TP - Programmation

Corrigé de la série de TP N°1 – Tableaux à une dimension (Vecteurs)

Exercice N°01 : Algorithme → Programme C

Soit l'algorithme suivant :

```
Algorithme Vecteur;
Variables
    T : Tableau [0..99] d'entier ;
    N, i, S : entier;
Début
  // Entrées
  Ecrire('Donner la taille du vecteur T : ');
  Lire(N);
  Ecrire('Donner les composantes du vecteur T : ');
  Pour i←0 à N-1 faire
      Lire(T[i]);
  FinPour;
  // Traitements
  S ← 0;
  Pour i←0 à N-1 faire
    Si (T[i] mod 2 = 0) alors
        S ← S+T[i];
    FinSi;
  FinPour;
  // Sorties
  Ecrire('La somme S=', S);
Fin.
```

Questions :

1- Traduire l'algorithme en Programme C.

2- Compiler et exécuter le programme pour :
N = 4 et T=[14, 3, 8, 22].

3- Dérouler l'algorithme pour les valeurs de N et T ci-dessus ?

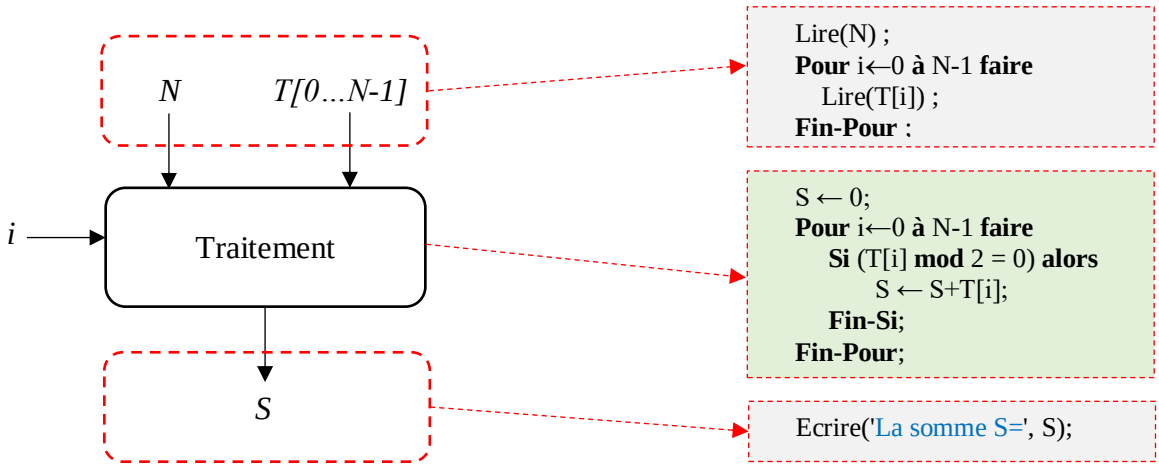
4- Déduire ce que fait l'algorithme ?

5- Ré-écrire le programme en remplaçant la boucle *Pour* par la boucle *Tant-que* **dans la partie traitements**.

6- Ré-écrire le programme en remplaçant la boucle *Pour* par la boucle *Répéter* **dans la partie traitements**.

Solution :

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



Remarque :

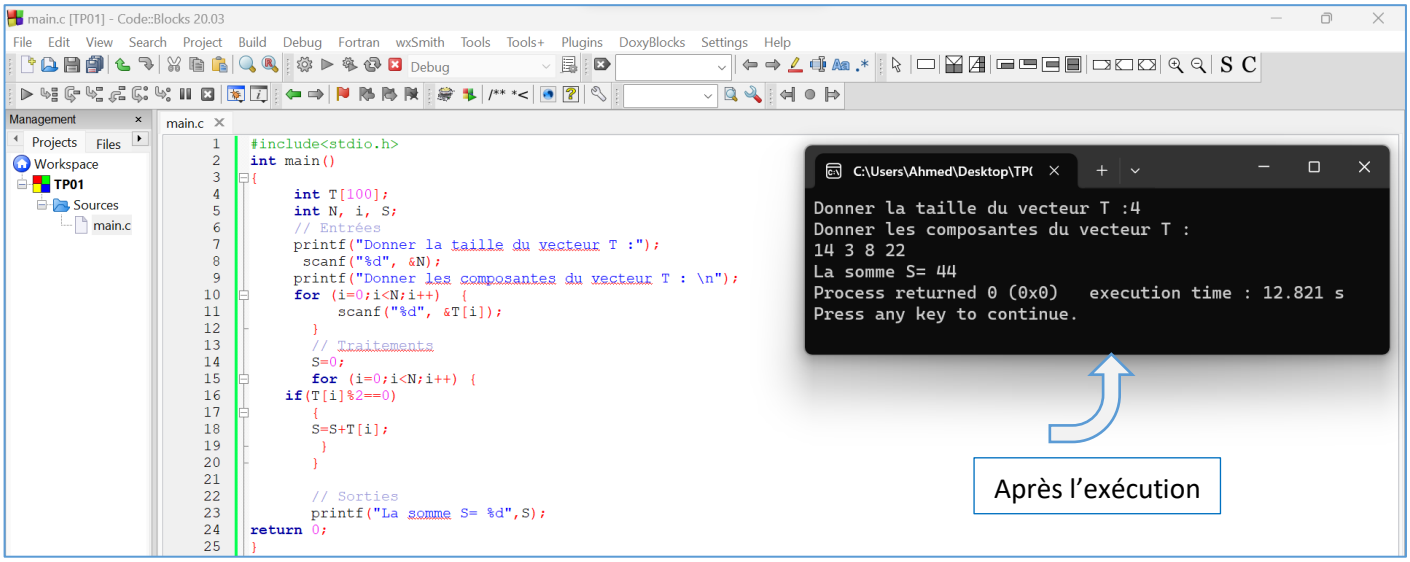
La variable ***i*** est une variable de traitement ou intermédiaire, utilisée pour parcourir le vecteur T.

1- Traduire l'algorithme en programme C

Algorithme	Programme C

Algorithme Vecteur ;
Variables
T : Tableau [0..99] **d'entier** ;
N, i, S : **entier**;
Début
// Entrées
Ecrire("Donner la taille du vecteur T : ");
Lire(N);
Ecrire("Donner les composantes du vecteur T : ");
Pour i←0 **à** N-1 **faire**
Lire(T[i]);
FinPour;
// Traitements
S ← 0;
Pour i←0 **à** N-1 **faire**
Si (T[i] **mod** 2 = 0) **alors**
S ← S+T[i];
FinSi;
FinPour;
// Sorties
Ecrire("La somme S=", S);
Fin.
	#include<stdio.h>
	int main()
	{
	int T[100];
	int N, i, S;
	// Entrées
	printf("Donner la taille du vecteur T :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur T : \n");
	for (i=0;i<N;i++) {
	scanf("%d", &T[i]);
	}
	// Traitements
	S=0;
	for (i=0;i<N;i++) {
	if(T[i]%2==0)
	{
	S=S+T[i];
	}
	}
	// Sorties
	printf("La somme S= %d",S);
	return 0;
	}

2- Compiler et exécuter le programme pour : N = 4 et T=[14, 3, 8, 22].



3- Dérouler l'algorithme pour : N = 4 et T=[14, 3, 8, 22].

Dérouler un algorithme (ou un programme) consiste à exécuter manuellement les instructions de cet algorithme et à visualiser l'impact de ces instructions sur les variables.

