République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Abderrahmane Mira - Bejaia

Faculté de la Technologie

Département de Technologie

Polycopié de cours

Première Année Technologie – Cycle LMD

Initiation à la programmation

Enseignant :

Dr. OUARET Ahmed

Année Universitaire : **2025/2026**

CHAPITRE I

Bases de l'algorithmique et de la programmation en langage C

Bases de l'algorithmique et de la programmation en langage C

I.1. Objectif de ce chapitre

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

- Comprendre les notions de base de l'informatique et les principes fondamentaux de la programmation.
- Maîtriser la structure d'un programme en langage C ainsi que les types de données et les variables.
- Utiliser les opérations d'entrées/sorties et les expressions pour traiter l'information.
- Mettre en œuvre les structures de contrôle conditionnelles et itératives pour résoudre des problèmes simples.

I.2. Concept d'un algorithme

- Le mot « Algorithme » est inventé par le mathématicien « AL-KHAWARISMI ». Un Algorithme est l'énoncé d'une séquence d'actions primitives réalisant un traitement. Il décrit le plan ou les séquences d'actions de résolution d'un problème donné.
- Un algorithme est un ensemble d'*actions* (ou d'*instructions*) séquentielles et logiquement ordonnées, permettant de transformer des données en entrée (*Inputs*) en données de sorties (*outputs* ou les *résultats*), afin de résoudre un problème.

Donc, un algorithme représente une solution pour un problème donné. Cette solution est spécifiée à travers un ensembles d'instructions (séquentielles avec un ordre logique) qui manipulent des données. Une fois l'algorithme est écrit (avec n'importe quelle langues : français, anglais, arabe,*etc*.), il sera transformé, après avoir choisi un langage de programmation, en un programme code source qui sera compilé (traduit) et exécuté par l'ordinateur.

Pour le langage de programmation qui sera utilisé, ça sera le langage ***C***.

I.3. La démarche et analyse d'un problème

Comme vu dans le point précédent, un algorithme représente une solution à un problème donné. Pour atteindre à cette solution algorithmique un processus d'analyse et de résolution sera appliqué.Ce processus est constitué des étapes illustrées dans la figure 1.

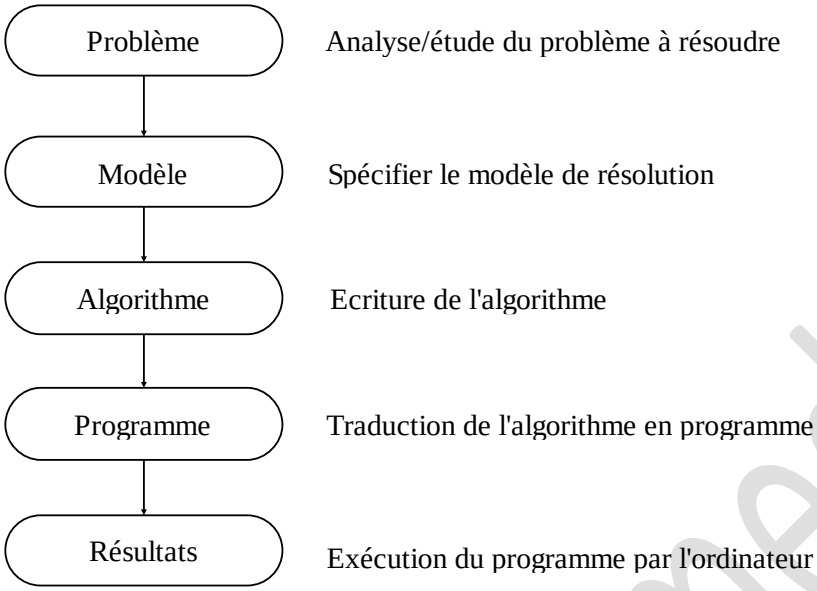


Figure 1 : Etapes d'analyse d'un problème

I.4. Structure d'un algorithme/programme

Un algorithme manipule des données, les données avant de les utiliser il faut les identifier et les déclarer en utilisant les identificateur. Un algorithme est constitué de trois parties :

- ***Entête :*** dans cette partie on déclare le nom de l'algorithme à travers un **identificateur**.
- ***Déclarations :*** dans cette partie on déclare toutes les données utilisées par l'algorithme.
- ***Corps :*** représente la séquence d'actions (instructions) Pour écrire un algorithme, il faut suivre la structure suivante :

Tableau 1 : Structure d'un algorithme

Algorithme	Langage C
Algorithme <identificateur_algo> ; <Déclarations> ; **Début** <Instructions> ; **Fin.**	**#include <stdio.h>** **int main ()** **{** <Déclarations> ; <Instructions> ; **return 0 ;** **}**

❖ Remarques

- Pour commenter un programme en langage C, on écrit les commentaires entre /*…. */. Par exemple : /**Ceci est un commentaire*/*

- On peut aussi utiliser les commentaire ligne :
  // Ceci est un commentaire ligne
- En langage C, la partie déclarative peut être avant la fonction main, comme peut être dans la fonction main.
- En langage C, la fonction main est le point d'entrée du programme : la première qui sera exécutée.

I.4.1. Notion d'identificateur

Un identificateur est une chaîne de caractères contenant uniquement des caractère alpha-numériques (alphabétiques de [a-z] et [A-Z] et numérique [0-9]) et tiré 8 '_' (trait souligné), et qui doit commencer soit par un une lettre alphabétique ou _.

Un identificateur permet d'identifier d'une manière unique un algorithme (ou un programme), une variable, une constante, une procédure ou une fonction.

Dans un langage de programmation donnée, on a pas le droit d'utiliser les mots réservés (mots clés) du langage comme des identificateurs. Parmi les mots clés du langage C :

auto, break, case, char, continue, do, double, else, extern, float, for, goto, if, int, long, return short, sizeof, static, struct, switch, typedef, union, unsigned, while, …

Exemple :

Tableau 2 : Exemples d'identificateurs valides et non valides

Identificateur valide	Identificateur non valide
a1	x1 y (à cause du blanc ou l'espace)
a_1	x1-y (à cause du signe –(tiret de 6))
A_1	1xy (commence par un caractère numérique)
x12y	R? (à cause de caractère spécial (?))
x1_y	Prix-HT (à cause du signe – (tiret de 6))
Hauteur	Exo 04 (à cause de l'espace)
Prix_HT	Program (mot clé)
Exo_04	Read (mot clé)

I.4.2. Déclarations

Dans la partie déclaration, on déclare toutes les données d'entrées et de sorties sous forme de **constantes** et de **variables**.

I.4.2.1. Constantes

Les constantes sont des objets contenant des valeurs non modifiables. Les constante sont déclarées comme suit :

```
<identificateur> = <valeur>;
```

Exemple :

Tableau 3 : Exemples de constantes

Algorithme		Langage C
PI = 3.14;	Constante réelle.	**const float** PI = 3.14;
MAX = 10;	Constante entière.	**const int** MAX = 10;
cc = 'a';	Constante caractère.	**const char** cc = "a";
ss = 'algo';	Constante chaîne de caractère.	**const char** ss[10] = "algo";
b1 = true;	Constante booléenne.	**const int** b1 = 1;
b2 = false;	Constante booléenne.	**const int** b2 = 0;

I.4.2.2. Variables

Les variables sont des objets contenant des valeurs pouvant êtes modifiées. Les variables sont déclarées comme suit :

```
<identificateur> : <type>;
```

Une variable appartient à un type de données. On a cinq types de données de base :

- *Entiers* : représente l'ensemble {…, -4, -3, -2, -1, 0, 1, 2, 3, 4, ...}
- *Réels* : représente les valeurs numériques fractionnels et avec des virgule fixes (ou flottante)
- *Caractères* : représente tous les caractères imprimable.
- *Chaînes de caractères* : une séquence d'un ou plusieurs caractères
- *Booléens (logique)* : représente les deux valeurs *TRUE* et *FALSE*.

Exemple :

Tableau 4 : Exemples de variables

Algorithme	Langage C	Signification
x : réel	**float** x ;	variable réelle
n, m : entier	**int** n, m ;	deux variables entières
s : chaîne de caractères	**char** s[100];	variables chaîne de caractères
b1, b2, b3 : booleen	**bool** b1, b2, b3;	3 variables booléennes
c1 : caractère	**char** c1;	variable caractère

N.B : En plus des constantes et des variables, il est possible de déclarer de nouveaux types, des fonctions et des procédures.

Le tableau 5 montre la correspondance Algorithme/Langage C

Tableau 5 : Correspondance Algorithme/Langage C

Algorithme	Langage C	Spécificateur
Entier	int	%d
Réel	float	%f
Booléen	bool	%d
Caractère	char	%c
Chaîne	char[ ]	%s

I.4.3. Corps

Le corps d'un algorithme est constitué d'un ensemble d'actions / instructions ordonnées de manière séquentielle et logique. Ces instructions se divisant en cinq types distincts:

- ***Lecture*** **:** Cette opération consiste à introduire des données dans l'algorithme. Une lecture consiste à donner une valeur arbitraire à une variable.
- ***Écriture*** **:** Cette opération implique l'affichage de données. Elle permet d'afficher des résultats ou des messages.
- ***Affectation*** **:** Elle permet de modifier les valeurs des variables en leur assignant de nouvelles valeurs.
- ***Structures de contrôle*** **:** Ces structures permettent de modifier la séquentialité de l'algorithme. Elles sont utilisées pour sélectionner différents chemins d'exécution ou pour répéter un traitement.
    - *Structure de* ***Test alternatif simple / double***
    - *Structure* répétitives (itérative)
        - *La boucle* ***Pour***
        - *La boucle* ***Tant-que***
        - *La boucle* ***Répéter***

Dans le langage C, chaque instruction se termine par un **point-virgule**.

I.5. Types d'instructions

Tous les instructions d'un programme sont écrits dans son corps. (entre *Début* et *Fin*, en C entre { et } de la fonction main). On peut regrouper ces instructions en deux grandes types :

- Les instructions séquentielles simples : Entrées, sorties et affectation.
- Les structures de contrôles : tests et boucles.

I.5.1. Instructions d'Entrées/Sorties (Lecture / Écriture)

I.5.1.1. Entrées (Lecture)

Une instruction d'entrée nous permet dans un programme de donner une valeur quelconque à une variable. Ceci se réalise à travers l'opération de lecture.

La syntaxe et la sémantique d'une lecture est comme suit :

Tableau 6 : La syntaxe et la sémantique d'une lecture

Algorithme	Langage C	Signification
Lire(<id_var>);	scanf ("%c", &<id_var>);	Donner une valeur quelconque à la variable dont l'identifiant <*id_var*>.
Lire(<iv1>, <iv2>, …);	scanf (" %c %c ….", &<iv1> , &<iv2>,…);	Donner des valeurs aux variables <iv1>, <iv2>, *etc.*

❖ **Remarque :** Il est important de noter que l'instruction de lecture concerne uniquement les variables, on peut pas liredes constantes ou des valeurs. Lors de la lecture d'une variable dans un programme C, le programme se bloque en attendant la saisie d'une valeur via le clavier. Une fois la valeur saisie, on valide par la touche ***entrée***, et le programme reprend l'exécution avec l'instruction suivante.

Exemple :

Tableau 7 : Exemples d'entrées

Algorithme	Langage C
Lire (a, b, c) ; Lire (hauteur) ;	scanf("%d %d %d", &a, &b, &c); scanf("%f", &hauteur);

I.5.1.2. Sorties (Écriture)

Une instruction de sortie nous permet dans un programme d'afficher un résultat (données traitées) ou bien un message (chaîne de caractères). Ceci se réalise à travers l'opération d'écriture.

La syntaxe et la sémantique d'une écriture est comme suit :

Tableau 8 : La syntaxe et la sémantique d'une écriture

Algorithme	Langage C	Signification
Ecrire(<id_var> | <id_const> | <valeur>, | <expression>)	printf("%f", <id_var> | <id_const> | <valeur> | <expression>);	Afficher une valeur d'une variable, d'une constante, valeur immédiate ou calculée à travers une expression.

❖ **Remarque :** Il est à noter que l'instruction d'écriture ne concerne pas uniquement les variables, on peut écrire des constantes, valeurs ou des expressions (arithmétiques ou logiques).

Exemple :

Tableau 9 : Exemples de sorties

Algorithme	Langage C	Signification
écrire("Bonjour") écrire(a, b, c)	printf ("Bonjour"); printf("%f %f %f", a, b, c);	Afficher le message Bonjour Afficher les valeurs des variables a, b et c
écrire(5+2)	printf("%f",5+2);	Afficher le résultat de la somme de 5 et 2 : afficher 7

I.5.2. Instruction d'affectation

Une affectation consiste à donner une valeur (immédiate, constante, variable ou calculée à travers une expression) à une variable.

La syntaxe d'une affectation est :

Tableau 10 : La syntaxe d'une affectation

Algorithme	Langage C
<id_varialbe> ← <valeur> | <id_variable> | <expression>	<id_varialbe> = <valeur> | <id_variable> | <expression> ;

Une affectation possède deux parties : la partie gauche qui représente toujours une variable, et la partie droite qui peut être : une valeur, variable ou une expression. La condition qu'une affectation soit correcte est que : la partie droite doit être du même type (ou de type compatible) avec la partie gauche.

Exemple :

Tableau 11 : Exemples d'affectation

Algorithme	Langage C	Signification
a ←5 *b ← a+5* *sup ← a>b*	*a=5;* *b=a+5;* *sup=a>b;*	Mettre la valeur 5 dans la variable a Mettre la valeur de l'expression a+5 dans la variable B a>b donne un résultat booléen, donc sup est une variablebooléenne

I.5.3. Structures de contrôles

En générale, les instructions d'un programme sont exécutés d'une manière séquentielle : la première instruction, ensuite la deuxième, après la troisième et ainsi de suite. Cependant, dans plusieurs cas, on est amené soit à choisir entre deux ou plusieurs chemins d'exécution (un choix entre deux ou plusieurs options), ou bien à répéter l'exécution d'un ensemble d'instructions, pour cela nous avons besoins de structures de contrôle pour contrôler et choisir les chemins d'exécutionsou refaire un traitement plusieurs fois. Les structures de contrôle sont de deux types : Structures decontrôles conditionnelles (Tests) et structures de contrôle répétitives (itératives, boucles).

