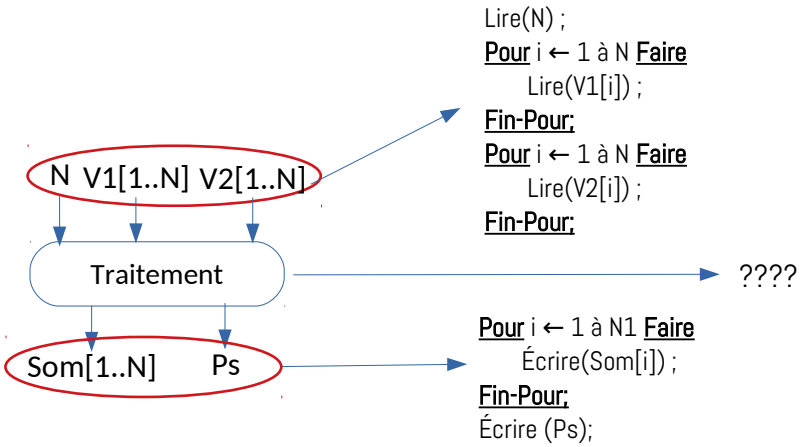
```
Program TP1_Supp_Exo4_Fragmentation;
  Var
      T, T1, T2 : Array[1..100] of real;   N, i, N1, N2 : integer;
Begin
    { -*-*-*- Entrées -*-*-*- }
    Write('Donner la taille du vecteur T : ') ;
    Read(N) ;
    Write('Donner les valeurs du vecteur T  : ');
    For i:=1 to N do
      Read(T[i]);

    {-*-*-*- Traitement -*-*-*- }
    N1 := 0 ;
    N2 := 0 ;
    For i:=1 to N do {Pour chaque élément du vecteur T }
      if  T[i]<0 then
      begin
          N1 := N1+1 ;          T1[ N1] := T[i] ;
      end
      Else
      begin
          N2 := N2+1 ;          T2[ N2] := T[i] ;
      End ;

    {-*-*-*- Sortie -*-*-*- }
    Writeln('Le vecteur T1  : ');
    For i:=1 to N1 do
      Write(T1[i]:10:2);

    Writeln('Le vecteur T2  : ');
    For i:=1 to N2 do
      Write(T2[i]:10:2);
End.
```

## EXERCICE SUP. N°05 : SOMME ET PRODUIT SCALAIRE DE DEUX VECTEURS

La somme de deux vecteurs donne un vecteur comme résultat, par contre, le produit scalaire de deux vecteurs donne une valeur scalaire (valeur numérique).

Nous suivons la même méthode de résolution de problème, tout d'abord, nous commençons par le schéma d'entrée/sortie par la suite nous passons au traitement. Et en fin, nous écrivons l'algorithme et le programme Pascal correspondant.

Comme variable d'entrée, nous avons N, V1[1..N1] et V2[1..N1] et comme variables de sortie nous avons : Som[1..N] et Ps




**Traitement :**

*{Pour chaque i=1,N : Som[i] = V1[i] + V2[i]}*
**Pour** i←1 **à** N **faire**
Som[i] ← V1[i] + V2[i];
**Fin-Pour**;

*{Ps = V1[1]*V2[1] + … + V1[N]*V2[N]}*
*{Ps = Somme des V1[i]*V2[i] pour i=1,N}*
Ps ← 1 ;
**Pour** i←1 **à** N **faire**
Ps ← Ps * V1[i]*V2[i];
**Fin-Pour**;

Voir le lien (pour l'exécution en ligne du programme) : https://onlinegdb.com/Szrvw8JbI

### Algorithme

```
Algorithme TP1_Supp_Exo5_Somme_Produit_Scalaire;
  Variables
    V1, V2, Som : Tableau[1..100] de réel;
    N, i  : entier; Ps : réel;
Début
   { -*-*-*- Entrées -*-*-*- }
   Lire(N) ;
   Pour i←1 à N faire
     Lire(V1[i]);
   Fin-Pour;
   Pour i←1 à N faire
     Lire(V2[i]);
   Fin-Pour;

   {-*-*-*- Traitement -*-*-*- }
   Pour i← 1 à N faire
       Som[i] ← V1[i] + V2[i];
   Fin-Pour;

   Ps ← 0 ;
   Pour i← 1 à N faire
      Ps ← Ps + V1[i]*V2[i] ;
   Fin-Pour;

   {-*-*-*- Sortie -*-*-*- }
   Pour i←1 à N1 faire
     Écrire(Som[i]);
   Fin-Pour;

   Écrire (Ps);
Fin.
```