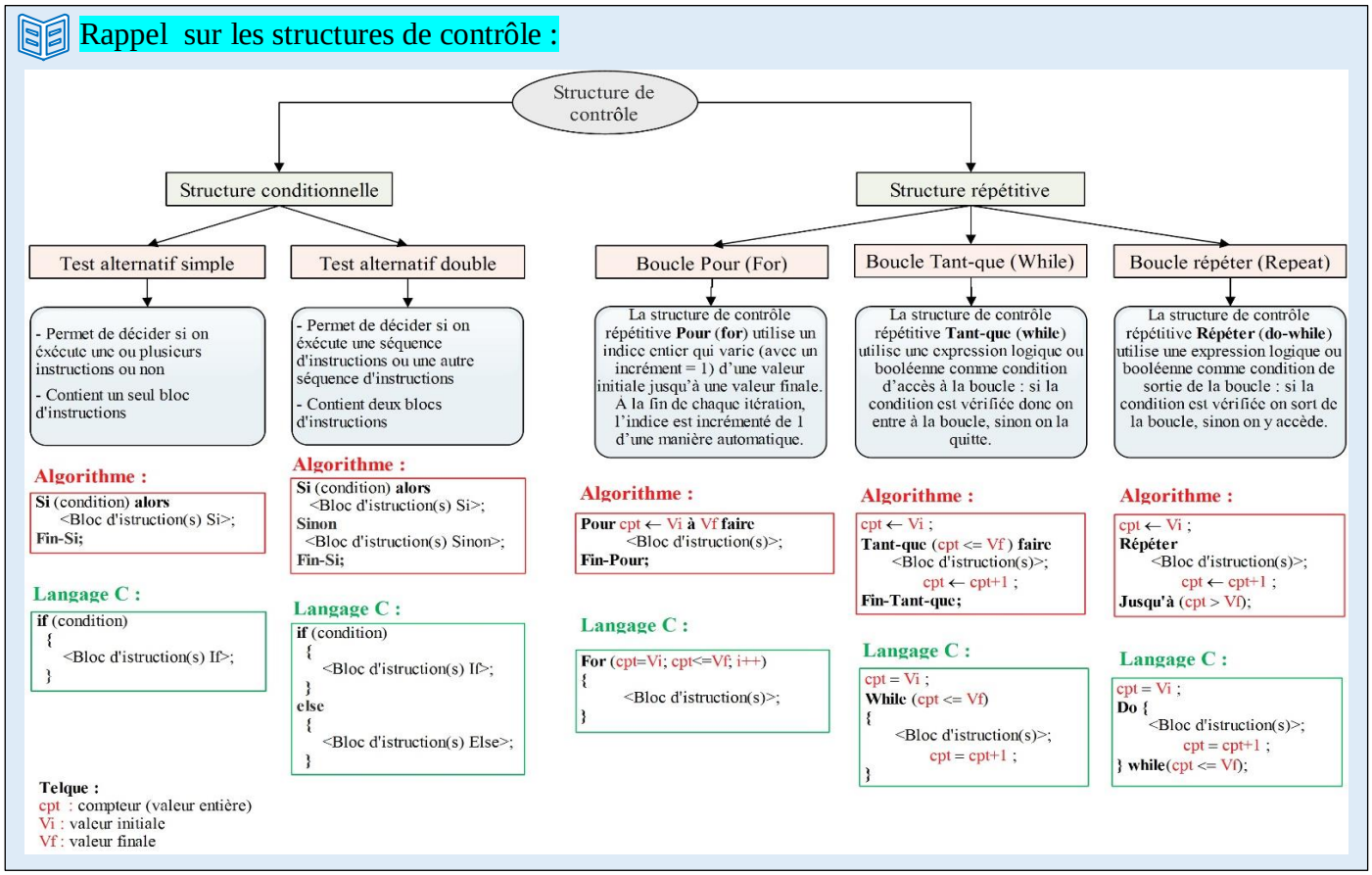
Autrement dit, dérouler un algorithme permet de visualiser les changements des valeurs des variables (évolution des valeurs).

Instructions	Variables				Affichage
	N	i	T	S	
Ecrire("Donner la taille du vecteur T : ");	/	/	/	/	Donner la taille du vecteur T :
Lire(N)	4	/	/	/	//
Ecrire("Donner les composantes du vecteur T : ");	4	/	/	/	Donner les composantes du vecteur T :
Pour i←0 **à** N-1 **faire** Lire(T[i]) **Fin-Pour**	4	0 1 2 3	0 1 2 3 [14, 3, 8, 22]	/	//
S ← 0	4	/	[14, 3, 8, 22]	0	//
Pour i=0 **Si** T[0] mod 2 = 0 **Alors** **14 mod 2 = 0** **Vrai** S ←S+T[i]=S+T[0] S ←0+14=14	4	0	[14, 3, 8, 22]	14	//
Pour i=1 **Si** T[1] mod 2 = 0 **Alors** **3 mod 2 = 0** **Faux**	4	1	[14, 3, 8, 22]	//	//
Pour i=2 **Si** T[2] mod 2 = 0 **Alors** **8 mod 2 = 0** **Vrai** S ←S+T[i]=S+T[2] S ←14+8=22	4	2	[14, 3, 8, 22]	22	//
Pour i=3 **Si** T[3] mod 2 = 0 **Alors** **22 mod 2 = 0** **Vrai** S ←S+T[i]=S+T[3] S ←22+22=44	4	3	[14, 3, 8, 22]	44	//
Ecrire("La somme S = ", S) ;	4		[14, 3, 8, 22]	44	La somme S = 44

4- Déduire ce que fait l'algorithme

L'algorithme permet de réaliser la somme des composantes du vecteur T divisibles par 2 (la somme des nombres pairs).

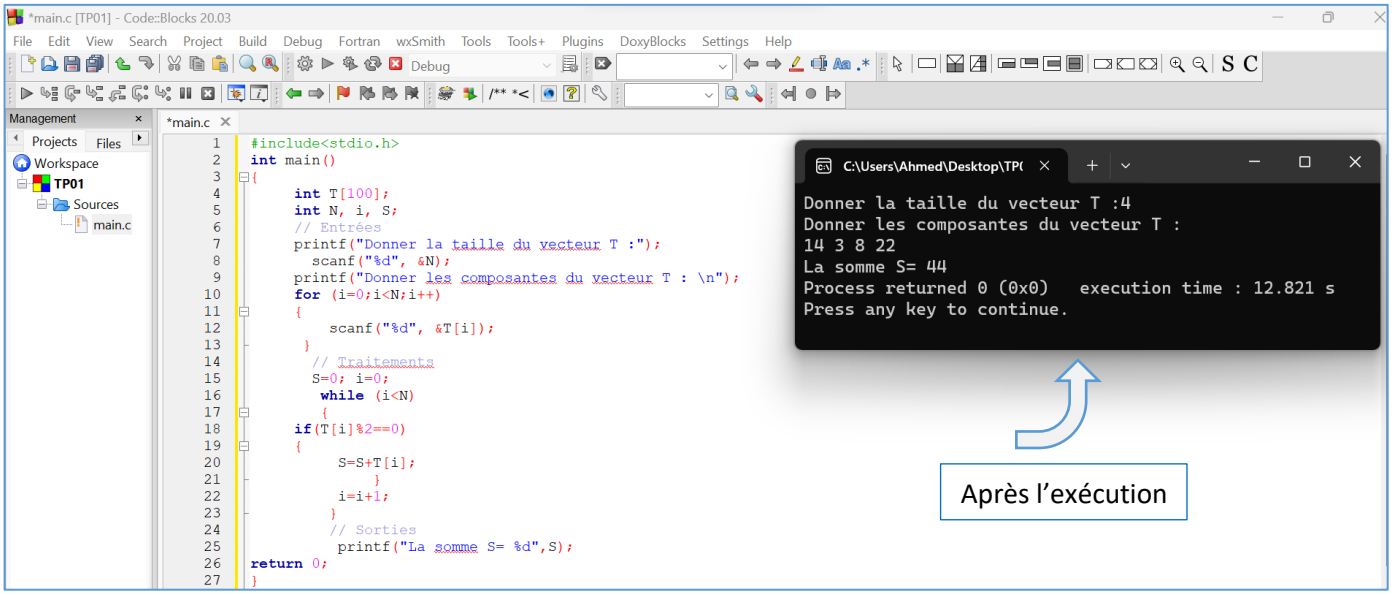
5- Boucle Tant-que (While)



Algorithme/Programme C

Algorithme	Programme C

Algorithme Vecteur ;
Variables
T : Tableau [0..99] **d'entier** ;
N, i, S : **entier**;
Début
// Entrées
Ecrire("Donner la taille du vecteur T : ");
Lire(N);
Ecrire("Donner les composantes du vecteur T : ");
Pour i←0 **à** N-1 **faire**
Lire(T[i]);
FinPour;
// Traitements
S ← 0;
i ← 0;
Tant-que (i<N) **faire**
Si (T[i] **mod** 2 = 0) **alors**
S ← S+T[i];
FinSi;
i ← i+1;
Fin-Tant-que;
// Sorties
Ecrire("La somme S=", S);
Fin.
	#include<stdio.h>
	int main()
	{
	int T[100];
	int N, i, S;
	// Entrées
	printf("Donner la taille du vecteur T :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur T : \n");
	for (i=0;i<N;i++)
	{
	scanf("%d", &T[i]);
	}
	// Traitements
	S=0; i=0;
	while (i<N)
	{
	if(T[i]%2==0)
	{
	S=S+T[i];
	}
	i=i+1;
	}
	// Sorties
	printf("La somme S= %d",S);
	return 0;
	}