I.5.3.1. Structures de contrôle conditionnelle

Ces structures sont utilisées pour décider de l'exécution d'un bloc d'instruction : est ce que ce bloc est exécuté ou non. Ou bien pour choisir entre l'exécution de deux blocs différents. Nous avons deux types de structures conditionnelles :

a) Test alternatif simple

Un test simple contient un seul bloc d'instructions. Selon une condition (expression logique), on décide est ce que le bloc d'instructions est exécuté ou non. Si la condition est vraie, on exécute le bloc, sinon on l'exécute pas.

La syntaxe d'un test alternatif simple est comme suit :

Tableau 12 : La syntaxe d'un test alternatif simple

Algorithme	Langage C
si <condition> **alors** <bloc_instructions_si> ; **finsi**;	**if** <condition> **{** <bloc_instructions_si>; **}**

Exemple :

Tableau 13 : Exemple d'un test alternatif simple

Algorithme	Langage C

lire(x)
si x > 2 **alors**
  x ← x + 3 ;
finsi
écrire (x)
	scanf("%d", &x);
	if (x > 2)
	{
	  x= x + 3;
	}
	printf("%d", x);

❖ **Remarque :** Dans le langage C, un bloc est délimité par les deux accolades **{** et **}**.

Si le bloc contient une seule instruction, les accolades **{** et **}** sont facultatifs (on peut les enlever).

b) Test alternatif double

Un test double contient deux blocs d'instructions : on est amené à décider entre le premier bloc ou le seconds. Cette décision est réalisée selon une condition (expression logique ou booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second.

La syntaxe d'un test alternatif double est :

Tableau 14 : La syntaxe d'un test alternatif double

Algorithme	Langage C
si <condition> **alors** <bloc_instructions_si> **sinon** <bloc_instrucitons_sinon> ; **finsi**	**if** <condition> **{** <bloc_instructions_si>; **}** **else** **{** <bloc_instructions_sinon>; **}**

Exemple :

Tableau 15 : Exemple d'un test alternatif double

Algorithme	Langage C

lire(x)
si (x > 2) **alors**
x ← x + 3
sinon
x ← x - 2
finsi
écrire (x)
	scanf("%d", &x);
	if (x > 2)
	{
	x= x + 3;
	}
	else
	{
	x= x - 2;
	}
	printf("%d", x);

❖ **Remarques :**

- Dans le langage C, il faut jamais mettre de pointvirgule après la condition (erreur logique).
- Dans l'exemple précédent, on peut enlever **{ }** du **if** et ceux du **else** puisqu'il y a uneseule instruction dans les deux blocs.

Exemple :

Écrire un algorithme (et un programme C) qui permet d'indiquer si un nombre entier est pair ou non.

I.5.3.2. Structures de contrôle répétitives

Les structures répétitives nous permettent de répéter un traitement un nombre fini de fois. Par exemple, on veut afficher tous les nombre premier entre 1 et N (N nombre entier positif donné). Nous avons trois types de structures itératives (boucles) :

a) Boucle Pour (For)

La structure de contrôle répétitive pour (for en langage C) utilise un indice entier qui varie (avec un incrément = 1) d'une valeur initiale jusqu'à une valeur finale. À la fin de chaque itération, l'indice est incrémenté de 1 d'une manière automatique (implicite).

La syntaxe de la boucle **pour** est comme suit :

Tableau 16 : La syntaxe de la boucle **pour**

Algorithme	Langage C
pour <cpt> ← <vi> **à** <vf> **faire** <instruction(s)> ; **finPour**;	**for (**<cpt>=<vi>**;** <cpt><=<vf>**;** <cpt>++**)** **{** <instruction(s)> ; **}**

<cpt> : compteur (variable entière)

<vi> : valeur initiale <vf> : valeur finale

La boucle pour contient un bloc d'instructions (les instructions à répéter). Si le bloc contient une seule instruction, les accolades **{** et **}** sont facultatifs.

Le bloc sera répété un nombre de fois = (<vf> - <vi> + 1) si la valeur finale est supérieure ou égale à la valeur initiale. Le bloc sera exécuté pour <cpt> = <vi>, pour <cpt> = <vi>+1, pour <cpt> = <vi>+2, …, pour <cpt> = <vf>.

b) Boucle Tant-que (While)

La structure de contrôle répétitive **tant-que** (**while** en langage C) utilise une expression logique ou booléenne comme condition d'accès à la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) donc on entre à la boucle, sinon on la quitte.

La syntaxe de la boucle **tant-que** est comme suit :

Tableau 17 : La syntaxe de la boucle **tant-que**

Algorithme	Langage C
tant-que <condition> **faire** <instruction(s)> ; **fin tant-que**;	**while** (<condition>) **{** <instruction(s)>; **}**

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions tant que la condition est vraie. Une fois la condition est fausse, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après fin **Tantque** (après **}**).

❖ **Remarque :** Il est possible de remplacer toute boucle "**pour**" par une boucle "**tant-que**", cependant, l'inverse n'est pas toujours réalisable.

c) Boucle Répéter (do --- while)

La structure de contrôle répétitive **répéter** utilise une expression logique ou booléenne comme condition de sortie de la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) on sort de la boucle, sinon on y accède (on répète l'exécution du bloc).

La syntaxe de la boucle **répéter** est comme suit :

Tableau 18 : La syntaxe de la boucle **répéter**

Algorithme	Langage C
repéter <instruction(s)> ; **Jusqu'à** <condition>;	**do{** <instruction(s)>; **} while** (<condition2>);

<condition> : expression logique qui peut être vraie ou fausse.

<condition2> : expression logique inverse de <condition>.

On exécute le bloc d'instructions jusqu'à avoir la condition correcte. Une fois la condition est vérifiée, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après **jusqu'à**. Dans la boucle **do …while** (en langage C) la condition est l'inverse de la condition de répéter.

La différence entre la boucle **répéter** et la boucle **tantque** est :

- La condition de **répéter** et toujours l'inverse de la condition **tantque** : pour **répéter** c'est la condition de sortie de la boucle, et pour **tantque** c'est la condition d'entrer.
- Le teste de la condition est à la fin de la boucle (la fin de l'itération) pour **répéter**. Par contre, il est au début de l'itération pour la boucle **tantque**. C'est-à-dire, dans **tantque** on teste la condition avant d'entrer à l'itération, et dans **répéter** on fait l'itération après on teste la condition.

Série de TP N°1 : Les instructions de lecture, écriture et affectation en langage C

But du TP :

Le but du TP est de permettre aux étudiants d'acquérir une compréhension approfondie des fondamentaux de la programmation en langage C, en mettant l'accent sur les aspects essentiels liés aux instructions de lecture, d'écriture et d'affectation.

Exercice N°01 : *(Algorithme ➔ Programme en langage C)*

Soit l'algorithme suivant :

```
Algorithme Exo1;
Constantes
    Pi=3.14 ;
Variables
    R,H,V : Réel;
Début
  //Entrées
  Écrire("Donner le rayon R : ");
  Lire(R);
  Écrire("Donner la hauteur H : ");
  Lire(H);

 //Traitement
  V ← Pi*R*R*H ;

 //Sortie
  Écrire("Volume = ", V);
Fin.
```

Questions :

1- Traduire l'algorithme en Programme C.

2- Compiler et exécuter le programme pour : R = 2 et H=6

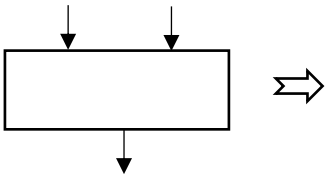
3- Remplacer la dernière instruction : *Écrire("Volume = ", V)*, par une autre instruction qui permet d'afficher le volume avec seulement deux chiffres après la virgule, puis ré-exécuter le programme.

4- Dérouler l'algorithme pour R=2 et H=6

Instructions	Variables				Affichage
	V_1	V_2	…	V_n	
Instruction 1					
Instruction 2					
⋮					
Instruction N					

5- Déduire ce que fait l'algorithme/programme ?

6- Compléter le schéma suivant :



Entrées :

Traitement :

Sortie :

Exercice N°02 : *(Enoncé du problème ➔ Algorithme ➔ Programme en langage C)*

Écrire un algorithme, puis traduit le en programme C, pour chacun des problèmes suivants :

1) Permuter entre les deux variables X et Y ?
2) Permuter entre les trois variables X, Y et Z de telle sorte que la valeur de X soit dans Y, celle de Y dans Z et la valeur de Z dans X ?
3) Calculer le quotient et le reste de la division euclidienne de a par b ?
4) Calculer la somme de a et b et le produit de b et c ?

5) Calculer la valeur absolue et le carré d'un nombre réel ?
6) Convertir en octets un nombre donné en bits ?
7) Lire les notes de trois matières (N1, N2 et N3) ensuite calculer et afficher leur moyenne M ? Modifier l'algorithme dans le cas où des coefficients (C1, C2 et C3) sont attribués aux trois matières.

Exercice N°03 : *Calcul du périmètre d'un cercle*

Écrire un programme en langage C qui demande à l'utilisateur de saisir le rayon d'un cercle, puis qui calcule et affiche le périmètre de ce cercle.

Corrigé de la série de TP N°1

Rappel :

Structure d'un algorithme / programme

Un algorithme manipule des données, les données avant de les utiliser il faut les identifier et les déclarer en utilisant les identificateurs. Un algorithme est constitué de trois parties :

– ***Entête :*** dans cette partie on déclare le nom de l'algorithme à travers un identificateur.

– ***Déclarations :*** dans cette partie on déclare toutes les données utilisées par l'algorithme.

– ***Corps :*** représente la séquence d'actions (instructions)

Pour écrire un algorithme, il faut suivre la structure suivante :

```
Algorithme <id_algorithme>
     <Déclarations>
Début
     <Corps : Instructions>
Fin.
```

```
#include <stdio.h>
int main()
{
     <déclarations>;
     <Instructions>;
     return 0;
}
```

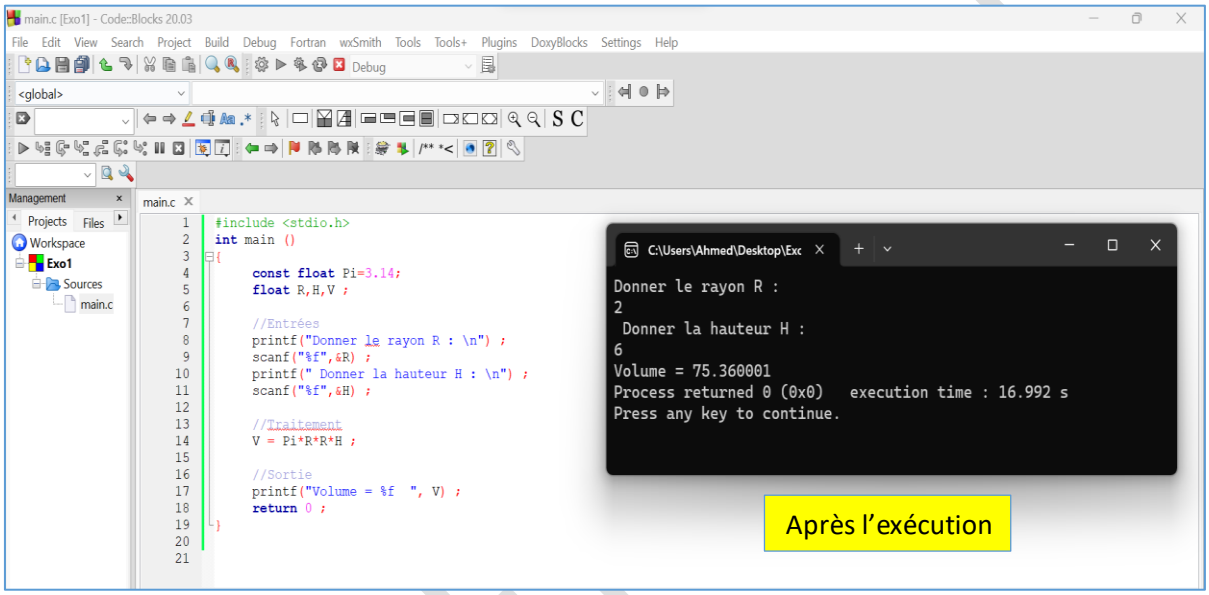
Exercice N°01 : *(Algorithme ➔ Programme C)*

1) Traduire l'algorithme en Programme C

Algorithme	**Programme C**
Algorithme Exo1; **Constantes** Pi=3.14 ; **Variables** R,H,V : **Réel**; **Début** //Entrées **Écrire**("Donner le rayon R : "); **Lire**(R); **Écrire**("Donner la hauteur H :");	**#include <stdio.h>** **int main ()** **{** **const float** Pi=3.14; **float** R,H,V ; //Entrées **printf**("Donner le rayon R : \n") ; **scanf**("%f",&R) ; **printf**(" Donner la hauteur H : \n") ;

Lire(H);
 	
//Traitement
V ← Pi*R*R*H ;
 	
//Sortie
Écrire("Volume = ", V);
Fin.
	scanf("%f",&H) ;
	
	//Traitement
	V = Pi*R*R*H ;
	
	//Sortie
	printf("Volume = %f ", V) ;
	return 0 ;
	}

2) Compiler et exécuter le programme pour : R = 2 et H=6



3) Remplacer la dernière instruction :

Écrire("Volume = ", V), par une autre instruction qui permet d'afficher le volume avec seulement deux chiffres après la virgule,

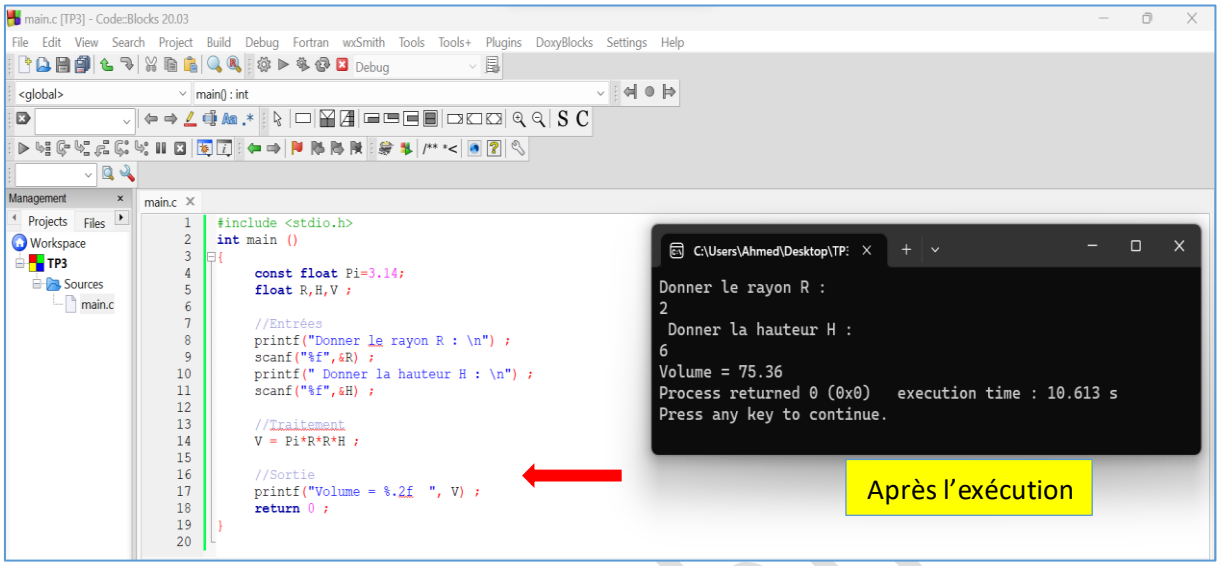
Programme C

```
#include <stdio.h>
int main ()
{
    const float Pi=3.14;
    float R,H,V ;
    //Entrées
    printf("Donner le rayon R : \n") ;
    scanf("%f",&R) ;
    printf(" Donner la hauteur H : \n") ;
    scanf("%f",&H) ;
    //Traitement
    V = Pi*R*R*H ;
    //Sortie
    printf("Volume = %.2f  ", V) ;
```

printf("Volume = %.2f ", V) ;