### Programme PASCAL

```
Program TP1_Supp_Exo5_Somme_Produit_Scalaire;
  Var
       V1, V2, Som : Array[1..100] of real;
       N, i : integer; Ps : real;
Begin
     { -*-*-*- Entrées -*-*-*- }
     Write('Donner la taille des vecteurs V1 et V2 : ') ;
     Read(N) ;
     Write('Donner les valeurs du vecteur V1 : ');
     For i:=1 to N do
       Read(V1[i]);

     Write('Donner les valeurs du vecteur V2 : ');
     For i:=1 to N do
       Read(V2[i]);

     {-*-*-*- Traitement -*-*-*- }
     For i:=1 to N do {Pour chaque élément des vecteurs V1 et V2 }
       Som[i] := V1[i] + V2[i];

     Ps := 0;
     For i:=1 to N do {Pour chaque élément des vecteurs V1 et V2 }
       Ps :=  Ps + V1[i] * V2[i];

     {-*-*-*- Sortie -*-*-*- }
     Writeln('Le vecteur Somme = V1 + V2  : ');
     For i:=1 to N1 do
       Write(Som[i]:10:2);

     Writeln;
     Write('Le produit scalaire  : ', Ps:0:2);
End.
```

## EXERCICE SUP. N°06 : SOMME, PRODUIT ET COMPTEUR D'ÉLÉMENTS D'UN VECTEUR

Dans ce sixième exercice de la série supplémentaire sur les vecteurs (tableaux à une dimension), nous allons voir quelques opérations sur les composantes (les éléments / les valeurs des cases) d'un vecteur quelconque. Comme rappel, voici l'énoncé de cet exercice :

*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 produits des éléments divisible par 4 et non divisible par 3.*
*– Compter le nombre d'éléments non-divisibles par 3 et non-divisibles par 4.*

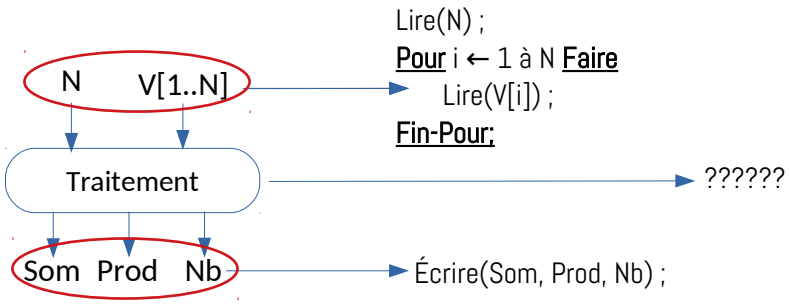
C'est clair que :

*Les entrées :* N (taille du vecteur V) et les cases V[i] / i=1, ..., N

*Les sorties :*

- Som : Sommes des éléments divisibles par 3 et non divisibles par 4
- Prod : Prod des éléments divisibles par 4 et non divisibles par 3
- Nb : Nombre d'éléments non-divisibles par 3 et aussi non-divisibles par 4

Ceci est résumé sur le schéma suivant :




Il y a trois résultats à calculer, donc la partie traitement sera divisée en trois sous-traitements :

**Sous-Traitement 1**

```
Somme des éléments divisibles par 3 et non divisible par 4
  Som ← 0; {Toute somme est initialisée à ZÉRO}
  Pour i←1 à N faire {Pour chaque indice i allant de 1 à N}
    Si (V[i] mod 3 = 0) ET (V[i] mod 4 <> 0) alors
       Som ← Som + V[i] ; {V[i] vérifie la condition ci-dessus}
    Fin-Si;
  Fin-Pour;
```

**Sous-Traitement 2**

```
Produit des éléments divisibles par 4 et non divisible par 3
  Prod ← 1; {Tout produit est initialisé à UN}
  Pour i←1 à N faire {Pour chaque indice i allant de 1 à N}
    Si (V[i] mod 4 = 0) ET (V[i] mod 3 <> 0) alors
       Prod ← Prod * V[i] ; {V[i] vérifie la condition ci-dessus}
    Fin-Si;
  Fin-Pour;
```