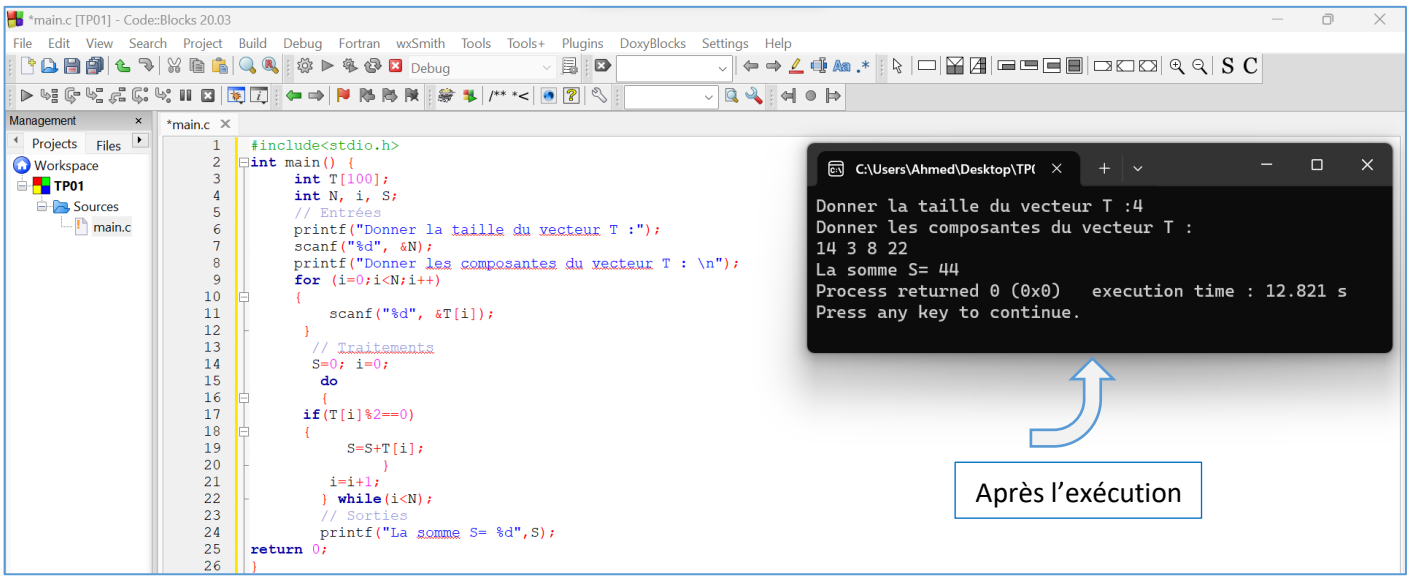


6- Boucle Répéter (do-while)

Algorithme/Programme C

Algorithme	Programme C

Algorithme Vecteur ;
Variables
T : Tableau [0..99] **d'entier** ;
N, i, S : **entier**;
Début
// Entrées
Ecrire("Donner la taille du vecteur T : ");
Lire(N);
Ecrire("Donner les composantes du vecteur T : ");
Pour i←0 **à** N-1 **faire**
Lire(T[i]);
FinPour;
// Traitements
S ← 0; i ← 0;
Répéter
Si (T[i] **mod** 2 = 0) **alors**
S ← S+T[i];
FinSi;
i ← i+1;
Jusqu'à (i>N);
// Sorties
Ecrire("La somme S=", S);
Fin.
	#include<stdio.h>
	int main() {
	int T[100];
	int N, i, S;
	// Entrées
	printf("Donner la taille du vecteur T :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur T : \n");
	for (i=0;i<N;i++)
	{
	scanf("%d", &T[i]);
	}
	// Traitements
	S=0; i=0;
	do
	{
	if(T[i]%2==0)
	{
	S=S+T[i];
	}
	i=i+1;
	} while(i<N);
	// Sorties
	printf("La somme S= %d",S);
	return 0;
	}



Remarques :

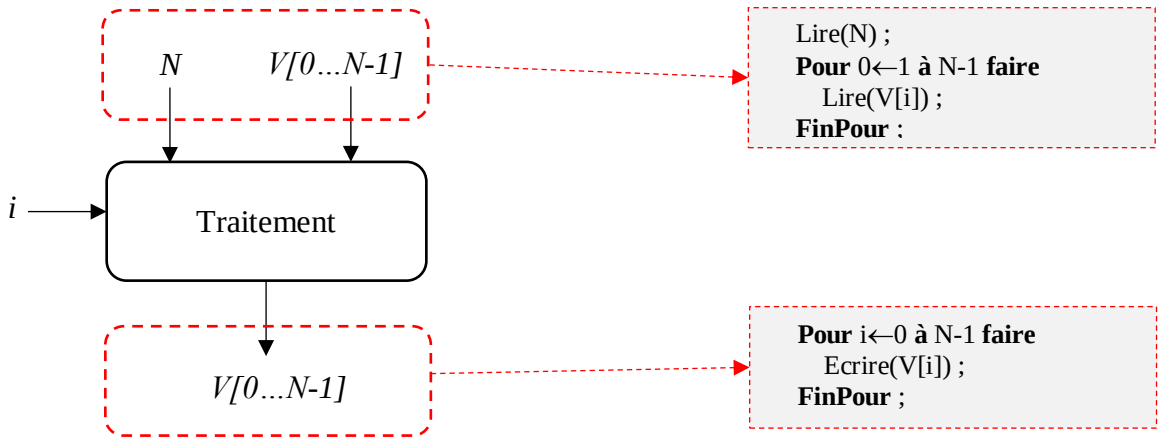
- Un vecteur (Tableau à une dimension) est une suite de cases mémoires adjacentes. Ces cases mémoires définissent des variables de même types.
- La déclaration suivante : T : **Tableau** [0..99] d'entier;
  Signifie que nous réservons 100 cases de type entier. 100 est la taille maximale du vecteur T. Chaque case est accessible par un indice qui peut prendre les valeurs 0 à 99.
- Pendant l'exécution, nous n'utiliserons pas les **100** cases du vecteur, nous utiliserons **N** cases, où **N** est une variable entière qui qui doit être introduite (lue) pendant l'exécution.
- Pour lire un vecteur T, il faut lire la taille que l'on veut utiliser (la variable ***N***) et lire toutes les cases **V[i]** tel-que **i** allant de **0** à la valeur **N-1**. Même chose pour l'affichage.

Exercice N°02 : Lecture et Affichage d'un vecteur

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

Solution :

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



Remarque :

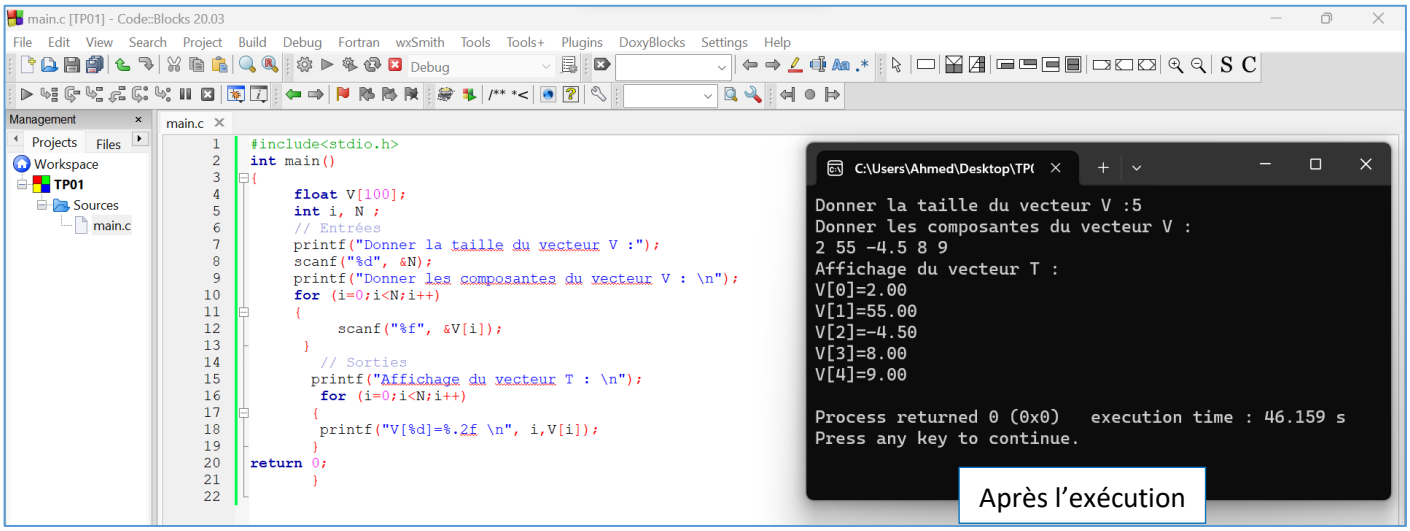
Il faut noter que ce programme ne réalise pas de traitement, il ne contient que des entrées et des sorties.