Où 2 représente deux chiffres après la virgule.

```
    return 0 ;
}
```



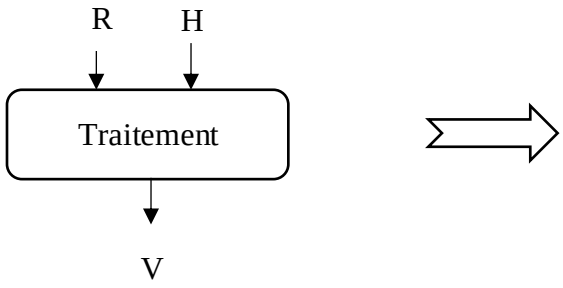
4) Déroulement de l'algorithme pour R=2 et H=6

Instructions	Variables			Affichage
	R	H	V	
Écrire("Donner le rayon R :")	/	/	/	Donner le rayon R :
Lire (R)	2	/	/	
Écrire("Donner la hauteur H :")	2	/	/	Donner la hauteur H :
Lire (H)	2	6	/	
V ← Pi*R*R*H	2	6	75.360001	
Ecrire ("Volume =", P) ;	2	6	75.360001	Volume = 75.360001

5) Déduire ce que fait le programme ?

Le programme calcule le volume d'un cylindre

6) Compléter le schéma suivant :



Entrées :

Lire (R) ;
Lire (H) ;

Traitements :

V ← Pi*R*R *H ;

Sortie :

Écrire (V) ;

Chaque algorithme possède des variables d'entrée, des variables de sorties, constantes et une partie du traitement :

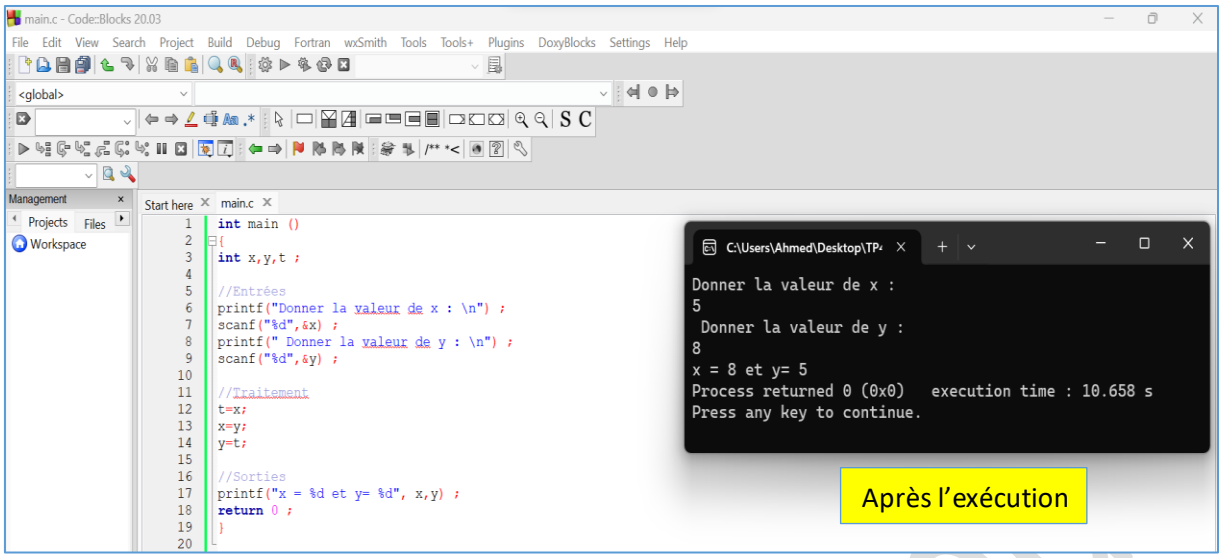
- Les variables d'entrée sont les variables lues (l'instruction **Lire**) ;
- Les variables de sorties sont les variables affichées (l'instruction **Ecrire**) ;
- Les données intermédiaires qui peuvent être des variables ou des constantes (dans notre cas **Pi**) qui sont des données non lues et non écrites utilisées pendant le traitement ;
- Traitement : contient les instructions d'affectation, tests et les boucles.

Exercice N°02 : *(Enoncé du problème ➔ Algorithme ➔ Programme C)*

1) Permuter entre les deux variables X et Y ?

Algorithme	**Programme C**

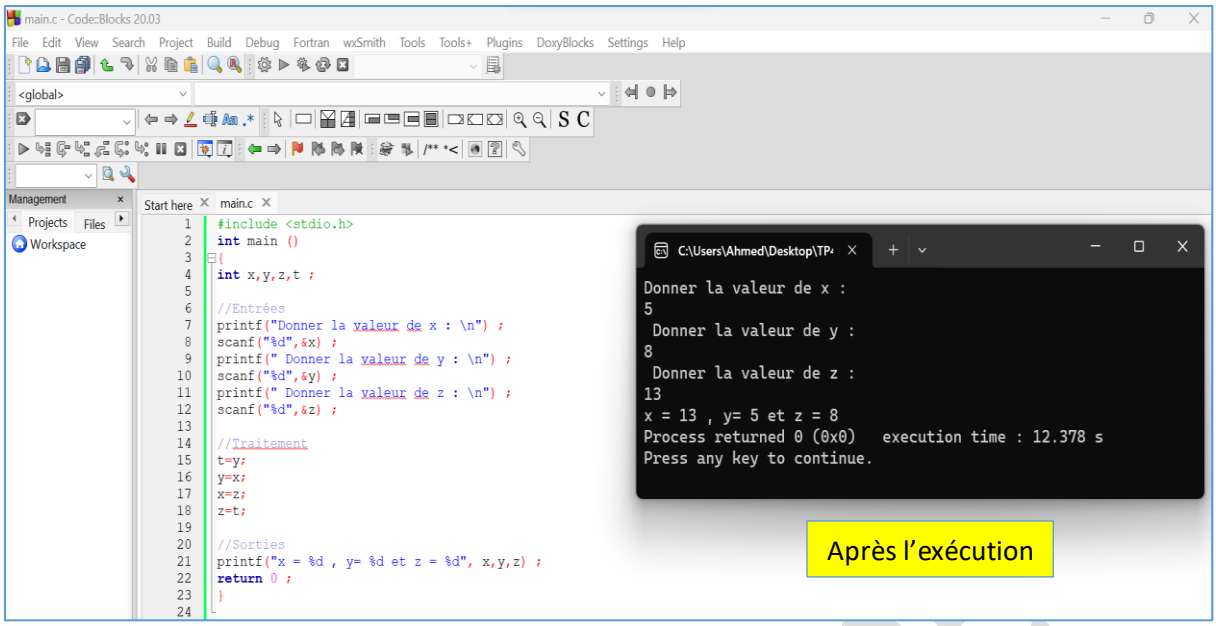
Algorithme Exo2_1;
Variables
x, y, t : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de x : ") ;
Lire(x) ;
Ecrire("Donner la valeur de y : ") ;
Lire(y) ;
 	
//Traitement
t ← x;
x ← y;
y ← t ;
 	
//Sorties
Écrire("x=", x, "y=", y) ;
Fin.
	#include <stdio.h>
	int main ()
	{
	int x,y,t ;
	
	//Entrées
	printf("Donner la valeur de x : \n") ;
	scanf("%d",&x) ;
	printf(" Donner la valeur de y : \n") ;
	scanf("%d",&y) ;
	
	//Traitement
	t=x;
	x=y;
	y=t;
	
	//Sorties
	printf("x = %d et y= %d", x,y) ;
	return 0 ;
	}



2) Permuter entre les trois variables X, Y et Z de telle sorte que la valeur de X soit dans Y, celle de Y dans Z et la valeur de Z dans X ?

Algorithme	**Programme C**

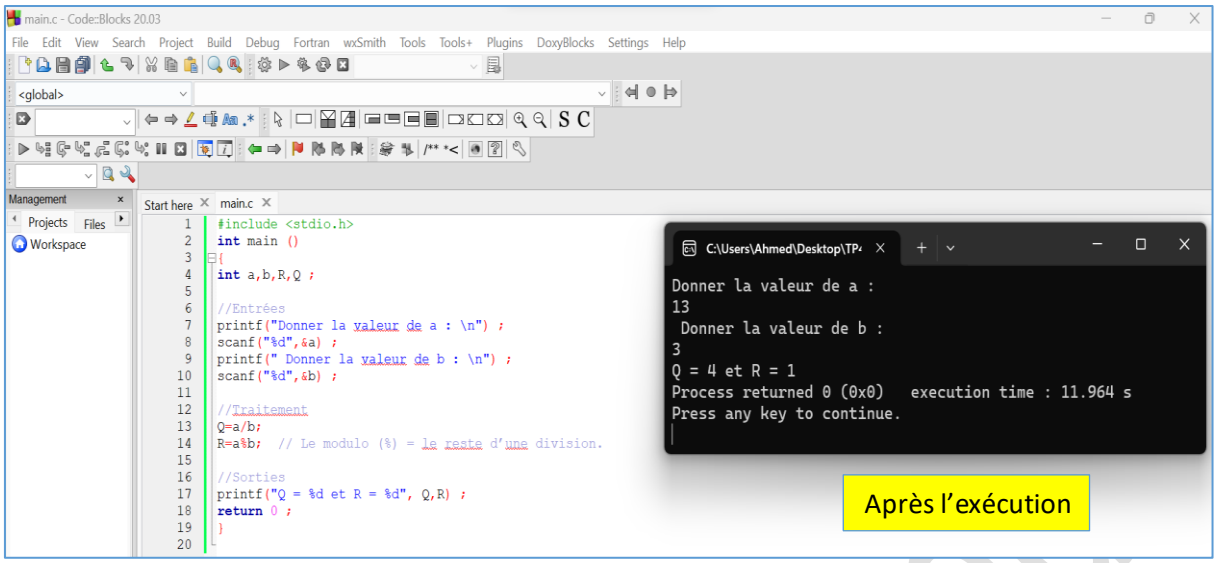
Algorithme Exo2_2;
Variables
x, y, z, t : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de x : ") ;
Lire(x) ;
Ecrire("Donner la valeur de y : ") ;
Lire(y) ;
Ecrire("Donner la valeur de z : ") ;
Lire(z) ;
 	
//Traitement
t ← y;
y ← x;
x ← z ;
z ← t ;
 	
//Sorties
Écrire("x=", x, "y=", y, "z=",z);
Fin.
	#include <stdio.h>
	int main ()
	{
	int x,y,z,t ;
	
	//Entrées
	printf("Donner la valeur de x : \n") ;
	scanf("%d",&x) ;
	printf(" Donner la valeur de y : \n") ;
	scanf("%d",&y) ;
	printf(" Donner la valeur de z : \n") ;
	scanf("%d",&z) ;
	
	//Traitement
	t=y;
	y=x;
	x=z;
	z=t;
	
	//Sorties
	printf("x = %d , y= %d et z = %d", x,y,z) ;
	return 0 ;
	}



3) Calculer le quotient et le reste de la division euclidienne de a par b ?

Algorithme	**Programme C**

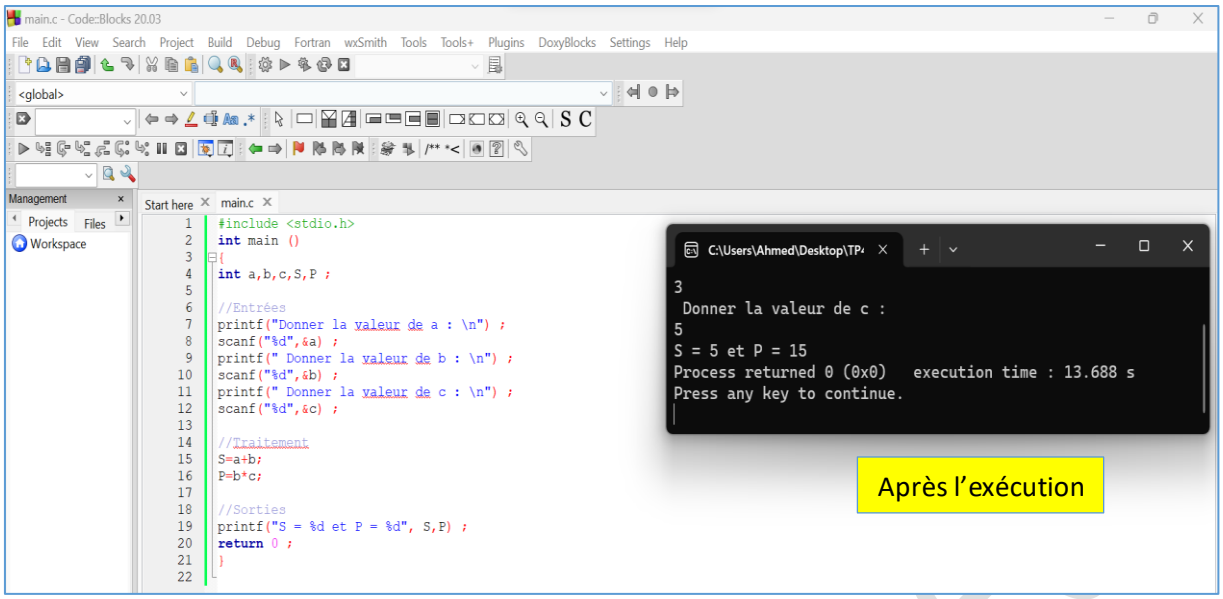
Algorithme Exo2_4;
Variables
a, b, Q, R : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de a : ") ;
Lire(a) ;
Ecrire("Donner la valeur de b : ") ;
Lire(b) ;
 	
//Traitement
Q ← a div b; //div : division entière
R ← a mod b; //mod : reste de division
 	
//Sorties
Écrire("Q=", Q, " et R= ", R) ;
Fin.
	#include <stdio.h>
	int main ()
	{
	int a,b,R,Q ;
	
	//Entrées
	printf("Donner la valeur de a : \n") ;
	scanf("%d",&a) ;
	printf(" Donner la valeur de b : \n") ;
	scanf("%d",&b) ;
	
	//Traitement
	Q=a/b;
	R=a%b; // Le modulo (%) = le reste d'une division.
	