**Sous-Traitement 3**

```
Nombre d'éléments non-divisibles par 3 et non divisible par 4
  Nb ← 0; {Tout compteur est initialisé à UN}
  Pour i←1 à N faire {Pour chaque indice i allant de 1 à N}
    Si (V[i] mod 3 <> 0) ET (V[i] mod 4 <> 0) alors
       Nb ← Nb * 1 ; {compteur V[i] vérifie la condition ci-dessus}
    Fin-Si;
  Fin-Pour;
```

Une fois toutes les *briques* (*composants algorithmiques : entrées / sorties / traitements / sous-traitements*) sont prêtes, nous pouvons construire la solution algorithmique ci-dessous avec le programme PASCAL (qui est juste une traduction de l'algorithme, autrement dit , et dans ce contexte : l'algorithme est plus important que le programme lui même et aussi l'analyse et la préparation des briques aussi plus important que l'algorithme lui même (puisque c'est la base de la solution algorithmique)) :

Pour le programme PASCAL, nous devons utiliser la fonction round pour convertir les réels aux entier. Vous pouvez l'exécuter en ligne à travers ce lien : https://onlinegdb.com/K6vAFvH7c.

## Algorithme

```
Algorithme TP1_Supp_Exo6_Somme_Produit_Compteur;
  Variables
    V : Tableau[1..100] de réel;
    N, i : entier;    Nb : entier; { — Compteur des V[i]— }
    Som, Prod : réel; { — Même type que le vecteur V — }
Début
  { -*-*-*- Entrées -*-*-*- }
  Lire(N) ;
  Pour i←1 à N faire
    Lire(T[i]);
  Fin-Pour;

  {-*-*-*- Traitement -*-*-*- }
  {Sous-Traitemt 1 : Somme des V[i] divisibles par 3 et non par 4}
   Som ← 0;
   Pour i← 1 à N faire
     Si (V[i] mod 3=0) ET (V[i] mod 4<>0)  alors
        Som ← Som + V[i] ;
     Fin-Si;
   Fin-Pour;

  {Sous-Traitemt 2 : Produit des V[i] divisibles par 4 et non par 3}
   Prod ← 1;
   Pour i← 1 à N faire
     Si (V[i] mod 4=0) ET (V[i] mod 3<>0)  alors
        Prod ← Prod * V[i] ;
     Fin-Si;
   Fin-Pour;

  {Sous-Traitemt 3 : Nombre des V[i] non divisibles par 3 et 4}
   Nb ← 1;
   Pour i← 1 à N faire
     Si (V[i] mod 3<>0) ET (V[i] mod 4<>0)  alors
        Nb ← Nb + 1 ;
     Fin-Si;
   Fin-Pour;


  {-*-*-*- Sortie -*-*-*- }
  Écrire(Som, Prod, Nb);
Fin.
```

## Programme PASCAL

```
Program TP1_Supp_Exo6_Somme_Produit_Compteur ;
  Var
      V : Array[1..100] of real;
      N, i : integer; Nb : integer ; { — Compteur — }
      Som, Prod : real; { — Même type que le vecteur V — }
Begin
  { -*-*-*- Entrées -*-*-*- }
  Write('Donner la taille du vecteur V : ') ;
  Read(N) ;
  Write('Donner les valeurs du vecteur V  : ');
  For i:=1 to N do
    Read(V[i]);

  {-*-*-*- Traitement -*-*-*- }
  {Sous-Traitement 1 : Somme des V[i] divisibles par 3 et non par 4}
  Som := 0 ;
  For i:=1 to N do {Pour chaque élément du vecteur T }
    if  (round(T[i]) mod 3=0) AND (round(V[i]) mod 4 <> 0) then
        Som := Som + V[i];

  {Sous-Traitement 2 : Produit des V[i] divisibles par 4 et non par 3}
  Prod := 1 ;
  For i:=1 to N do {Pour chaque élément du vecteur T }
    if  (round(T[i]) mod 4=0) AND (round(V[i]) mod 3 <> 0) then
        Prod := Prod * V[i];

  {Sous-Traitement 3 : Nombre des V[i] non-divisibles par 3 et 4}
  Nb := 0 ;
  For i:=1 to N do {Pour chaque élément du vecteur T }
    if  (round(T[i]) mod 3 <> 0) AND (round(V[i]) mod 4 <> 0) then
        Nb := Nb + 1;

  {-*-*-*- Sorties -*-*-*- }
  Writeln('La somme des cases divisibles par 3 et non par 4 : ', Som:0:1);
  Writeln('Le prodtuis des cases divisibles par 4 et non par 3 : ',Prod:0:1);
  Writeln('Le prodtuis des cases divisibles par 4 et non par 3 : ',Prod:0:1);
End.
```