Algorithme	Programme C

Algorithme Affichage_Vecteur ;
Variables
V : Tableau [0..99] **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←0 **à** N-1 **faire**
Lire(V[i]) ;
FinPour ;
// Sorties
Ecrire("Affichage du vecteur V : ") ;
Pour i←0 **à** N-1 **faire**
Ecrire(V[i]) ;
FinPour ;
Fin.
	#include<stdio.h>
	int main()
	{
	float V[100];
	int i, N ;
	// Entrées
	printf("Donner la taille du vecteur V :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur V : \n");
	for (i=0;i<N;i++)
	{
	scanf("%f", &V[i]);
	}
	// Sorties
	printf("Affichage du vecteur T : \n");
	for (i=0;i<N;i++)
	{
	printf("V[%d]=%.2f \n", i,V[i]);
	}
	return 0;
	}

Une autre méthode :

```
for (i=0;i<N;i++)
  printf("%d", V[i]);
```



Explication☺

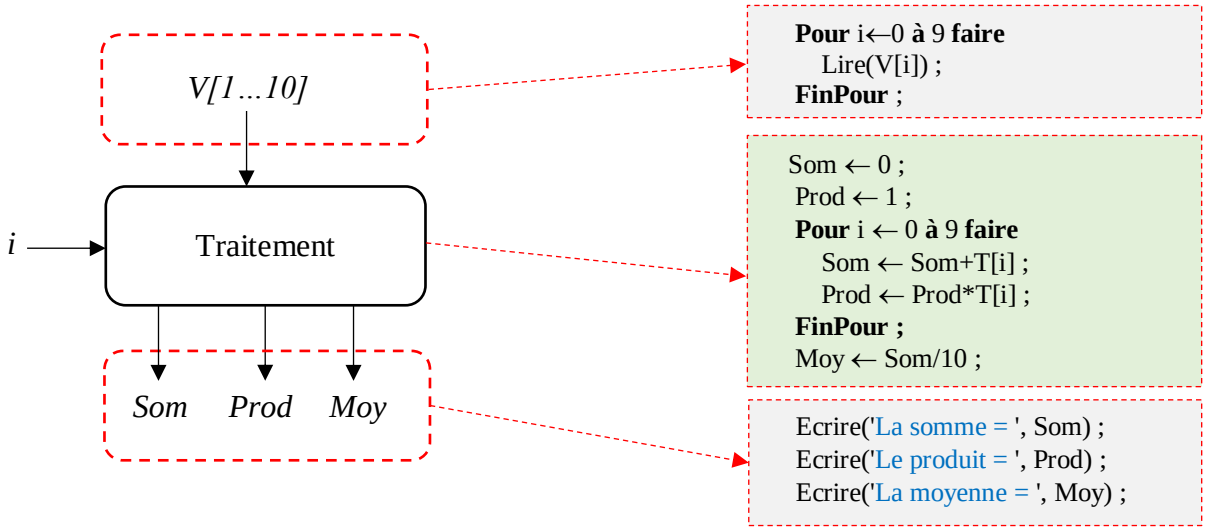
Ce programme montre comment lire et écrire un vecteur. Pour les deux opérations (lecture et écriture), nous aurons toujours besoin d'une boucle **Pour**. Lors de la déclaration, nous réservons 100 cases réelles (la taille maximale du vecteur), et nous utilisons la variable **N** pour déterminer la taille 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[1])**, et **Ecrire(V[3])** pour afficher la valeur de la 4ème case.

Exercice N°03 : La somme, le produit et la moyenne des éléments d'un tableau

Ecrire un algorithme/programme PASCAL qui permet de calculer la somme, le produit et la moyenne des éléments d'un vecteur V de dix réels.

Solution :

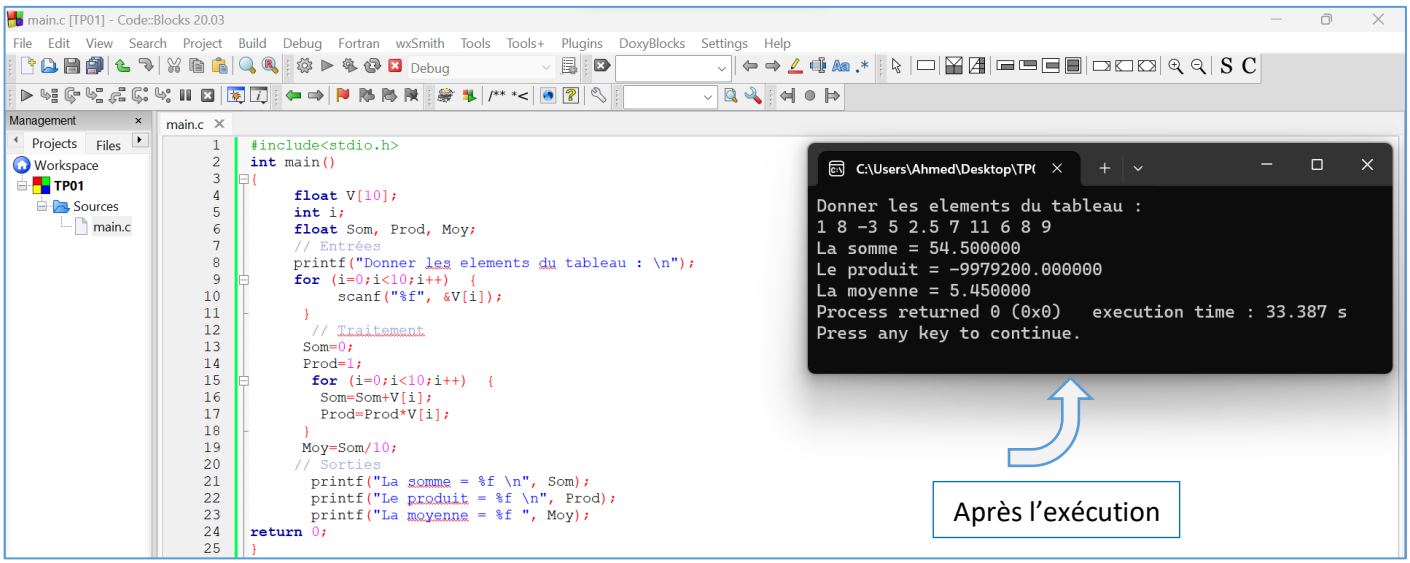
Les variables d'entrée, variable de sorite et la partie traitement sont présentées dans le schéma ci-dessous :



Algorithme/Programme C

Algorithme	Programme PASCAL

Algorithme Som_Moy_Prod_Tab ;
Variables
V : Tableau [0..9] **de réel** ;
i : **entier** ;
Som, Prod, Moy : **réel** ;
Début
// Entrées
Ecrire("Donnez les éléments du tableau : ") ;
Pour i ← 0 **à** 9 **faire**
Lire(V[i]) ;
FinPour ;
// Traitement
Som ← 0 ;
Prod ← 1 ;
Pour i ← 0 **à** 9 **faire**
Som ← Som+V[i] ;
Prod ← Prod*V[i] ;
FinPour ;
Moy ← Som/10 ;
// Sorties
Ecrire("La somme = ", Som) ;
Ecrire("Le produit = ", Prod) ;
Ecrire("La moyenne = ", Moy) ;
Fin.
	#include<stdio.h>
	int main()
	{
	float V[10];
	int i;
	float Som, Prod, Moy;
	// Entrées
	printf("Donner les éléments du tableau : \n");
	for (i=0;i<10;i++) **{**
	scanf("%f", &V[i]);
	}
	// Traitement
	Som=0;
	Prod=1;
	for (i=0;i<10;i++) **{**
	Som=Som+V[i];
	Prod=Prod*V[i];
	}
	Moy=Som/10;
	// Sorties
	printf("La somme = %f \n", Som);
	printf("Le produit = %f \n", Prod);
	printf("La moyenne = %f ", Moy);
	return 0;
	}



Explication ☺

Pour calculer la somme des nombres contenus dans le tableau, il faut ajouter un à un le contenu des cases depuis la première jusqu'à la dernière, en utilisant une variable initialisée à zéro.

Pour calculer le produit des nombres contenus dans le tableau, il faut multiplier un par un le contenu des cases depuis la première jusqu'à la dernière, en utilisant une variable initialisée à un.