	//Sorties
	printf("Q = %d et R = %d", Q,R) ;
	return 0 ;
	}



4) Calculer la somme de a et b et le produit de b et c ?

Algorithme	**Programme C**

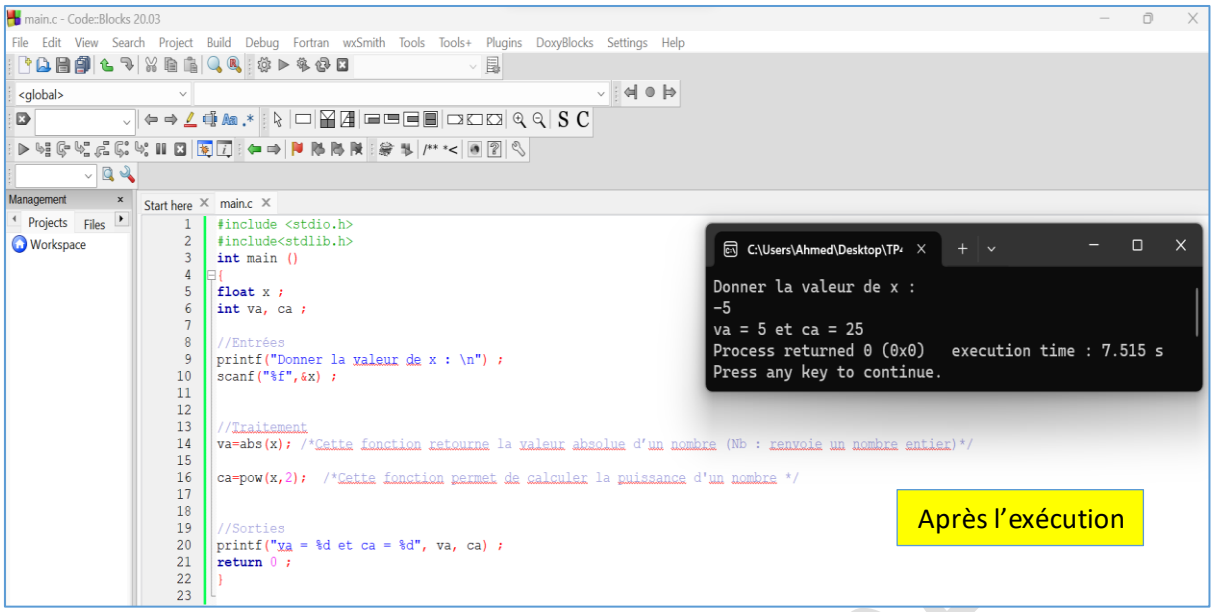
Algorithme Exo2_5;
Variables
a, b, c, S, P : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de a : ") ;
Lire(a) ;
Ecrire("Donner la valeur de b : ") ;
Lire(b) ;
Ecrire("Donner la valeur de c : ") ;
Lire(c) ;
 	
//Traitement
S ← a + b;
P ← b * c;
 	
//Sorties
Écrire("S=", S, "P= ", P) ;
Fin.
	#include <stdio.h>
	int main ()
	{
	int a,b,c,S,P ;
	
	//Entrées
	printf("Donner la valeur de a : \n") ;
	scanf("%d",&a) ;
	printf(" Donner la valeur de b : \n") ;
	scanf("%d",&b) ;
	printf(" Donner la valeur de c : \n") ;
	scanf("%d",&c) ;
	
	//Traitement
	S=a+b;
	P=b*c;
	
	//Sorties
	printf("S = %d et P = %d", S,P) ;
	return 0 ;
	}



5) Calculer la valeur absolue et le carré d'un nombre ?

Algorithme	**Programme C**

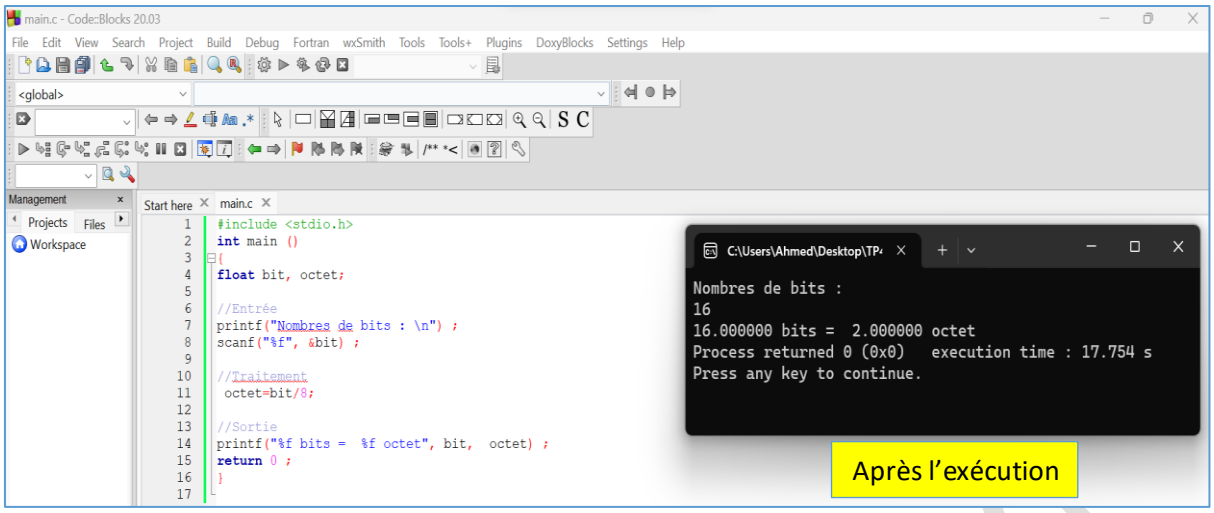
Algorithme Exo2_6;
Variables
x : **réel**;
va, ca : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de x : ") ;
Lire(x) ;
 	
//Traitement
va ← |x|;
ca ← x^2;
 	
//Sorties
Écrire("va=", va, "ca=", ca) ;
Fin.
	#include <stdio.h>
	#include<stdlib.h>
	int main ()
	{
	float x ;
	int va, ca ;
	
	//Entrées
	printf("Donner la valeur de x : \n") ;
	scanf("%f",&x) ;
	
	//Traitement
	va=abs(x); /*Cette fonction retourne la valeur absolue d'un nombre (**Nb :** renvoie un nombre entier)*/
	
	ca=pow(x,2); /*Cette fonction permet de calculer la puissance d'un nombre */
	
	//Sorties
	printf("va = %d et ca = %d", va, ca) ;
	return 0 ;
	}



6) Convertir en octets un nombre donné en bits ?

Algorithme	**Programme C**

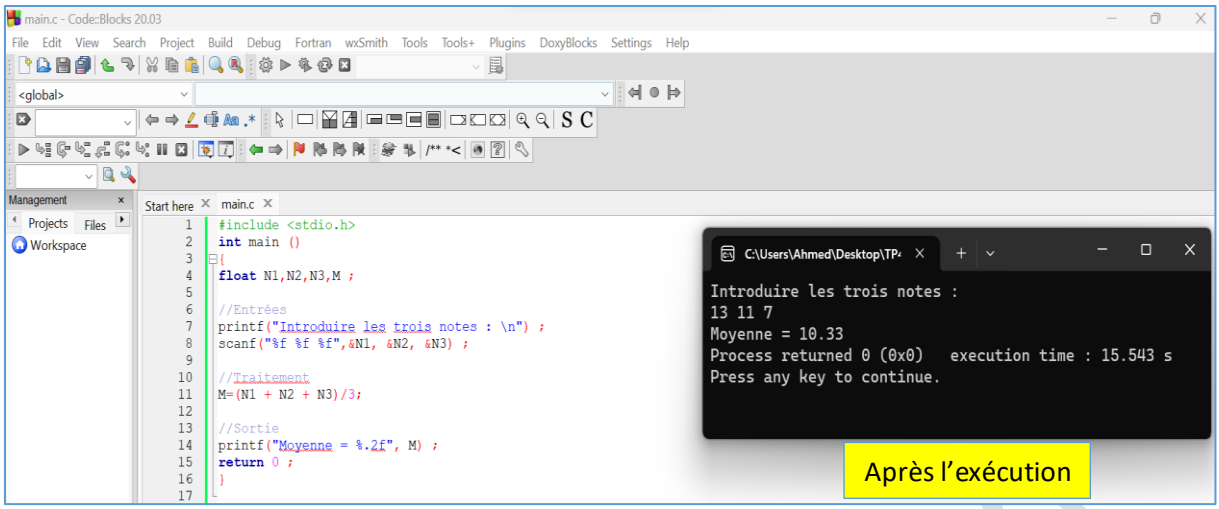
Algorithme Exo2_7;
Variables
bit, octet : **réel**;
 	
Début
//Entrée
Écrire("Nombres de bits =") ;
Lire(bit) ;
 	
//Traitement
octet ← bit/8;
 	
//Sortie
Écrire(bit, " bits =", octet, " octet");
Fin.
	#include <stdio.h>
	int main ()
	{
	float bit, octet;
	
	//Entrée
	printf("Nombres de bits : \n") ;
	scanf("%f", &bit) ;
	
	//Traitement
	octet=bit/8;
	
	//Sortie
	printf("%f bits = %f octet", bit, octet) ;
	return 0 ;
	}



7) Lire les notes de trois matières (N1, N2 et N3) ensuite calculer et afficher leur moyenne M

Algorithme	**Programme C**

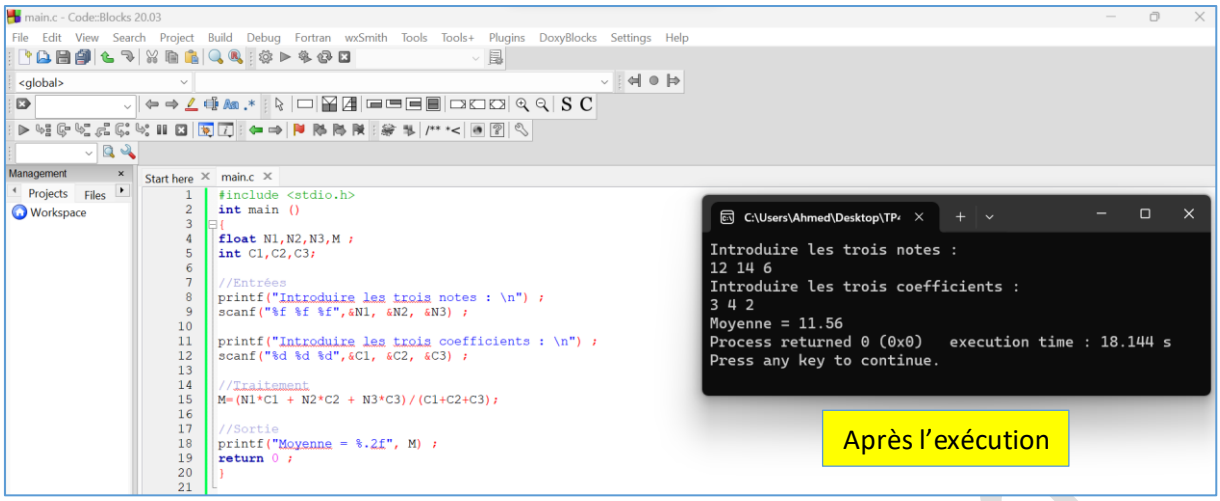
Algorithme Exo2_7_a;
Variables
N1, N2, N3, M : **réel** ;
 	
Début
//Entrées
Écrire ("Introduire les trois notes : ") ;
Lire(N1, N2, N3) ;
 	
//Traitement
M ← (N1 + N2 + N3)/3;
 	
//Sorties
Écrire("Moyenne =", M:0:2) ;
Fin.
	#include <stdio.h>
	int main ()
	{
	float N1,N2,N3,M ;
	
	//Entrées
	printf("Introduire les trois notes : \n") ;
	scanf("%f %f %f",&N1, &N2, &N3) ;
	
	//Traitement
	M=(N1 + N2 + N3)/3;
	
	//Sortie
	printf("Moyenne = %.2f", M) ;
	return 0 ;
	}



Modifier l'algorithme dans le cas où des coefficients (*C1*, *C2* et *C3*) sont attribués aux trois matières.

Algorithme	**Programme C**

Algorithme Exo2_7_b;
Variables
N1, N2, N3, M : **réel** ;
C1, C2, C3 : **entier** ;
Début
//Entrées
Écrire ("Introduire les trois notes : ") ;
Lire(N1, N2, N3) ;
 	
Écrire ("Introduire les trois coefficients : ") ;
Lire(C1, C2, C3) ;
 	
//Traitement
M ←(N1*C1 + N2*C2 + N3*C3)/(C1+C2+C3);
 	
//Sorties
Écrire("Moyenne =", M:0:2) ;
Fin.
	#include <stdio.h>
	int main ()
	{
	float N1,N2,N3,M ;
	int C1,C2,C3;
	
	//Entrées
	printf("Introduire les trois notes : \n") ;
	scanf("%f %f %f",&N1, &N2, &N3) ;
	
	printf("Introduire les trois coefficients : \n") ;
	scanf("%d %d %d",&C1, &C2, &C3) ;
	
	//Traitement
	M=(N1*C1 + N2*C2 + N3*C3)/(C1+C2+C3);
	
	//Sortie
	printf("Moyenne = %.2f", M) ;
	return 0 ;
	}



Exercice N°03 :

Rappel : Le périmètre **P** d'un cercle de rayon **R** est donné par la formule : **P = 2 * π * R**, où π est la constante mathématique pi (approximativement 3.14159).