**Remarque :**

Une bonne pratique dans l'algorithmique est de séparer les différents calculs dans des sous-traitement (ceci prépare la modularité que nous allons voir en deuxième chapitre : Sous-programmes - procédures et fonctions).

## EXERCICE SUP. N°07 : CONVERTIR UN NOMBRE DE BASE 10 À BASE 2

Nous avons vu, en chapitre I du premier semestre (module Informatique 1) comment convertir un nombre écrit en base 10 (décimal) vers la base b (b un nombre entier supérieur ou égale à 2), et nous avons vu, en particulier, cette conversion entre le système décimal et le système binaire (base 2).

Comme rappel, voici une conversion réalisée dans un exercice de la série de TP N°2 du semestre 1 :

Réaliser les conversions suivantes :

**$2021 = (?)_2$**

Nous réalisons les divisons euclidiennes successives, comme suit :

| | |
|---|---|
| 2021 | 2 |
| 1 | 1010 \| 2 |
| 0 | 505 \| 2 |
| 1 | 252 \| 2 |
| 0 | 126 \| 2 |
| 0 | 63 \| 2 |
| 1 | 31 \| 2 |
| 1 | 15 \| 2 |
| 1 | 7 \| 2 |
| 1 | 3 \| 2 |
| 1 | 1 \| 2 |
| 1 | 0 |

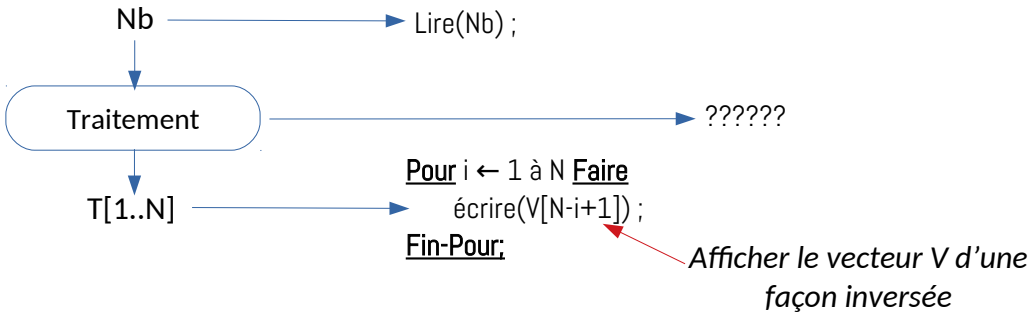
$$2021 = (11111100101)_2$$

Dans ce dernier exercice (de la série supplémentaire), nous devons enregistrer les chiffres binaires (les restes des divisions) dans un vecteur T, et pour l'exemple ci dessous, nous aurons **T = [ 1 0 1 0 0 1 1 1 1 1 1 ]** *(Le premier reste est à le première case)*. Cependant, au niveau de l'affichage, il faut commencer du dernier élément vers le premier : T[N], T[N-1], ..., T[1]

Pour résoudre ce problème, nous devons répondre à deyx ces questions :

- *Comment faire la conversion de la base 10 vers la base 2 ---→ Comment obtenir les restes des divisions ?*
- *Comment mettre les restes dans un vecteur T ?*

Avant de répondre à ces deux questions, voici le schéma d'entrée/sortie de l'algorithme à réaliser :



Vous constatez au niveau de l'affichage : écrire(V[N-i+1]) / i=1...N, qu'on commence d'afficher à partir du dernier élément V[N], puis V[N-1], ...., jusqu'à V[1], puisque le premier reste est enregistré à la premier case (*Indice = 1*), et c'est le chiffre du poids faible (le plus à gauche) et le dernier reste enregistré à la dernière case (Indice = N) est chiffre du poids fort (le plus à gauche).