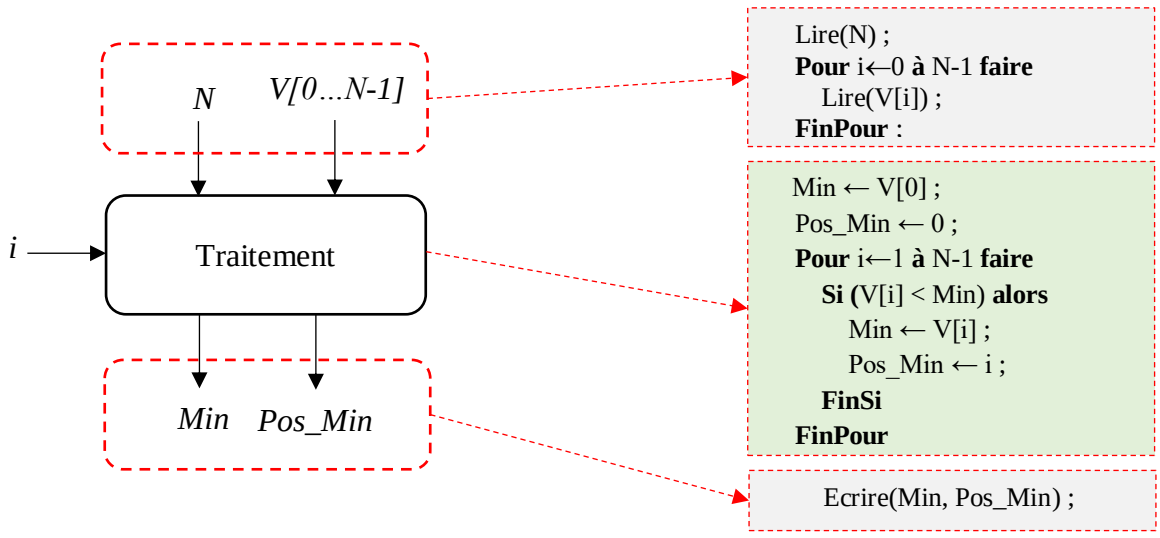
Pour calculer la moyenne, il suffit de diviser la somme par le nombre de cases du tableau.

Exercice N°04 : Le Min et le Max dans un vecteur

1. Ecrire un algorithme/programme C qui permet de rechercher le plus petit élément dans un vecteur réel V ainsi que sa position.
2. Ecrire un algorithme / programme C qui permet de rechercher le plus grand élément dans un vecteur réel V ainsi que sa position.

Solution : (Question 1)

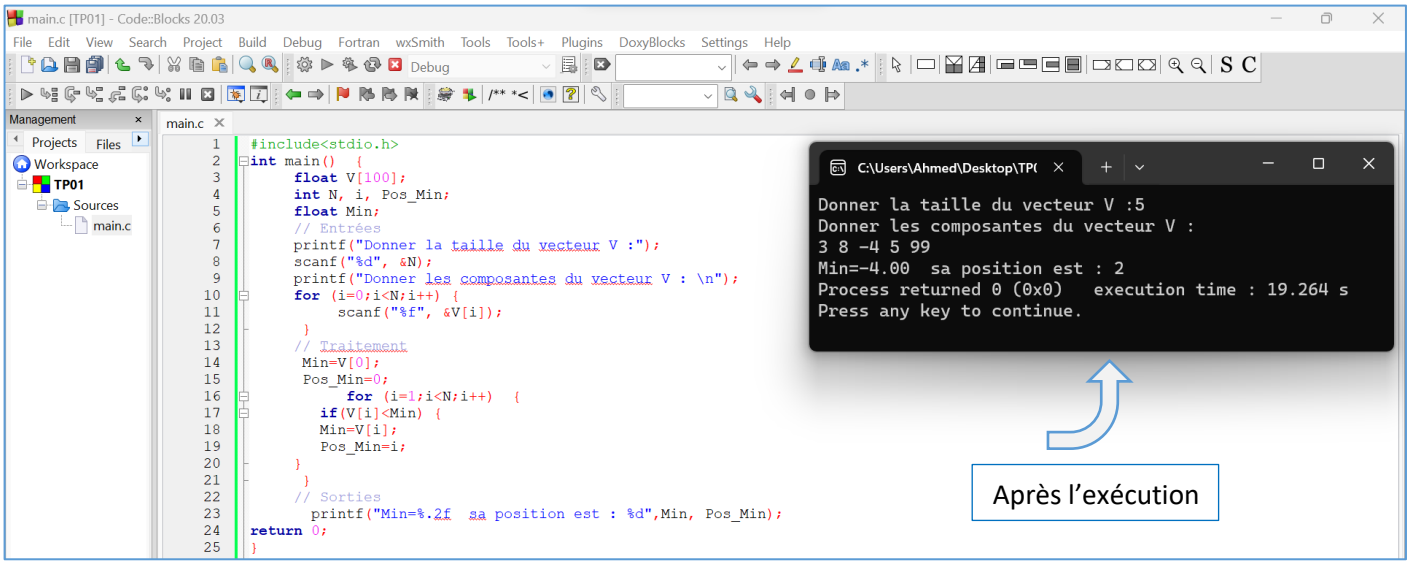
Les variables d'entrée, variables de sorite et la partie traitement sont présentées dans le schéma ci-dessous :



Algorithme/Programme C

Algorithme	Programme C

Algorithme Min_PosMin ;
Variables
V : Tableau [0..99] **de réel ;**
i, N, Pos_Min : **entier** ;
Min : **réel** ;
Début
// Entrées
Ecrire("Donner la taille du vecteur V : ");
Lire(N) ;
Ecrire("Donner les composantes du vecteur V : ");
Pour i←0 **à** N-1 **faire**
Lire(V[i]) ;
FinPour
// Traitement
Min ← V[0] ; Pos_Min ← 0 ;
Pour i←1 **à** N-1 **faire**
Si (V[i] < Min) **alors**
Min ← V[i] ;
Pos_Min ← i ;
FinSi
FinPour
// Sorties
Ecrire(Min, Pos_Min) ;
Fin.
	#include<stdio.h>
	int main() {
	float V[100];
	int N, i, Pos_Min;
	float Min;
	// Entrées
	printf("Donner la taille du vecteur V :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur V : \n");
	for (i=0;i<N;i++) {
	scanf("%f", &V[i]);
	}
	// Traitement
	Min=V[0]; Pos_Min=0;
	for (i=1;i<N;i++) {
	if(V[i]<Min) {
	Min=V[i];
	Pos_Min=i;
	}
	}
	// Sorties
	printf("Min=%.2f sa position est : %d",Min, Pos_Min);
	return 0;
	}

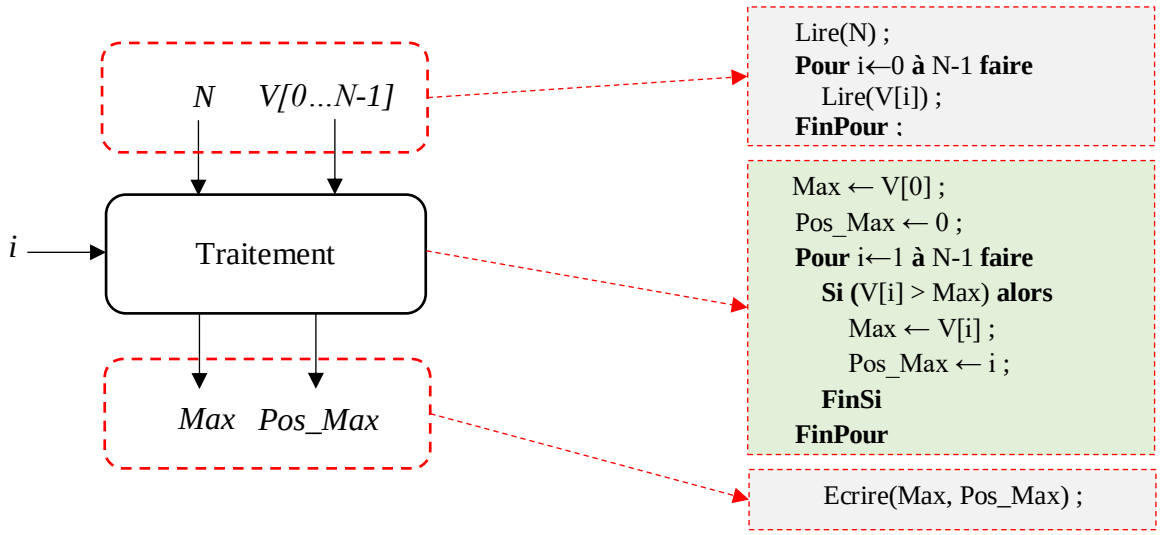


Explication☺

La solution de la recherche du min dans un vecteur ainsi que sa position commence par la supposition que le premier élément du vecteur est le minimum (*Min:=V[0] ; Pos_Min:=0;*). Par la suite, on parcours toutes les autres cases de l'indice *2* jusqu'à l'indice N pour vérifier s'il y parmi ces cases celles qui vérifiassent la condition *V[i]<Min*, pour chaque élément *V[i]* qui vérifie la condition précédente, on met à jour le *Min* par *V[i]* et la valeur de *Pos_Min* par *i*.