Algorithme	**Programme C**

Algorithme Exo1;
Constante
pi=3.14 ;
Variables
R, P : **réels**;
 	
Début
//Les entrées
Écrire(" Donner le rayon R : ") ;
Lire(R) ;
 	
//Traitement
P ← 2*pi*R ;
 	
//Les sorties
Écrire("Le périmètre = ", P) ;
Fin.
	#include <stdio.h>
	int main()
	{
	const float pi=3.14;
	float R,P;
	
	//Les entrées
	printf("Donner le rayon R : \n");
	scanf("%f",&R);
	
	//Traitement
	P=2*pi*R;
	
	//Les sorties
	printf("Le périmètre = %.3f", P);
	
	return 0;
	}

Exercices supplémentaires sur la série de TP N°1

Exercice Sup-01 :

Écrire un algorithme puis la traduction en C d'un programme, qui calcule la surface d'un rectangle de dimensions données et affiche le résultat sous la forme suivante : "La surface du rectangle dont la longueur mesure .... m et la largeur mesure .... m, a une surface égale à .... mètres carrés".

Exercice Sup-02 :

Écrire un algorithme puis la traduction en C d'un programme qui lit une **température** en degrés Celsius et affiche son équivalent en Fahrenheit.

Exercice Sup-03 :

Exécuter les séquences d'instructions suivantes manuellement et donner les valeurs finales des variables A, B, C et celles de X, Y, Z.

a) A← 5; B← 3; C← B+A; A← 2; B← B+4; C← B-2

b) X← -5; Y← 2*X; X← X+1; Y← sqr(-X-Y); Z ← sqr(-X+Y); X← -(X+3*Y)+2

Écrire les algorithmes correspondants, puis les programmes en langage C correspondants, et fin, procéder à leur exécution.

Exercice Sup-04 :

Écrire un algorithme permettant d'effectuer une permutation circulaire de trois nombre entiers a, b et c.

Exemple : a=10, b=20 et c=30

Après permutation : a=30, b=10 et c=20

Exercice Sup-05 :

Écrire un programme en C qui effectue une permutation entre deux variables X et Y sans avoir utiliser une troisième variable.

Exercice Sup-06 :

Ecrire un algorithme puis la traduction en langage C d'un programme **Trapèze**, qui lit les dimensions d'un trapèze et affiche sa surface.

Série de TP N°2 : Les instructions de test : *Si…Fin-Si* & *Si…Sinon…Fin-Si*

But de TP :

Le but du TP est de permettre aux étudiants de comprendre et de maîtriser les structures de contrôle conditionnelles en programmation.

Exercice N°01 : *(Algorithme ➔ Programme en langage C)*

Soit l'algorithme suivant :

```
Algorithme exo1 ;
Variables
a, b, c, d: entier ;
Début
    // Entrées
    Écrire("Donner trois nombres
            entiers : ") ;
    Lire(a,b,c) ;
    // Traitements
    Si (a<b) alors
        Si (a<c) alors
            d ← a
        Sinon
            d ← c ;
        Fin-Si
    Sinon
        Si (b<c) alors
            d ← b
        Sinon
            d ← c ;
        Fin-Si
    Fin-Si
    // Sorties
    Écrire ("Le résultat = ", d) ;
Fin.
```

Questions :

1- Traduire l'algorithme en un programme en langage C.

2- Compiler et exécuter le programme pour les valeurs suivantes :
   - a=1, b=2, c=4
   - a=4, b=2, c=0
   - a=2, b= -1, c=4

3- Déduire ce que fait cet algorithme ?

4- Dérouler l'algorithme/Programme pour les cas suivants :
   - a=1, b=2, c=4
   - a=4, b=2 c=0
   - a=2, b= -1 et c=4

Exercice N°02 :

Écrire un algorithme/programme en langage C qui permet de lire deux nombres entiers (a, b), calcule et affiche la valeur de c comme suit :

$$c = \begin{cases} 0 & si\ a = b \\ a - b & si\ a > b \\ b - a & si\ a < b \end{cases}$$

Exercice N°03 :

Soit un service d'impression qui établit le prix d'impression d'une page selon le nombre de pages (nb_pages) :

a- Si nb_pages est inférieure ou égale à 10 : 5 D.A.
b- Si nb_pages est entre 11 et 20 : 4.5 D.A.
c- Si nb_pages est entre 21 et 60 : 3 D.A.
d- Si nb_pages est supérieure à 60 : 2.5 D.A.

Écrire un algorithme, puis le programme en langage C, qui permet de calculer le prix d'impression pour un nombre de pages quelconque.

Exercice N°04 :

Écrire un algorithme/programme en langage C qui permet d'afficher trois valeurs numériques A, B et C avec ordre croissant ?

Exercice N°05 :

Ecrire un programme C qui saisit un nombre entier et détecte si ce nombre est pair ou impair.

Corrigé de la série de TP N°2

Rappel :

Structures de contrôle conditionnelles

Ces structures sont utilisées pour déterminer l'exécution d'un bloc d'instructions : soit ce bloc est exécuté, soit il ne l'est pas. Elles servent également à choisir entre l'exécution de deux blocs différents. Nous avons deux types de structures conditionnelles :

1. Test alternatif simple

Un test simple contient un seul bloc d'instructions. Selon une condition (expression logique), on décide si le bloc d'instructions est exécuté ou non. Si la condition est vraie, on exécute le bloc, sinon on ne l'exécute pas.

La syntaxe d'un test alternatif simple est la suivante :

Algorithme	Langage C
si <Condition> **alors** <bloc_instructions_si> ; **finsi ;**	**if** (Condition) **{** <bloc_instructions_if> ; **}**

Remarque : Dans le langage C, un bloc est délimite par deux accolades **{** et **}**. Si le bloc contient une seule instruction, les accolades **{** et **}** sont facultatives (on peut les enlever).

2. Test alternatif double

Un test double contient deux blocs d'instructions : on est amené à choisir entre le premier bloc ou le second. Cette décision est réalisée sur une condition (expression logique ou

booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second.

La syntaxe d'un test alternatif simple est la suivante :

Algorithme	Langage C
si <Condition> **alors** <bloc_instructions_si> **sinon** <bloc_instructions_sinon> ; **finsi ;**	**if** (Condition) **{** <bloc_instructions_if> ; **}** **else** **{** <bloc_instructions_else> ; **}**

Remarques :

- En langage C, il ne faut pas mettre de point-virgule après la condition (erreur logique).
- Dans l'exemple précédent, les accolades {} du **if** et du **else** peuvent être omises puisqu'il y a une seule instruction dans les deux blocs.

Nous avons aussi, les **structures conditionnelles doubles et imbriquées** :

Un test double et imbriqué, tout comme un test double, contient deux blocs instructions avec au moins un des deux blocs (bloc Si et/ou bloc Sinon) est composé d'une instruction de condition simple ou double. Donc un test double et imbriqué contient au moins trois blocs d'instructions avec au moins deux conditions.

La syntaxe d'un test alternatif double imbriqué avec trois blocs d'instructions est :

Algorithme	Langage C

si <Condition> **alors**
  <bloc_instructions_si>
sinon
  si <Condition> **alors**
  <bloc_instructions_si>
  sinon
  <bloc_instructions_sinon> ;
  finsi ;
finsi ;
	if (Condition)
	{
	  <bloc_instructions_if> ;
	}
	else
	if (Condition)
	{
	  <bloc_instructions_if> ;
	}
	else
	{
	  <bloc_instructions_else> ;
	}

Exercice N°01 : *(Algorithme ➔ Programme en langage C)*

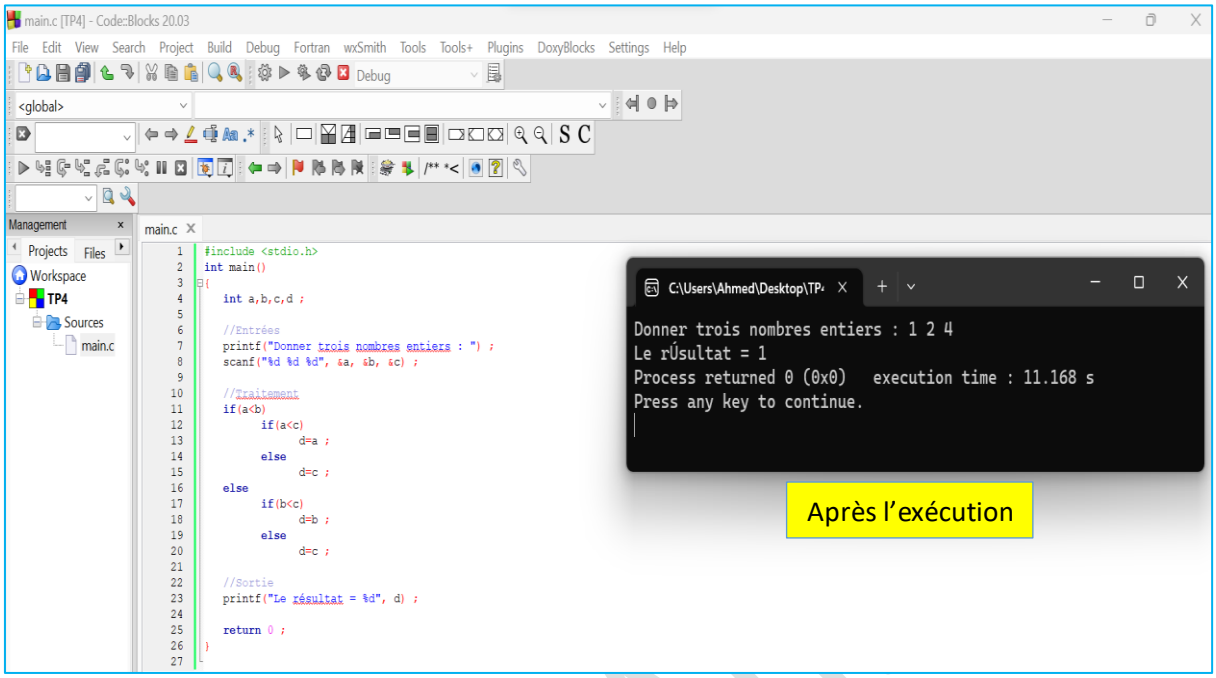
1) Traduire l'algorithme en un programme en langage C

Algorithme	**Programme C**

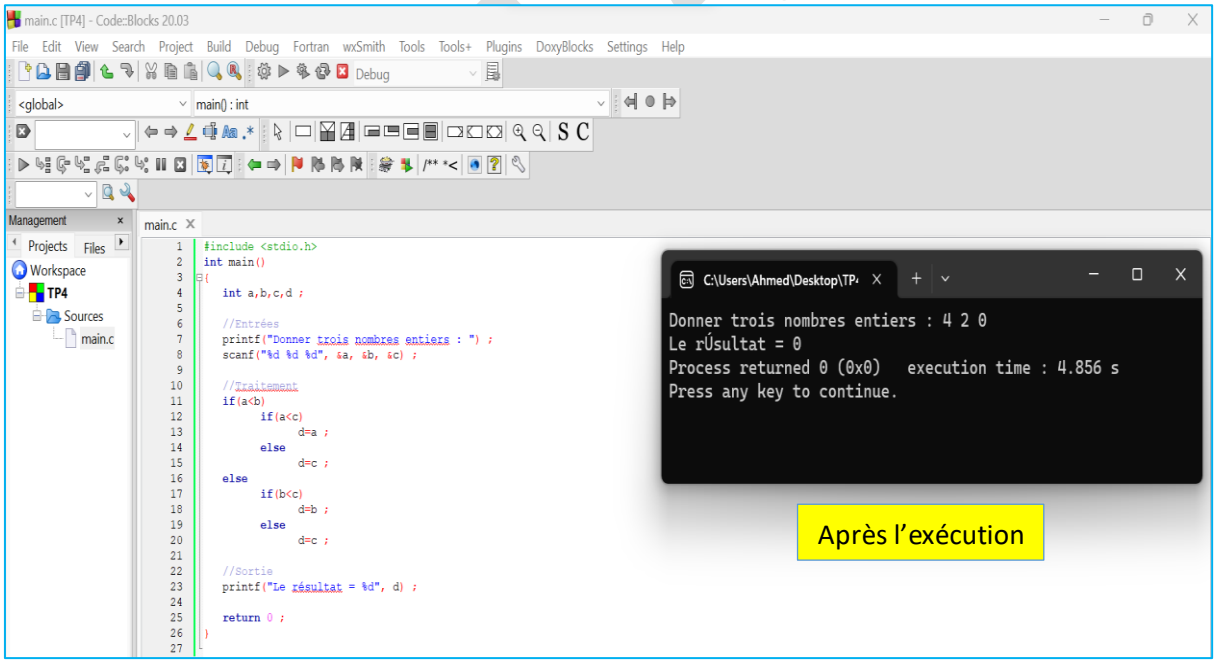
Algorithme exo1 ;
Variables
a, b, c, d : **entier** ;
 	
Début
//Entrées
Écrire ("Donner trois nombres entiers : ") ;
Lire (a,b,c) ;
 	
//Traitement
Si (a<b) **alors**
Si (a<c) **alors**
d ← a
Sinon
d ← c ;
Fin-Si
Sinon
Si (b<c) **alors**
d ← b
Sinon
d ← c ;
Fin-Si
Fin-Si
 	
//Sortie
Écrire ("Le résultat = ", d) ;
Fin.
	#include <stdio.h>
	int main()
	{
	int a,b,c,d ;
	
	//Entrées
	printf("Donner trois nombres entiers : ") ;
	scanf("%d %d %d", &a, &b, &c) ;
	
	//Traitement
	if(a<b)
	if(a<c)
	d=a ;
	else
	d=c ;
	else
	if(b<c)
	d=b ;
	else
	d=c ;
	
	//Sortie
	printf("Le résultat = %d", d) ;
	
	return 0 ;
	}

2) Compiler et exécuter le programme pour les valeurs suivantes :

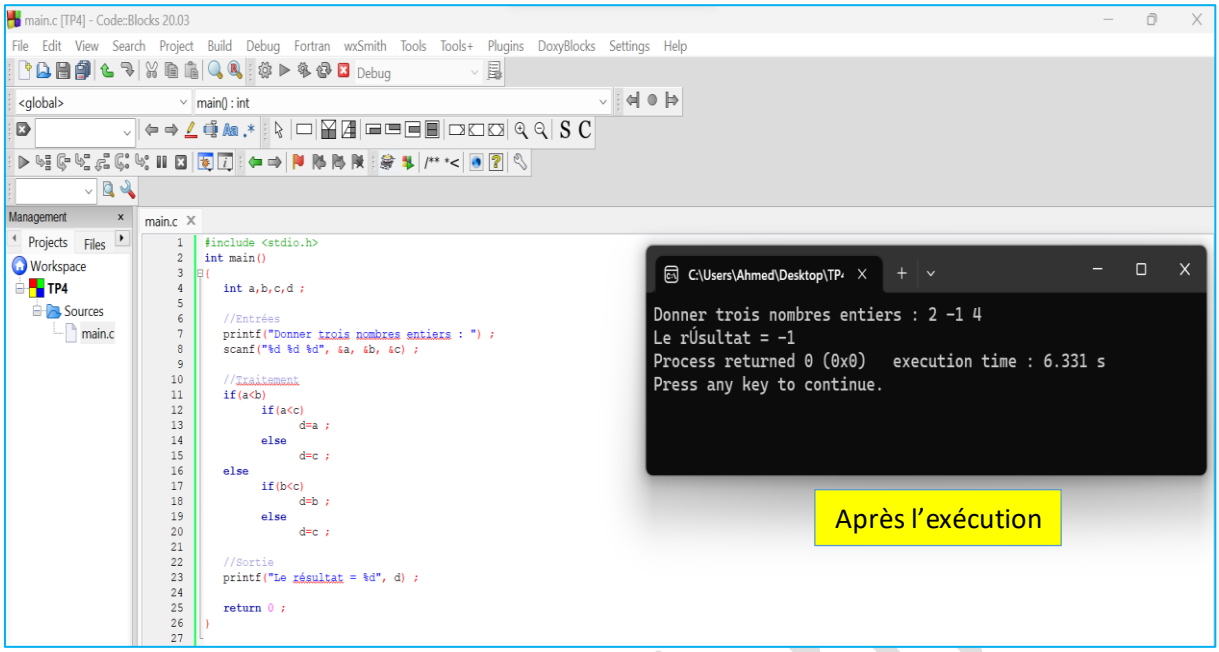
- **a=1, b=2, c=4**



- **a=4, b=2, c=0**



➢ **a=2, b= -1, c=4**



3) Déduire ce que fait cet algorithme ?