Pour obtenir tous les restes de divisions, on réalise les étapes (formules mathématiques) suivantes :

$r_0 = N \bmod 2$ ; $q_0 = N \operatorname{div} 2$ → $r_1 = q_0 \bmod 2$ ; $q_1 = q_0 \operatorname{div} 2$ → $r_2 = q_1 \bmod 2$ ; $q_2 = q_1 \operatorname{div} 2$ → ... → $r_{M-1} = q_{M-2} \bmod 2$ ; $q_{M-1} = q_{M-2} \operatorname{div} 2$ → $r_M = q_{M-1} \bmod 2$ ; $q_M = q_{M-1} \operatorname{div} 2 = 0$

À chaque étape *i* (i=0 ... M), nous préparons le quotient $q_i$ pour qu'il soit utilisé à l'étape suviante (i+1), et s'arrête à l'étape où le quotient est nul ($q_M = 0$).

Si on veut convertir les étapes précédente, en pseudo-algorithmique, ça sera comme suit:

r0 ← N mod 2 ; q0 ← N div 2 ; T[1] ← r0 → r1 ← q0 mod 2 ; q1 ← q0 div 2 ; T[2] ← r1 → r2 ← q1 mod 2 ; q2 ← q1 div 2 ; T[3] ← r2 → ... → rM_1 ← qM_2 mod 2 ; qM_1 ← qM_2 div 2 ; T[M_1] ← rM_1 → rM ← qM_1 mod 2 ; qM ← qM_1 div 2 ; T[M] ← rM

Nous pouvons réduire le nombre d'instruction dans chaque étape, comme suit :

T[1] ← N mod 2 ; q0 ← N div 2 → T[2] ← q0 mod 2 ; q1 ← q0 div 2 → T[3] ← q1 mod 2 ; q2 ← q1 div 2 → ... → T[M_1] ← qM_2 mod 2 ; qM_1 ← qM_2 div 2 → T[M] ← qM_1 mod 2 ; qM ← qM_1 div 2

Algorithmiquement parlant, les étapes ci-dessus ne sont pas correctes (complètement), il faut remplacer les points de suspension (. . .) par une boucle dont chaque itération représente une étape pour obtenir le reste de la division et préparer le quotient suivant, l'idée est :

- (1) initialiser à q à Nb et N à 0 : q←Nb ; N←0 (N est la taille du vecteur T)
- (2) Incrémenter N : N ← N+1
- (3) mettre le reste de division de q par 2 dans la case N de T : T[N]←q **mod** 2;
- (4) préparer le prochain quotient : q ← q div 2;
- (5) si q = 0 alors Fin
- (6) sinon aller à (2)

En traduisant le pseudo-algorithme précédent, nous obtenons :

```
Traitement :

  q ← Nb ;
  N ← 0;

  Répéter
     N ← N+1;
     T[N] ← q mod 2;
     q ← q div 2;
  Jusqu'à q = 0;
```

En regroupant le schéma des entrées / sorties ainsi que la partie traitement (les briques algorithmiques) nous obtenons l'algorithme suivant ainsi que sa traduction en programme PASCAL :

**Algorithme**

```
Algorithme TP1_Supp_Exo7_Conversion_base_2_base_10;
  Variables
     T : Tableau[1..100] de entier;
     N, i : entier;     Nb : entier;
Début
   { -*-*-*- Entrées -*-*-*- }
   Lire(Nb) ;

   {-*-*-*- Traitement -*-*-*- }
   q ← Nb;
   N ← 0 ;

   Répéter
      N ← N+1;
      T[N] ← q mod 2;
      q ← q div 2;
   Jusqu'à q = 0;


   {-*-*-*- Sortie -*-*-*- }
   Pour i←1 à N faire
       Écrire(T[N-i+1]);
   Fin-Pour;
Fin.
```