Solution : (Question 2)

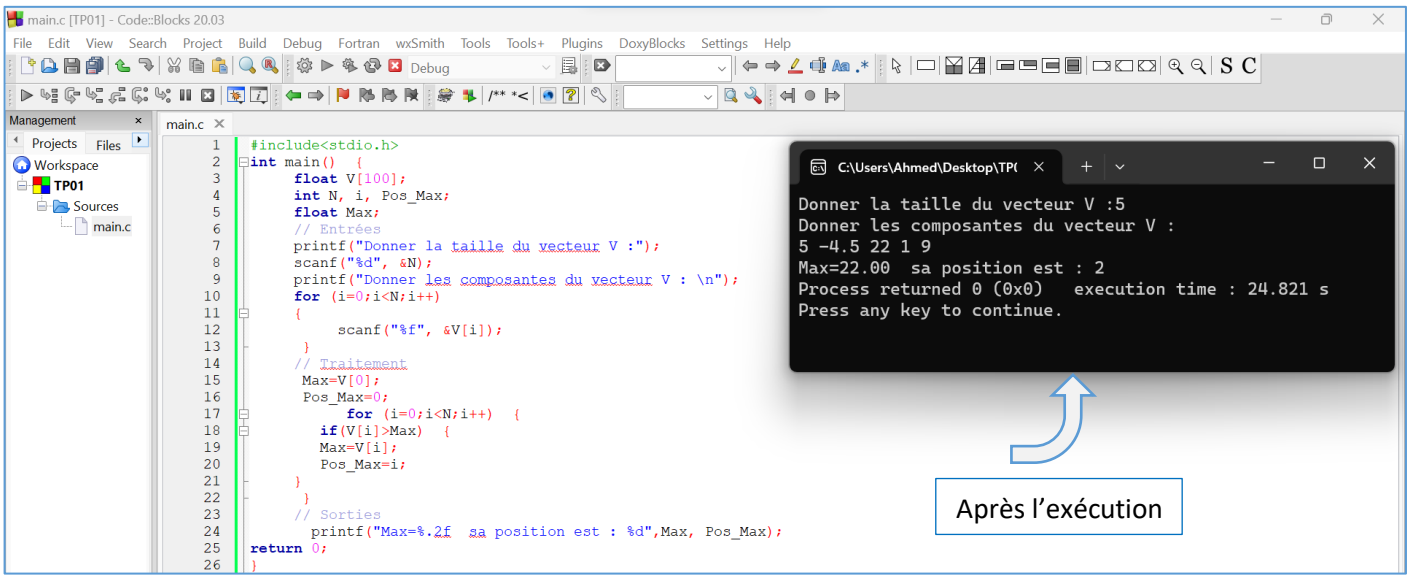
Les variables d'entrée, variables de sorite et la partie traitement sont présentées dans le schéma ci-dessous :



Algorithme/Programme C

Algorithme	Programme C

Algorithme Max_PosMax ;
Variables
V : Tableau [0..99] **de réel** ;
i, N, Pos_Max : **entier** :
Max : **réel** ;
Début
// Entrées
Ecrire("Donner la taille du vecteur V : ");
Lire(N) ;
Ecrire("Donner les composantes du vecteur V: ");
Pour i←0 **à** N-1 **faire**
Lire(V[i]) ;
FinPour
// Traitement
Max ← V[0] ;
Pos_Max ← 0 ;
Pour i←1 **à** N-1 **faire**
Si (V[i] > Max) **alors**
Max ← V[i] ;
Pos_Max ← i ;
FinSi
FinPour
// Sorties
Ecrire(Max, Pos_Max) ;
Fin.
	#include<stdio.h>
	int main() {
	float V[100];
	int N, i, Pos_Max;
	float Max;
	// Entrées
	printf("Donner la taille du vecteur V :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur V : \n");
	for (i=0;i<N;i++)
	{
	scanf("%f", &V[i]);
	}
	// Traitement
	Max=V[0];
	Pos_Max=0;
	for (i=0;i<N;i++) {
	if(V[i]>Max) {
	Max=V[i];
	Pos_Max=i;
	}
	}
	// Sorties
	printf("Max=%.2f sa position est : %d",Max,
	Pos_Max);
	return 0;
	}



Explication☺

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

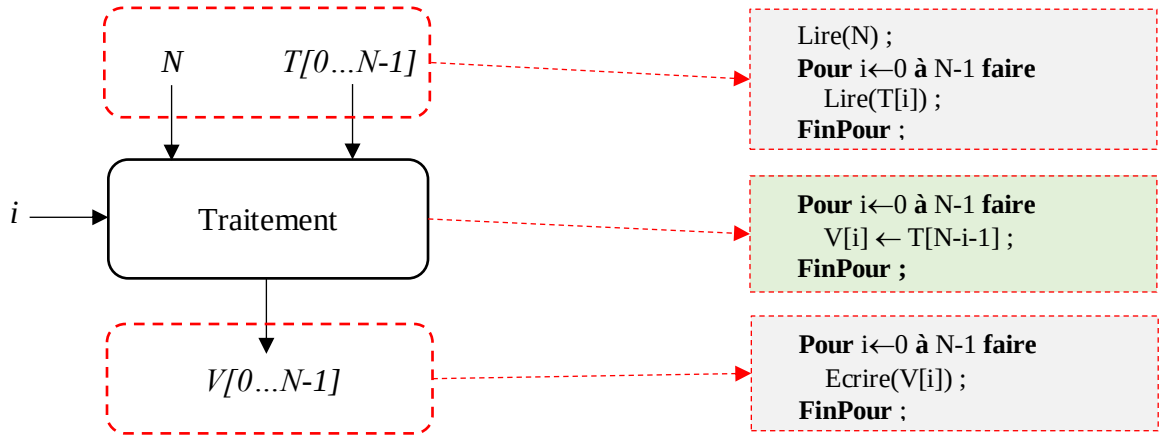
Exercice N°05 : Inverser les éléments d'un vecteur

1. Ecrire un algorithme/programme C qui permet d'inverser les éléments d'un vecteur de type réel T dans un autre vecteur V.
2. Réaliser la même opération dans le même vecteur T (sans utiliser le vecteur V).

Solution :

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 0 à N-1, avec N=6)

Pour i=0 ➔ V[0]=T[5]	V[0]=T[6-0-1]
Pour i=1 ➔ V[1]=T[4]	V[1]=T[6-1-1]
Pour i=2 ➔ V[2]=T[3]	V[2]=T[6-2-1]
Pour i=3 ➔ V[3]=T[2]	V[3]=T[6-3-1]
Pour i=4 ➔ V[4]=T[1]	V[4]=T[6-4-1]
Pour i=5 ➔ V[5]=T[0]	V[5]=T[6-5-1]