Le programme donne le plus petit des trois nombres entiers

4-1) Déroulement du programme pour a=1, b=2 et c=4

Instructions	Variables				Affichage
	a	b	c	d	
Écrire ("Donner trois nombres entiers : ") ;	/	/	/	/	Donner trois nombres entiers :
Lire (a,b,c) ;	1	2	4	/	
Si (a<b) **alors** **1 < 2** **True** ⇨ On exécute le bloc **Si** **Si** (a<c) **alors** **1 < 4** **True** ⇨ On exécute le bloc **Si** d ← a **d ← 1** **Fin-Si**	1	2	4	**1**	
Écrire ("Le résultat = ", d) ;	1	2	4	**1**	Le résultat = **1**

4-2) Déroulement du programme pour a=4, b=2 et c=0

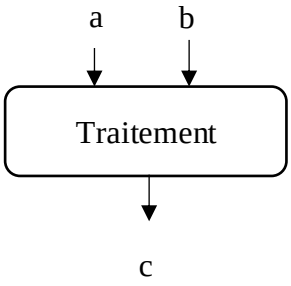
Instructions	Variables				Affichage
	a	b	c	d	
Écrire ("Donner trois nombres entiers : ") ;	/	/	/	/	Donner trois nombres entiers :
Lire (a,b,c) ;	4	2	0	/	
Si (a<b) **alors** **4 < 2** **False** ⇨ On n'exécute pas le bloc **Si** (on passe au bloc **Sinon**) **Si** (b<c) **alors** **4 < 0** **False** ⇨ On n'exécute pas le bloc **Si** (on passe au bloc **Sinon**) d ← c ; **d ← 0** **Fin-Si**	4	2	0	**0**	
Écrire ("Le résultat = ", d) ;	4	2	0	**0**	Le résultat = **0**

4-3) Déroulement du programme pour a=2, b=-1 et c=4

Instructions	Variables				Affichage
	a	b	c	d	
Écrire ("Donner trois nombres entiers : ") ;	/	/	/	/	Donner trois nombres entiers :
Lire (a,b,c) ;	2	-1	4	/	
Si (a<b) **alors** **2 < -1** **False** ⇨ On n'exécute pas le bloc **Si** (on passe au bloc **Sinon**) **Si** (b<c) **alors** **-1 < 4** **True** ⇨ On n'exécute le bloc **Si** d ← b ; **d ← -1** **Fin-Si**	2	-1	4	**-1**	
Écrire ("Le résultat = ", d) ;	2	-1	4	**-1**	Le résultat = **-1**

Exercice N°02 :

Le schéma entrées, traitement et sorties.



Entrées :

Lire (a,b) ;

Traitements :

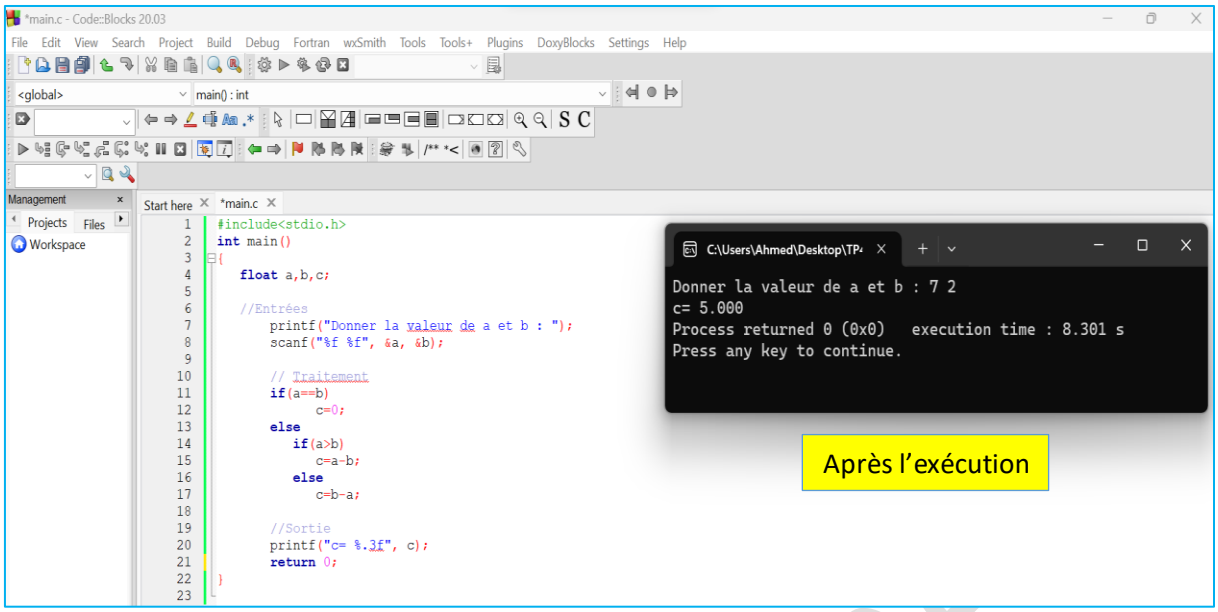
Si a=b **alors**
c ← 0;
Sinon
Si a>b **alors**
c ← a-b;
Sinon
c ← b-a;
Fin-Si
Fin-Si

Sortie :

Écrire (c) ;

Algorithme	**Programme C**

Algorithme exo2;
Variables
a, b, c : **réel**;
Début
//Entrées
Ecrire("Donner la valeur de a et b : ");
Lire (a,b);
 	
//Traitement
Si (a=b) **alors**
c ← 0;
Sinon
Si (a>b) **alors**
c ← a-b;
Sinon
c ← b-a;
Fin-Si
Fin-Si
 	
//Sortie
Écrire ("c = ", c)
Fin.
	#include<stdio.h>
	int main()
	{
	float a,b,c;
	
	//Entrées
	printf("Donner la valeur de a et b : ");
	scanf("%f %f", &a, &b);
	
	// Traitement
	if(a==b)
	c=0;
	else
	if(a>b)
	c=a-b;
	else
	c=b-a;
	
	//Sortie
	printf("c= %.3f", c);
	return 0;
	
	}

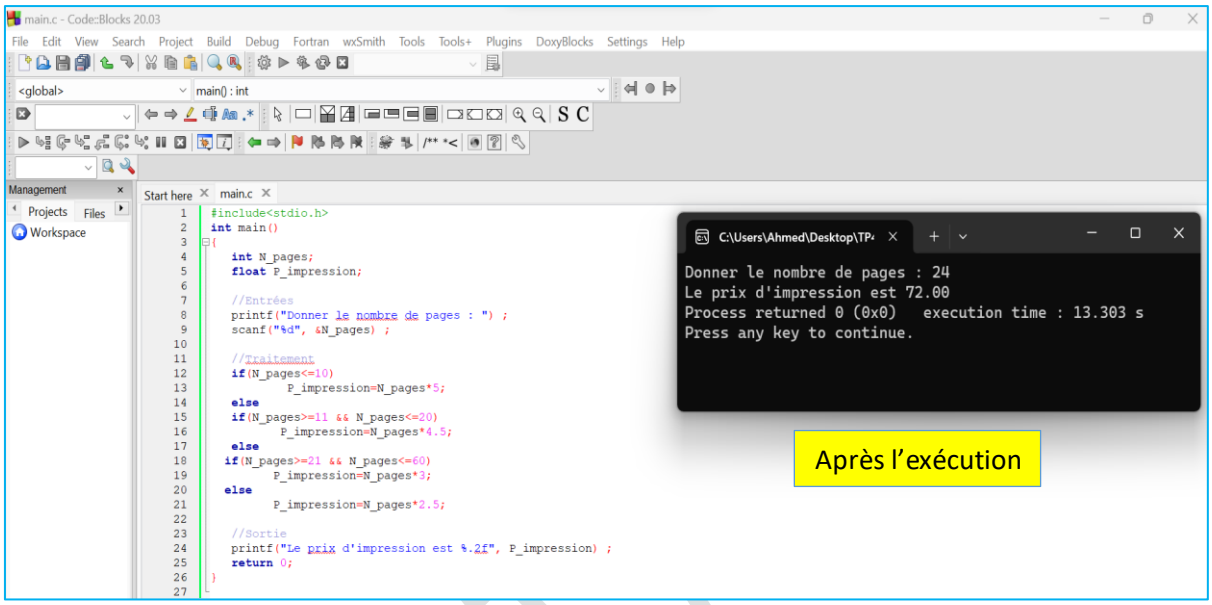


Exercice N°03 :

Algorithme	**Programme C**

Algorithme Prix_impression;
Variables
N_pages : **entier**;
P_impression : **réel**;
 	
Début
//Entrées
Écrire ("Donner le nombre de pages : ") ;
Lire (N_pages) ;
 	
//Traitement
 	
Si (N_pages <= 10) **alors**
P_impression ← N_pages*5 ;
Sinon
Si (N_pages >= 11 **ET** N_pages<=20) **alors**
P_impression ← N_pages*4.5 ;
Sinon
Si (N_pages >= 21 **ET** N_pages<=60) **alors**
P_impression ← N_pages*3 ;
Sinon
P_impression ← N_pages*2.5 ;
Fin-Si ;
Fin-Si ;
	#include<stdio.h>
	int main()
	{
	int N_pages;
	float P_impression;
	
	//Entrées
	printf("Donner le nombre de pages : ") ;
	scanf("%d", &N_pages) ;
	
	//Traitement
	if(N_pages<=10)
	P_impression=N_pages*5;
	else
	if(N_pages>=11 && N_pages<=20)
	P_impression=N_pages*4.5;
	else
	if(N_pages>=21 && N_pages<=60)
	P_impression=N_pages*3;
	else
	P_impression=N_pages*2.5;
	
	//Sortie
	printf("Le prix d'impression est %.2f", P_impression) ;

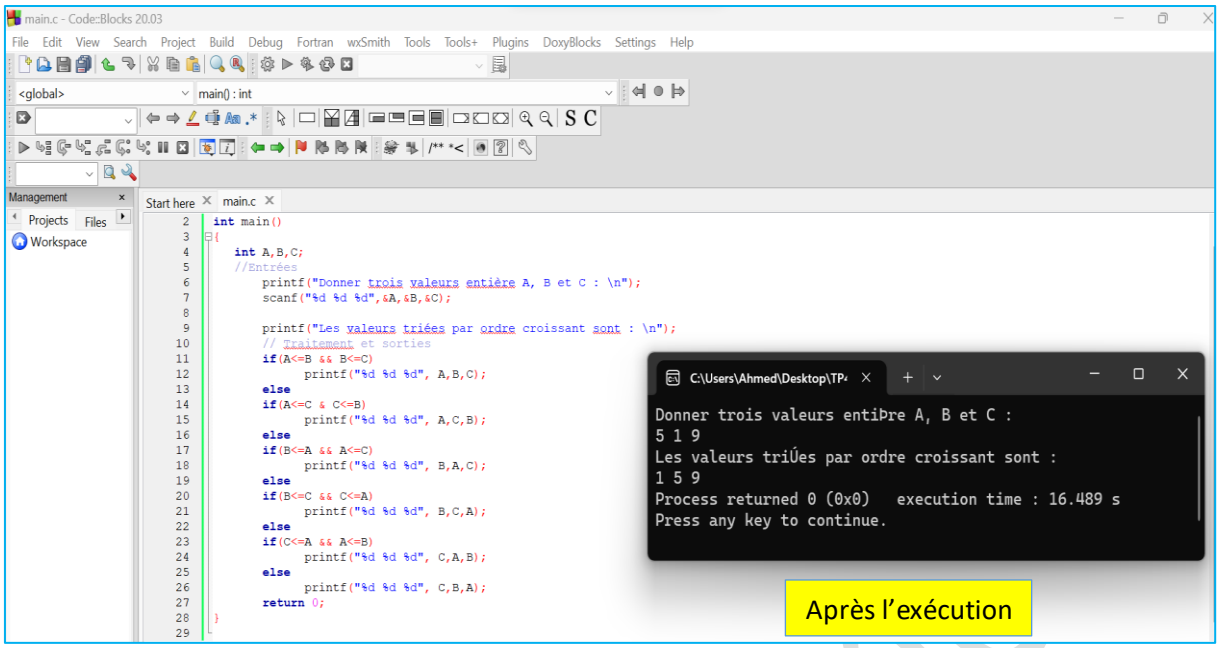
Fin-Si ; //Sortie **Écrire**("Le prix d'impression est " , P_impression); **Fin.**	**return 0;** **}**



Exercice N°04 :

Algorithme	Programme C

Algorithme ordre_croissant;
Variables
A, B, C : **entier**;
Début
//Entrées
Écrire("Donner trois valeurs entière A, B et C : ");
Read(A, B, C) ;
//Traitement & Sorties
Écrire("Les valeurs triées par ordre croissant sont : ");
Si (A <= B **ET** B <= C) alors
Écrire(A, B, C);
Sinon
Si (A <= C **ET** C <= B) **alors**
Écrire (A, C, B) ;
Sinon
Si (B <= A **ET** A <= C) **alors**
Écrire(B, A, C)
Sinon
Si (B <= C **ET** C<=A) **alors**
Écrire(B, C, A)
Sinon
Si (C <= A **ET** A<=B) **alors**
Écrire(C, A, B);
Sinon
Écrire(C, B, A);
Fin-Si ;
Fin-Si ;
Fin-Si ;
Fin-Si ;
Fin-Si ;
Fin.
	#include<stdio.h>
	int main()
	{
	int A,B,C;
	//Entrées
	printf("Donner trois valeurs entière A, B et C : \n");
	scanf("%d %d %d",&A,&B,&C);
	// Traitement et sorties
	printf("Les valeurs triées par ordre croissant sont : \n");
	if(A<=B && B<=C)
	printf("%d %d %d", A,B,C);
	else
	if(A<=C & C<=B)
	printf("%d %d %d", A,C,B);
	else
	if(B<=A && A<=C)
	printf("%d %d %d", B,A,C);
	else
	if(B<=C && C<=A)
	printf("%d %d %d", B,C,A);
	else
	if(C<=A && A<=B)
	printf("%d %d %d", C,A,B);
	else
	printf("%d %d %d", C,B,A);
	return 0;
	}



N.B : Il existe d'autres solutions qui permettent d'afficher trois valeurs numériques A, B et C avec ordre croissant.