**Programme PASCAL**

```
Program TP1_Supp_Exo7_convertir_base_2_base_10 ;
  Var
       T : Array[1..100] of real;
       N, i : integer; Nb : integer ;
Begin
   { -*-*-*- Entrées -*-*-*- }
   Write('Donner la valeur de Nb : ') ;
   Read(Nb) ;

   {-*-*-*- Traitement -*-*-*- }
   q := Nb;
   N := 0;

   Repeat
      N := N +1;
      T[N] := q mod 2;
      q := q div 2;
   Until q=0;

   {-*-*-*- Sorties -*-*-*- }
   Writeln('Le nombre Nb s''écrit en base 2 : ',);
   For i:=N downto 1 do
      write(T[i]) ;
End.
```

Le programme ci-dessous est disponible en ligne pour exécution : https://onlinegdb.com/cJjU-wVBG

J'espère bien que vous avez bien compris le principe, pour toute difficulté ou problème veuillez contacter l'un de vos enseignant, de cours ou de T.P., ou me contacter par mail : redouane.ouzeggane@univ-bejaia.dz ou bien via la plateforme e-learning.

# ANNEXE : Remarques Récapitulatives

## 1 – Autres façons de déclarer un vecteur

Dans toute déclaration d'un vecteur T de taille maximale 100 et d'une taille à utiliser N, on déclare le vecteur, l'entier N ainsi que la variable entière i qui joue le rôle d'indice pour accéder aux cases du vecteur (parcourir les cases du vecteur T) : T[i] / i = 1, ..., N.

**a–** Dans les exercices de cette série, pour déclarer un vecteur T (supposant que c'est un vecteur entier) nous avons utilisé cette syntaxe :

| Algorithme | Programme PASCAL |
|---|---|
| Variables<br>T : Tableau[1..100] de entier;<br>N, i : entier; | Var<br>T : Array[1..100] of integer ;<br>N, i : integer ; |

**b–** Une deuxième syntaxe (méthode), qui n'est pas différente de la précédente, consiste à utiliser une constante, par exemple MAX, pour la taille maximale du vecteur T :

| Algorithme | Programme PASCAL |
|---|---|
| Constante<br>MAX = 100;<br>Variables<br>T : Tableau[1..MAX] de entier;<br>N, i : entier; | Const<br>TMAX = 100;<br>Var<br>T : Array[1..MAX] of integer ;<br>N, i : integer ; |

***NB :***

*– Il n y aura aucun changement au niveau des instructions (lectures, traitement, écritures).*

**c–** Une troisième méthode est de définir un nouveau type, nommé, par exemple, T_VECTEUR :

| Algorithme | Programme PASCAL |
|---|---|
| Type<br>T_VECTEUR = Tableau[1..100] d'entier;<br>Variables<br>T : T_VECTEUR;<br>N, i : entier; | Type<br>T_VECTEUR = Array[1..100] of integer ;<br>Var<br>T : T_VECTEUR;<br>N, i : integer ; |

*Voir le lien pour voir un exemple de cette déclaration : https://onlinegdb.com/gcrzdIzKn*

**NB :**

*– Il n y aura aucun changement au niveau des instructions (lectures, traitement, écritures).*

*– Cette méthode est importante lorsque on veut faire des paramètres tableau pour des sous-programmes (Procédures et Fonctions). Voir le chapitre N°02 des sous-programmes.*

*– Le nom du nouveau type est arbitraire, chaque programmeur le nomme comme il le veut, et ici, c'est juste un exemple.*

*– Pour le nom T_VECTEUR, T_ signifie type (juste une convention d'écriture et ce n'est pas une règle à respecter). Comme le langage DELPHI ou LAZARUS (version PASCAL amélioré : voir les liens : https://www.embarcadero.com/fr/products/delphi et https://www.lazarus-ide.org/) les types commence par T (TForm, TButton, ...)*

**d–** Une autre méthode pour déclarer un vecteur est de regrouper le le cases du vecteur et sa taille utilisée N dans une même structure : Enregistrement (les étudiants ne sont pas concernés par les enregistrements), mais c'est bien d'avoir une idée :

**Algorithme**

```
Type
        T_VECTEUR = Enregistrement
            Valeurs Tableau[1..100] d'entier;
            N : entier;
        Fin;
Variables
      T : T_VECTEUR;
      i : entier;
```

**Programme PASCAL**

```
Type
        T_VECTEUR = Record
            Valeurs Array[1..100] of integer;
            N : integer;
        End;
Var
      T : T_VECTEUR;
      i : integer;
```