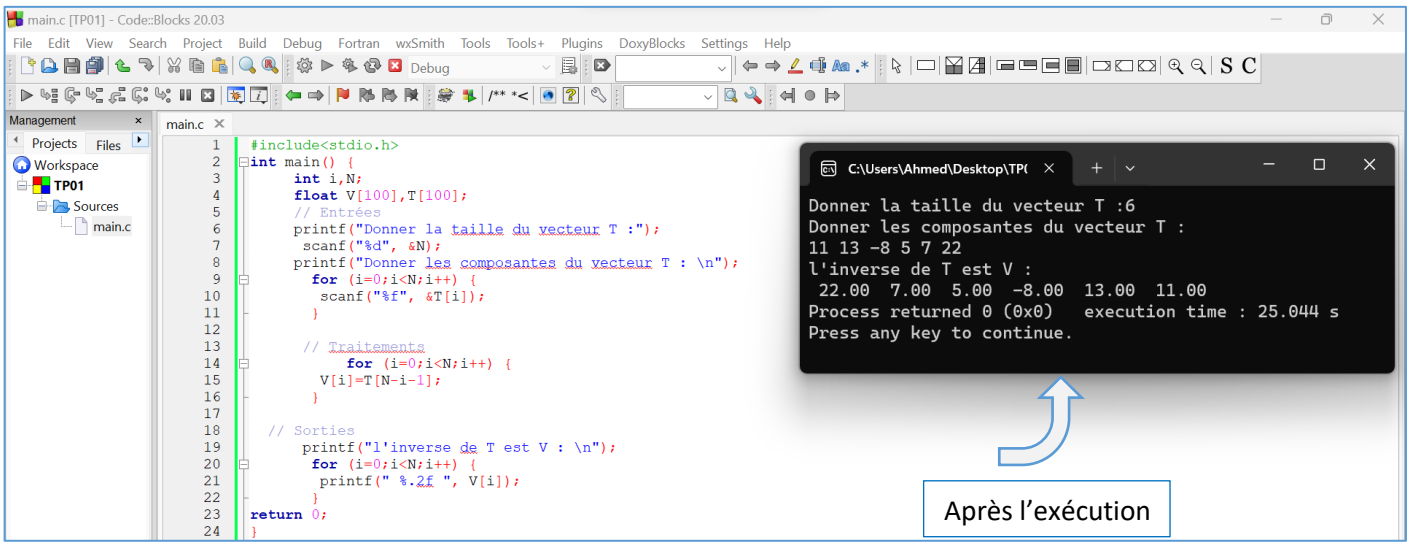
Pour toutes égalités, on peut écrire :

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

Algorithme/Programme Pascal

Algorithme	Programme C

Algorithme inverser_T_dans_V;
Variables
i,N : **entier**;
V,T : Tableau [0..99] **de réel**;
Début
// Entrées
Ecrire("Donner la taille du vecteur T : ");
Lire(N);
Ecrire("Donner les composantes de T : ");
Pour i←0 **à** N-1 **faire**
Lire(T[i]);
FinPour
 	
// Traitement
Pour i←0 **à** N-1 **faire**
V[i] ← T[N-i-1];
FinPour
 	
// Sorties
Ecrire("L'inverse de T est V : ");
Pour i←0 **à** N-1 **faire**
Ecrire(V[i]);
FinPour
Fin.
	#include<stdio.h>
	int main() {
	int i,N;
	float V[100],T[100];
	// Entrées
	printf("Donner la taille du vecteur T :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur T : \n");
	for (i=0;i<N;i++) **{**
	scanf("%f", &T[i]);
	}
	
	// Traitements
	for (i=0;i<N;i++) {
	V[i]=T[N-i-1];
	}
	
	// Sorties
	printf("l'inverse de T est V : \n");
	for (i=0;i<N;i++) {
	printf(" %.2f ", V[i]);
	}
	return 0;
	}



Question 02 : Inverser le vecteur T dans lui même

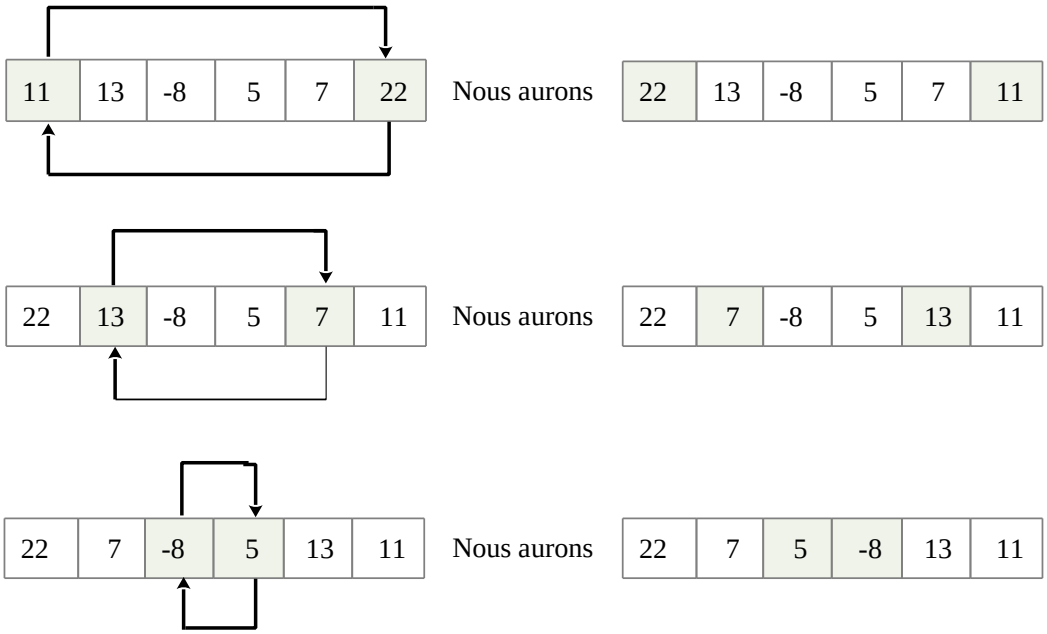
Le même problème que la question 01, sauf que, dans cette deuxième question, nous voulons inverser le vecteur T dans lui-même.

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).

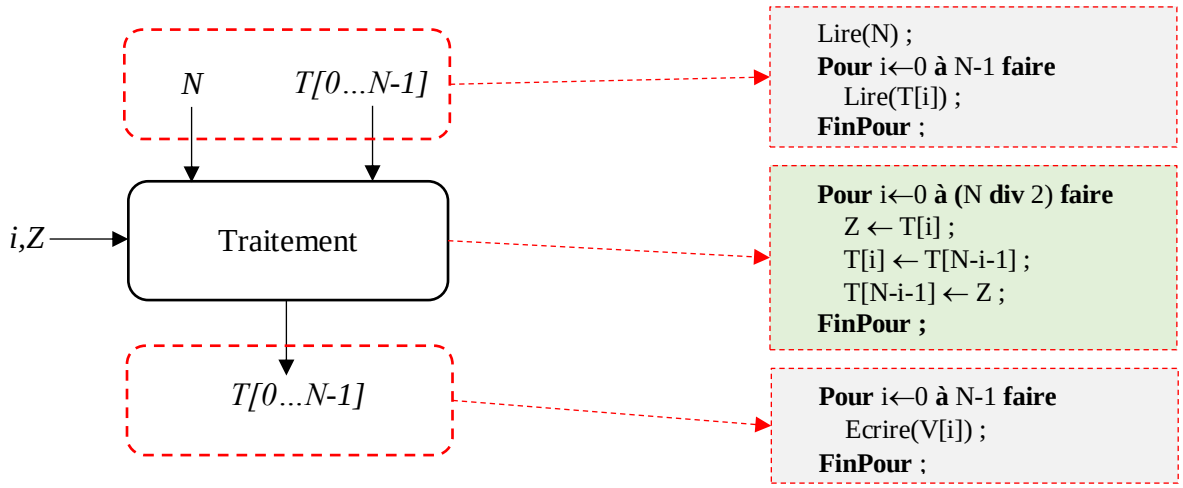
Remarques à retenir :

- Inverser un vecteur dans lui-même en permutant des cases.
- Le nombre de permutations est la moitié du vecteur.