Exercice N°05 :

Algorithme	Programme C

Algorithme Partie ;
Variables
N, R : **entier**;
 	
Début
//Les entrées
Écrire ("Donner un entier : ") ;
Lire (N) ;
 	
//Traitement
R ← N mod 2;
 	
//Les sorties
Si (R = 0) **alors**
Écrire (N, " est pair")
Sinon
Écrire (N, " est impair") ;
Fin-Si ;
Fin.
	#include <stdio.h>
	
	int main()
	{
	int N, R;
	
	//Les entrées
	printf("Donner un entier : ");
	scanf("%d", &N);
	
	//Traitement
	R= N % 2 ;
	
	//Les sorties
	if(R==0)
	printf("%d est pair", N) ;
	else
	printf("%d est impair", N) ;
	
	return 0;
	}

Exercices supplémentaires sur la série de TP N°2

Exercice Sup-01 :

Écrire un algorithme/programme Pascal qui permet d'introduire l'IMC (Indice de Masse Corporelle) d'une personne et d'afficher des informations concernant la catégorie de son IMC comme suit :

« **Sous-poids »** Si IMC < 18.5
« Normal » Si 18.5 ≤ IMC ≤ 27.0
« Sur-poids « Si 27.0 < IMC < 32.0

Exercice Sup-02 :

Écrire un programme en langage C qui permet de résoudre l'équation du second degré $ax^2 + bx + c = 0$

Exercice Sup-03 :

On demande d'écrire l'algorithme d'une fiche de paie journalière d'un ouvrier rémunéré à la tâche. Pour cela, on donne :

- La valeur de cette rémunération par pièces réalisées VP,
- Le salaire brut (SB) est calculé selon le nombre de pièces correctes réalisées pendant la journée (NPC) comme suit :

  Si NPC ≤ 100, l'ouvrier touche NPC*VP
  Si NPC > 100, l'ouvrier touche 150*VP
- On enlève à la fin 10% du salaire pour les charges sociales (CS).

Calculer et afficher le salaire journalier brut (SB), les charges sociales (CS) et salaire journalier net (SN).

NB : Salaire brut=salaire totale ; Salaire net=salaire sans les charges sociales.

Exercice Sup-04 :

Écrire un algorithme et sa traduction en programme C qui permet de calculer et afficher le nombre de **centaines, dizaines et unités** constituants **un nombre entier « nb »** (0 < nb <1000)

Exemple : nb = 385, nb est constitué de 3 centaines, 8 dizaines et 5 unités
nb = 93, nb est constitué de 9 dizaines et 3 unités
nb = 4, nb est constitué de 4 unités

Exercice Sup-05 :

Ecrire un programme en langage C permettant de lire la valeur de la température de l'eau et d'afficher son état :

« Glace» Si la température <=0,
« Liquide» Si 0 < la température <100,
« Vapeur» Si la température >=100.

Série de TP N°3 : Les boucles : *Pour – Tant-que - Répéter*

Exercice N°01 : *(Algorithme ➔ Programme en langage C)*

Soit l'algorithme suivant :

```
Algorithme Exo1 ;
Variables
   X, P, S : réel ;
   i, N : entier ;
Début

   // Entrées

   Ecrire ('Donner les valeurs de N
            et X : ') ;
   Lire (N, X) ;

   // Traitement

   S ← 0 ;
   P ← X ;
   Pour i ←1 à N faire

      S ← S+ P/i ;
      P ← P*X ;

   Fin-Pour

   // Sortie

   Ecrire ('S= ', S) ;
Fin.
```

Questions :

1- Traduire l'algorithme en un programme en langage C.
2- Compiler et exécuter le programme pour N=3 et X=3.
3- Dérouler le programme pour N=3 et X=3.
4- Déduire l'expression générale du résultat S en fonction de X et N ?
5- Ré-écrire l'algorithme/Programme en remplaçant la boucle **Pour** par la boucle **Tant-que**.
6- Ré-écrire l'algorithme/Programme en remplaçant la boucle **Pour** par la boucle **Répéter**.
7- Modifier l'algorithme pour calculer la somme S2 :

$$S2 = X + \frac{X^3}{2} + \frac{X^5}{3} + \frac{X^7}{4} + \cdots + N^{ème}\ \text{terme}$$

Exercice N°02 : *(Enoncé du problème ➔ Algorithme ➔ Programme en langage C)*

Ecrire un algorithme/programme C pour calculer chacune des sommes / produits suivants :

1) Calculer la somme $S = 1^2 + 3^2 + 5^2 + \cdots + (2N + 1)^2$
2) Calculer le produit $P = 1 \times 3 \times \ldots \times N$
3) Calculer la somme $S = x + \frac{x^3}{2} + \frac{x^5}{4!} + \frac{x^7}{6!} + \cdots + (N^{ème}\ terme)$
4) Calculer la somme $S = x - x^2 + x^3 - \cdots \mp x^N$

Exercice N°03 :

Soit A et B deux entier tel-que A < B. Introduire N valeurs entières entre A et B, et réaliser la somme de valeurs pairs non-nuls et le produit des valeurs impaires.

Corrigé de la série de TP N°3

Rappel :

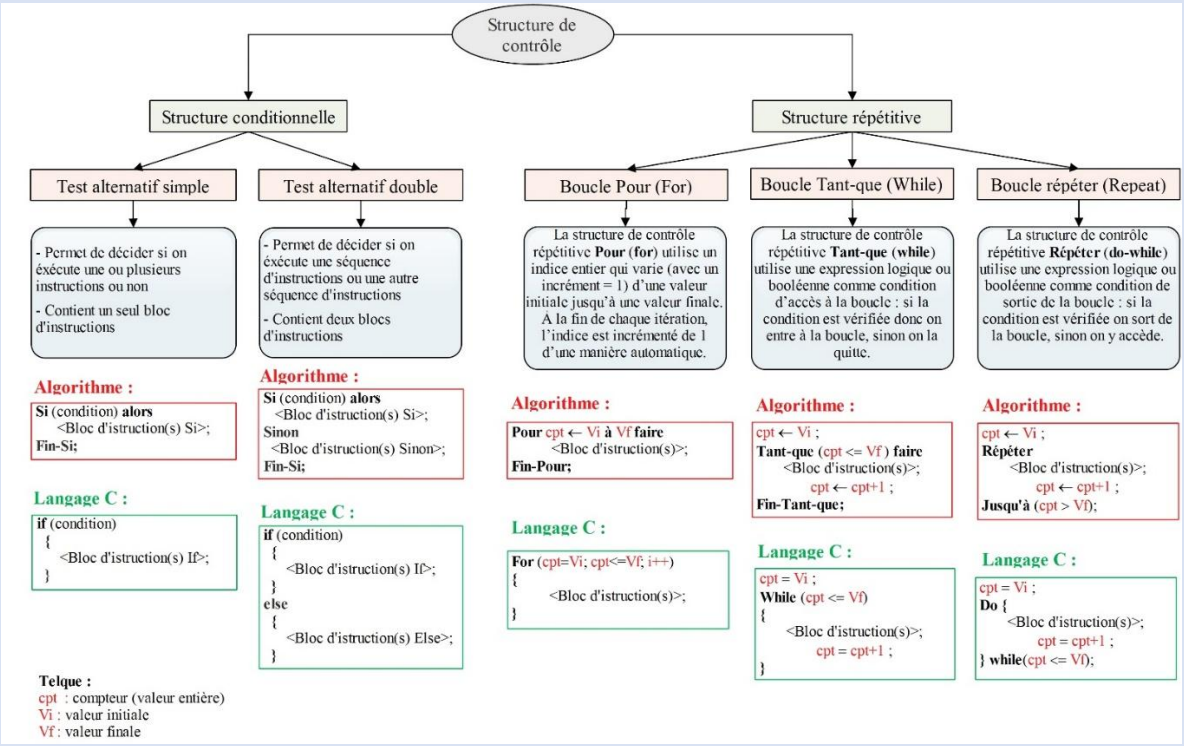
Structures de contrôle répétitives

Les structures répétitives nous permettent de répéter un traitement un nombre fini de fois. *Par exemple*, on veut afficher tous les nombres premiers entre 1 et N (N est un nombre entier positif donné).

Nous avons trois types de structures itératives (boucles) :

1. **Boucle pour (For)**
2. **Boucle tant-que (while)**
3. **Boucle répéter (do-while)**

Les syntaxes des trois boucles sont illustrées dans la figure ci-dessous :



La différence entre la boucle **répéter** et la boucle **tant-que** est :

- La condition de **répéter** et toujours l'inverse de la condition **tant-que** : pour **répéter** c'est la condition de sortie de la boucle, et pour **tant-que** c'est la condition d'entrer.
- Le teste de la condition est à la fin de la boucle (la fin de l'itération) pour **répéter**. Par contre, il est au début de l'itération pour la boucle **tant-que**. C'est-à-dire, dans **tant-que** on teste la condition avant d'entrer à l'itération, et dans **répéter** on fait l'itération après on teste la condition.

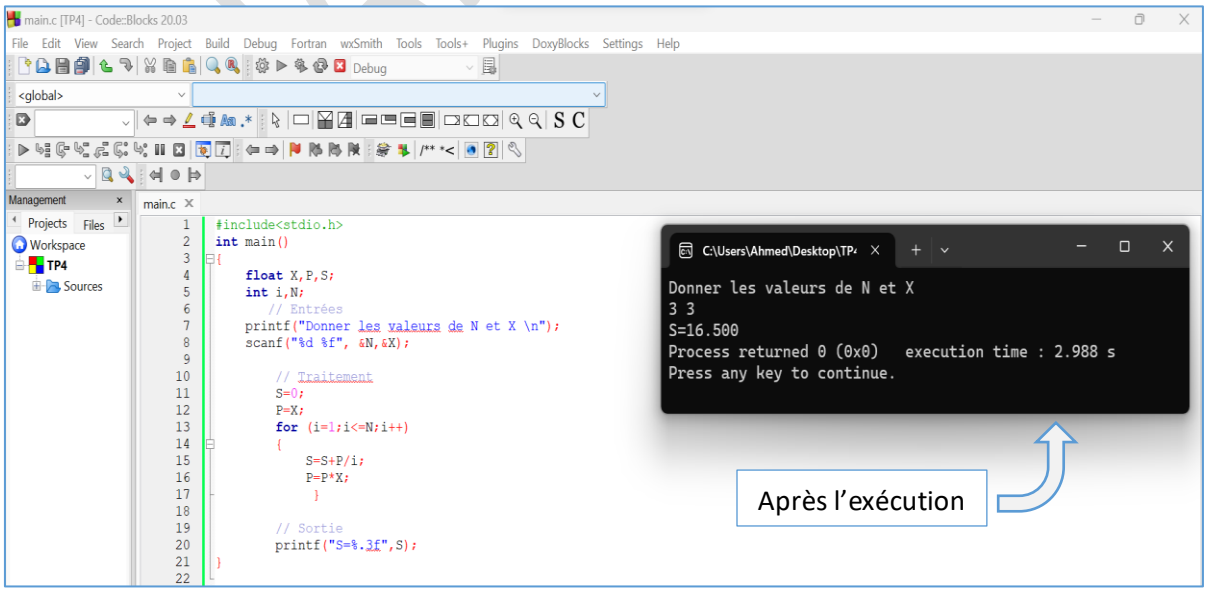
Exercice N°01 : *(Algorithme ➔ Programme en langage C)*

1) Traduire l'algorithme en Programme C

Algorithme	Programme C

Algorithme Exo1 ;
Variables
X, P, S : **réel** ;
i, N : **entier** ;
Début
 	
// Entrées
Ecrire ("Donner les valeurs de N et X : ") ;
Lire (N, X) ;
 	
// Traitement
S ← 0 ;
P ← X ;
Pour i ←1 **à** N **faire**
S ← S+ P/i ;
P ← P*X ;
Fin-Pour
 	
// Sortie
Ecrire ("S = ", S) ;
Fin.
	#include<stdio.h>
	int main()
	{
	float X,P,S;
	int i,N;
	
	// Entrées
	printf("Donner les valeurs de N et X \n");
	scanf("%d %f", &N,&X);
	
	// Traitement
	S=0;
	P=X;
	for (i=1;i<=N;i++)
	{
	S=S+P/i;
	P=P*X;
	}
	
	// Sortie
	printf("S=%.3f",S);
	}

2) Compiler et exécuter le programme pour N=3, X=3.



3) Dérouler le programme pour N=3, X=3.