**NB :**

*– Le type complexe **enregistrement** permet de regrouper plusieurs types d'information dans un même type plus complexe. Le type T_VECTEUR ci-dessus est constitué de deux champs : Valeurs et N.*

*– Il y aura des modifications au niveau des instructions (lectures, traitement et écritures), voici un exemple :*

**Algorithme**

```
Algorithme exemple_Vecteur;
   Type
        T_VECTEUR = Enregistrement
            Valeurs Tableau[1..100] d'entier;
            N : entier;
        Fin;
   Variables
      V : T_VECTEUR;
      i : entier;
Début
    {Entrée : La lecture du vecteur V}
    Écrire('Donner la taille du vecteur V : ');
    Lire(V.N);
    Écrire('Donner les valeurs du vecteur V : ');
    Pour i←1 à V.N faire
        Lire (V.Valeurs[i]);
    Fin-Pour;

    {Sortie : L'affichage du vecteur V}
    Écrire('Donner les valeurs du vecteur V : ');
    Pour i←1 à V.N faire
        Lire (V.Valeurs[i]);
    Fin-Pour;
Fin.
```

*Voir le lien pour voir un exemple de cette déclaration : https://onlinegdb.com/17xsFZW_f*

## 2 – Démarche à suivre pour réaliser un algorithme

Vous avez remarqué, dans tous les exercices de ce document, nous avons suivé une démarche afin de réaliser un algorithme, et elle consiste en :

***– Établir le schéma des Entrées/Sorties***

Ici, nous devons déterminer les variables d'entrée et celles de sortie (et éventuellement les variables intermédiaire)

***– Établir la partie traitement***

Dans cette étape, nous devons trouvez les instructions qui permettent de transformer les variables d'entrée en variables de sortie. Souvent, nous prenons un exemple pour comprendre le problème et rendre les choses plus visible. La partie traitement est la partie la plus importante (et souvent la plus difficile) dans un algorithme (le noyau de l'algorithme).

***– Regrouper le schéma et la partie traitement***

Enfin, nous regroupons l'étape 1 (le schéma des entrée/sortie) et l'étape 2 (la partie traitement) afin d'écrire l'algorithme final.

***– Valider l'algorithme***

Cette étape, qui n'est pas toujours réalisé dans les exercices, consiste à dérouler l'algorithme manuellement et vérifier s'il donne des bons résultats suivant un jeu de tests (selon les valeurs des variables d'entrée).

***– Traduction en programme et exécution***

L'algorithme sera traduit en un programme (selon un langage de programmation : Pascal, C, Python, Java, …) et exécuté par la machine. On peut valider l'algorithme selon les résultats affichés par le programme.

## 3 – Schéma d'E/S et Traitement

Pour écrire un algorithme, nous aurons besoin du schéma d'E/S (entrée/sortie) et des instructions de la partie traitement : de cette façon, nous pouvons écrire tout l'algorithme (entête, déclaration et le corps) et il faut utiliser les données de l'exercice et le bon sens pour déduire toute information concernant les détails de l'algorithme.

Vu le nombre important de problèmes algorithmiques, c'est mieux d'établir un résumé de la solution de ces problèmes sous forme de schéma d'algorithme : Entrées (Variables d'entrées), Sorties (Variables de sortie) et la partie traitement, puis traduire ce schéma en algorithmique et, éventuellement, en langage PASCAL (ou d'autres langages : C, C++, Python, Java, Javascript, ….).

Sur le lien ci-dessous, un résumé de solution de quelques problèmes sur les vecteurs (avec la partie traitement) en algorithmique et en programme PASCAL :

https://drive.google.com/file/d/1wi3bbNcQP6fH2In-5VFOzjtr8sIHZldM/view?usp=sharing

Le même résumé pour les matrices (tableaux à deux dimensions) :

https://drive.google.com/file/d/1DhwnQMRMlSexQYRGR6nayDqvaJjKzLQu/view?usp=sharing

# Bon Courage

# &

# Travaillez bien.

---

Cours Elearning :

https://elearning.univ-bejaia.dz/course/view.php?id=7944

Page facebook :

https://www.facebook.com/InitiationAlgoProgrammation/

La chaîne Youtube :

https://www.youtube.com/@algo-prog

---

Adapté par: Redouane OUZEGGANE

rouzeggane@gmail.com - redouane.ouzeggane@univ-bejaia.dz