- Les permutations : (0 avec N-1), (1 avec N-2), … 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 = 0… (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 0 à (N div 2)
```

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



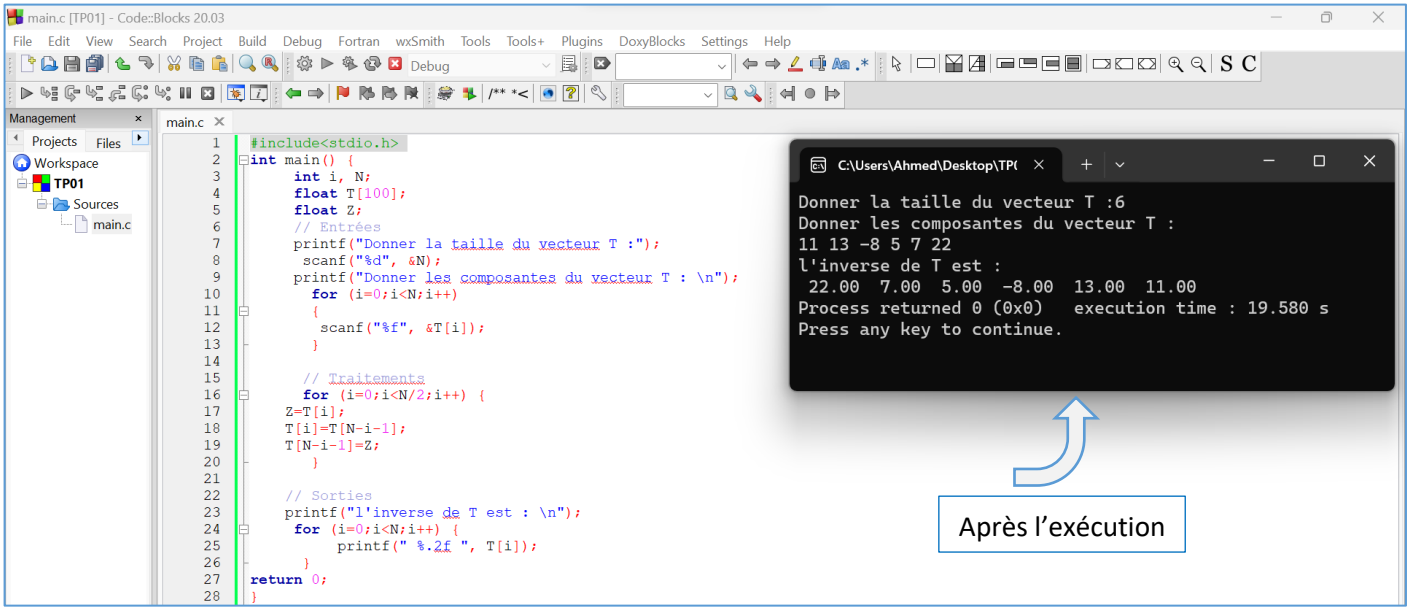
Algorithme/Programme C

Algorithme	Programme C

Algorithme inverser_T_dans_T;
Variables
i, N : **entier**;
T : Tableau [0..99] **de réel**;
Z : **réel**;
Début
// Entrées
Ecrire("Donner la taille du vecteur T : ");
Lire(N);
Ecrire("Donner les composantes de T : ");
Pour i←0 **à** N-1 **faire**
Lire(T[i]);
FinPour ;
 	
// Traitement
Pour i←0 **à** (N div 2) **faire**
Z ← T[i];
T[i] ← T[N-i-1];
T[N-i-1] ← Z;
FinPour ;
	#include<stdio.h>
	int main() {
	int i, N;
	float T[100];
	float Z;
	// Entrées
	printf("Donner la taille du vecteur T :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur T : \n");
	for (i=0;i<N;i++)
	{
	scanf("%f", &T[i]);
	}
	
	// Traitements
	for (i=0;i<N/2;i++) {
	Z=T[i];
	T[i]=T[N-i-1];
	T[N-i-1]=Z;
	}

// Sorties
Ecrire("L'inverse de T est : ");
Pour i←0 **à** N-1 **faire**
  Ecrire(T[i]);
FinPour ;
Fin.
	// Sorties
	printf("l'inverse de T est : \n");
	for (i=0;i<N;i++) {
	  printf(" %.2f ", T[i]);
	}
	return 0;
	}

N.B : div veut dire la division entière



Exercice N°06 : La recherche d'une valeur dans un vecteur.

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

Solution :

Explication☺

La recherche d'une valeur X dans un vecteur V de N éléments donnera deux résultats possibles :

1. X ne se trouve pas dans le vecteur V
2. X se trouve dans le vecteur V dans tel position

Pour le traitement, nous allons parcourir (une boucle) les éléments du vecteur V pour chercher si V[i] est égale à X. Dès qu'on trouve V[i]=X, on sauvegarde la position i dans une variable pos_X, et on arrête la boucle de recherche. Sinon, nous allons à l'itération suivante. A la fin de cette boucle (la boucle de recherche), nous devons avoir une indication si la valeur a été trouvée ou non ?

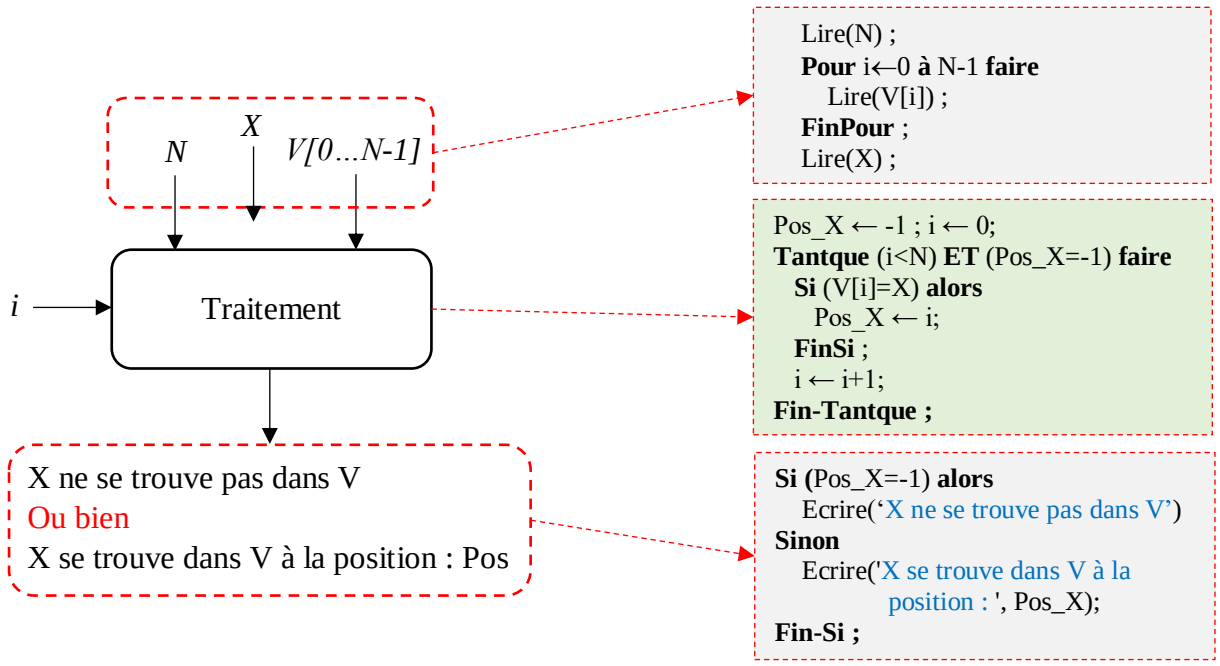
Une première idée, est d'initialiser la variable pos_X à -1, (pos_X ← -1), par la suite, nous allons parcourir toutes les cases de V (V[0], V[1], ..., V[N-1]) pour chercher une case V[i] égale à X. Nous aurons deux cas :

- Si on trouve une case V[i] = X alors : pos_X ← i et on arrête la boucle (puisque nous avons trouvé X dans V).
- Sinon, (aucun V[i] = X), nous allos avoir à la fin pos_X = -1.

Ainsi, à la fin de boucle, nous testons pos_X à (-1) :

- Si pos_X = -1 : la valeur X ne se trouve pas dans X.
- Si pos_X différent de -1 : X se trouve dans V à la position : pos_X.

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



Algorithme/Programme C

Algorithme	Programme C

Algorithme rechercher_X_dans_V;
Variables
V : Tableau [0..99] **de réel**;
i, N, Pos_X : **entier**;
X : **réel**;
Début
// Entrées
Ecrire("Donner la taille du vecteur V : ");
Lire(N);
Ecrire("Donner les composantes de V : ");
Pour i←0 **à** N-1 **faire**
Lire(V[i]);
FinPour ;
Ecrire("Donner la valeur rechercher : ");
Lire(X);
// Traitement
Pos_X ← -1 ; i ← 0;
Tantque (i<N **ET** Pos_X=-1) **faire**
Si (V[i]=X) **alors**
Pos_X ← i;
FinSi ;
i ← i+1;
Fin-Tantque ;
// Sorties
Si (Pos_X=-1) **alors**
Ecrire(X, " ne se trouve pas dans V")
Sinon
Ecrire(X,"se trouve dans V à la position : ", Pos_X);
FinSi ;
Fin.
	#include<stdio.h>
	int main() {
	float V[100];
	int i, N, Pos_X;
	float X;
	// Entrées
	printf("Donner la taille du vecteur V :");
	scanf("%d", &N);
	printf("Donner les composantes du vecteur V : \n");
	for (i=0;i<N;i++) {
	scanf("%f", &V[i]);
	}
	printf("Donner la valeur rechercher : ");
	scanf("%f", &X);
	// Traitements
	Pos_X=-1; i=0;
	while (i<N **&&** Pos_X==-1) {
	if(V[i]==X) {
	Pos_X=i;
	}
	i=i+1;
	}
	// Sorties
	if (Pos_X==-1) {
	printf("%f ne se trouve pas dans V", X);
	}
	else {
	printf("%f se trouve dans V à la position %d ",X, Pos_X);
	}
	return 0;
	}