Instructions	Variables					Affichage
	N	X	i	P	S	
Ecrire ("Donner les valeurs de N et X : ") ;						Donner les valeurs de N et X :
Lire (N,X);	3	3	/	/	/	/
S ← 0;	3	3	/	/	0	/
P ← X; P ← 3	3	3	/	3	0	/
Pour i ←1 S ← S+ P/i ; S ← 0+ 3/1 ; S ← 3 ; P ← P*X ; P ← 3*3 ; P ← 9 ;	3	3	1	**9**	**3**	
Pour i ← 2 S ← S+ P/i ; S ← 0+ 3/1+ 9/2 ; S ← **7.5** ; P ← P*X ; P ← 3*3*3 ; P ← 27 ;	3	3	1	**27**	**7.5**	
Pour i ← 3 S← S+ P/i ; S ← 0+ 3/1+ 9/2 + 27/3 ; S← **16.5**; P ← P*X ; P ← 3*3*3*3 ; P ← 81 ;	3	3	1	**81**	**16.5**	
Ecrire ('S = ', S) ;	3	3	1	**81**	**16.5**	S = **16.500**

4) Déduire l'expression générale du résultat S en fonction de X et N ?

Selon le déroulement ci-dessus, nous avons :

Pour i = 1, nous avons S = 3

Pour i = 2, nous avons $S = 3 + \frac{9}{2} = 7.5$

Pour i = 3, nous avons $S = 3 + \frac{9}{2} + \frac{27}{3} = 16.5$

Pour i = N nous aurons : $S = X + \frac{X^2}{2} + \frac{X^3}{3} + \ldots\ldots + \frac{X^N}{N}$

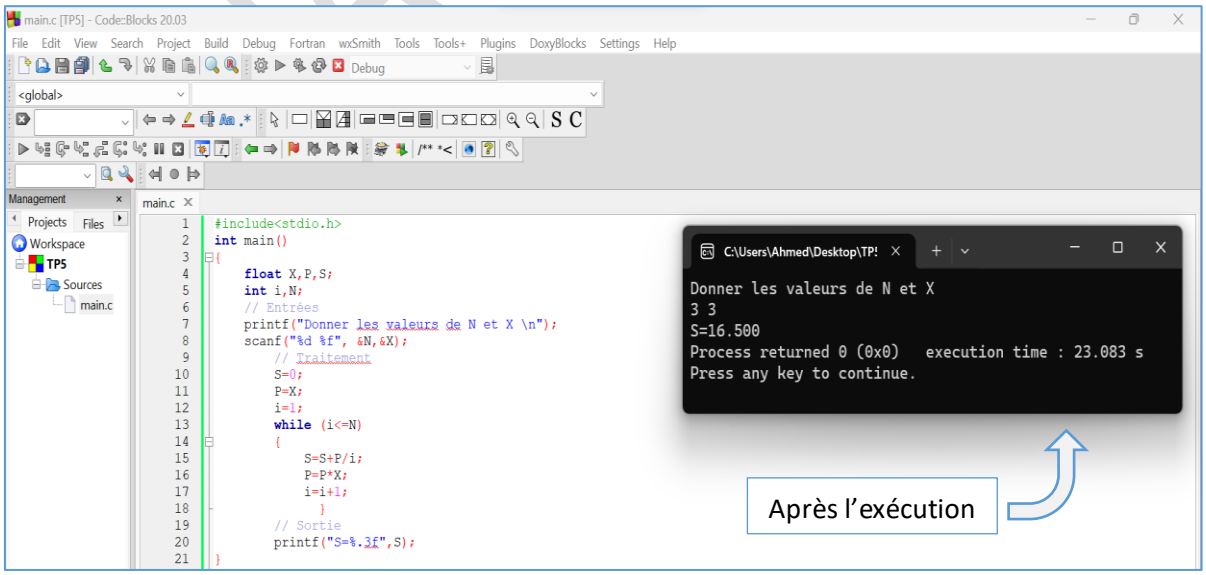
On peut généraliser par la formule suivante :

$$S = \sum_{i=1}^{N} \frac{X^i}{i} \quad ou \quad S = X + \frac{X^2}{2} + \frac{X^3}{3} + \cdots + \frac{X^N}{N}$$

5) Réécrire l'algorithme en remplaçant la boucle *Pour* par la boucle *Tant-que*.

Algorithme	**Programme C**

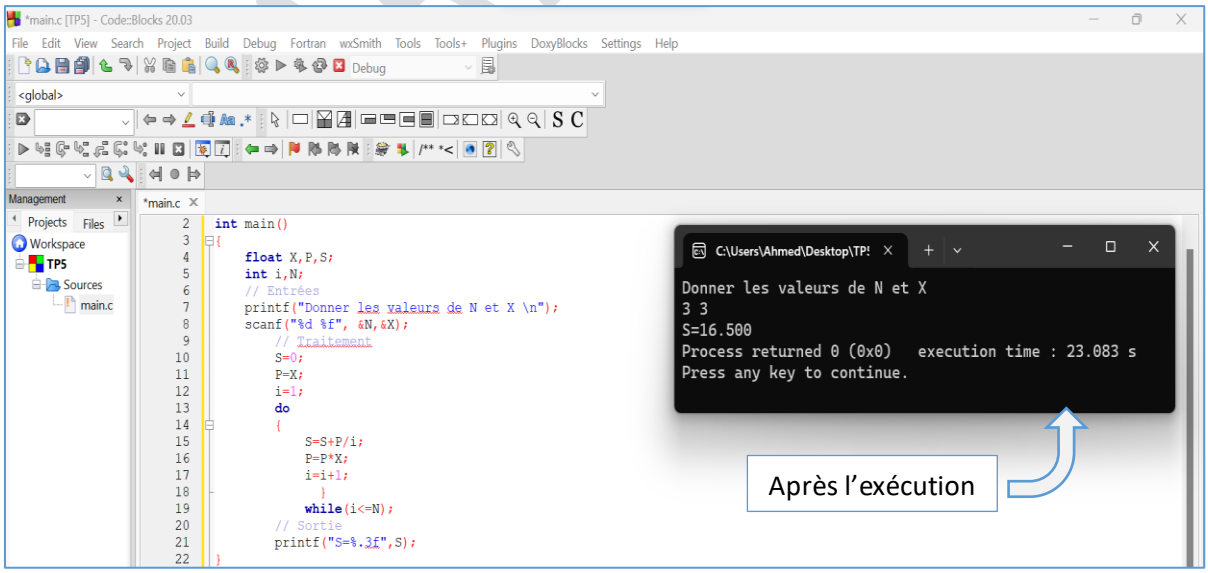
Algorithme Exo1 ;
Variables
X, P, S : **réel** ;
i, N : **entier** ;
Début
 	
// Entrées
Ecrire ("Donner les valeurs de N et X : ") ;
Lire (N, X) ;
 	
// Traitement
S ← 0 ;
P ← X ;
i ← 1 ;
Tant-que i ≤ N **faire**
S ← S+ P/i ;
P ← P*X ;
i ← i+1 ;
Fin Tant-que
 	
// Sortie
Ecrire ("S =", S) ;
Fin.
	#include<stdio.h>
	int main()
	{
	float X,P,S;
	int i,N;
	
	// Entrées
	printf("Donner les valeurs de N et X \n");
	scanf("%d %f", &N,&X);
	
	// Traitement
	S=0;
	P=X;
	i=1;
	while (i<=N)
	{
	S=S+P/i;
	P=P*X;
	i=i+1;
	}
	
	// Sortie
	printf("S=%.3f",S);
	}



6) Réécrire l'algorithme en remplaçant la boucle *Pour* par la boucle *Répéter*.

Algorithme	Programme C

Algorithme Exo1 ;
Variables
X, P, S : **réel** ;
i, N : **entier** ;
Début
 	
// Entrées
Ecrire ("Donner les valeurs de N et X : ") ;
Lire (N, X) ;
 	
// Traitement
S ← 0 ;
P ← X ;
i ← 1 ;
Répéter
S ← S+ P/i ;
P ← P*X ;
i ← i+1 ;
Jusqu'a i>N;
 	
// Sortie
Ecrire ("S =", S) ;
Fin.
	#include<stdio.h>
	int main()
	{
	float X,P,S;
	int i,N;
	
	// Entrées
	printf("Donner les valeurs de N et X \n");
	scanf("%d %f", &N,&X);
	
	// Traitement
	S=0;
	P=X;
	i=1;
	do
	{
	S=S+P/i;
	P=P*X;
	i=i+1;
	}
	while(i<=N);
	
	// Sortie
	printf("S=%.3f",S);
	}

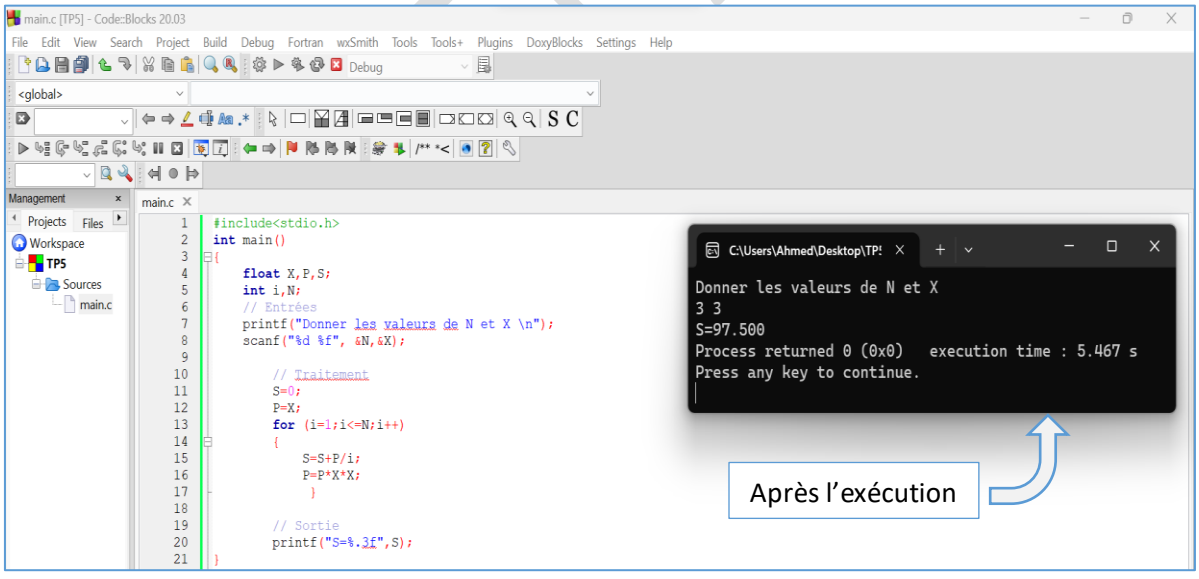


7) Modifier l'algorithme pour calculer la somme S2 :

$$S2 = X + \frac{X^3}{2} + \frac{X^5}{3} + \frac{X^7}{4} + \cdots + N^{ème}\ \text{terme}$$

Algorithme	**Programme C**

Algorithme Exo2 ;
Variables
X, P, S : **réel** ;
i, N : **entier** ;
Début
 	
// Entrées
Ecrire ("Donner les valeurs de N et X : ") ;
Lire (N, X) ;
 	
// Traitement
S ← 0 ;
P ← X ;
Pour i ←1 **à** N **faire**
S ← S+ P/i ;
P ← P*X*X ;
Fin-Pour;
 	
// Sortie
Ecrire ("S =", S) ;
Fin.
	#include<stdio.h>
	int main()
	{
	float X,P,S;
	int i,N;
	
	// Entrées
	printf("Donner les valeurs de N et X \n");
	scanf("%d %f", &N,&X);
	
	// Traitement
	S=0;
	P=X;
	for (i=1;i<=N;i++)
	{
	S=S+P/i;
	P=P*X*X;
	}
	
	// Sortie
	printf("S=%.3f",S);
	}

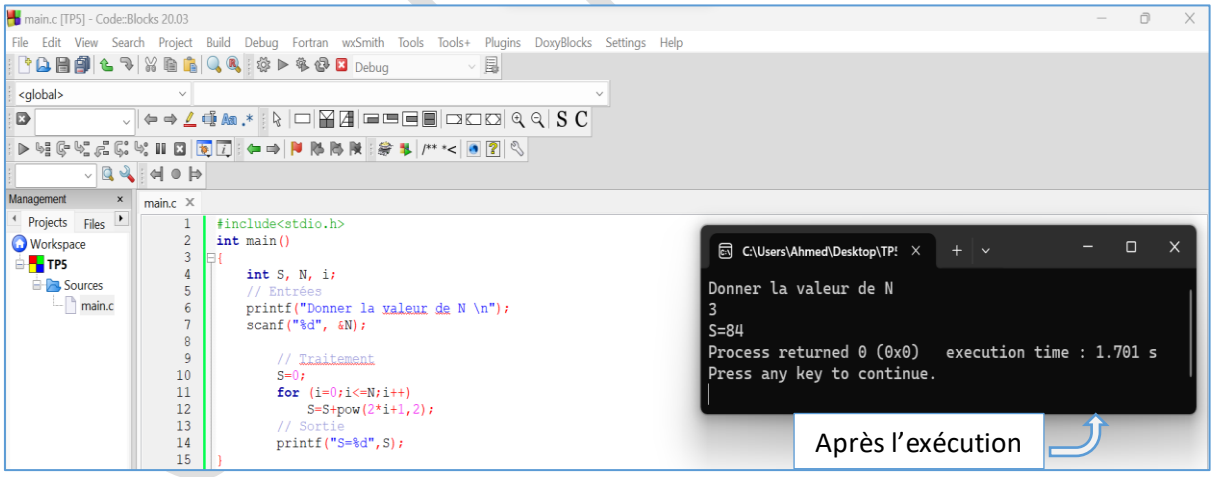


Exercice N°02 : *(Enoncé du problème ➔ Algorithme ➔ Programme en langage C)*

1) Calculer la somme $S = 1^2 + 3^2 + 5^2 + \cdots + (2 * N + 1)^2$

Algorithme	**Programme C**

Algorithme exo2_1;
variables
S, N, i: **entier**;
Début
 	
// Entrées
Ecrire("Donner la valeur de N : ");
Lire(N);
 	
// Traitement
S ← 0;
Pour i ← 0 **à** N **faire**
S ← S + (2*i+1)²;
Fin-pour
 	
// Sortie
Ecrire ("S =", S);
Fin.
	#include<stdio.h>
	int main()
	{
	int S, N, i;
	
	// Entrées
	printf("Donner la valeur de N \n");
	scanf("%d", &N);
	
	// Traitement
	S=0;
	for (i=0;i<=N;i++)
	S=S+pow(2*i+1,2);
	// S=S+(2*i+1)* (2*i+1);
	
	// Sortie
	printf("S=%d",S);
	}

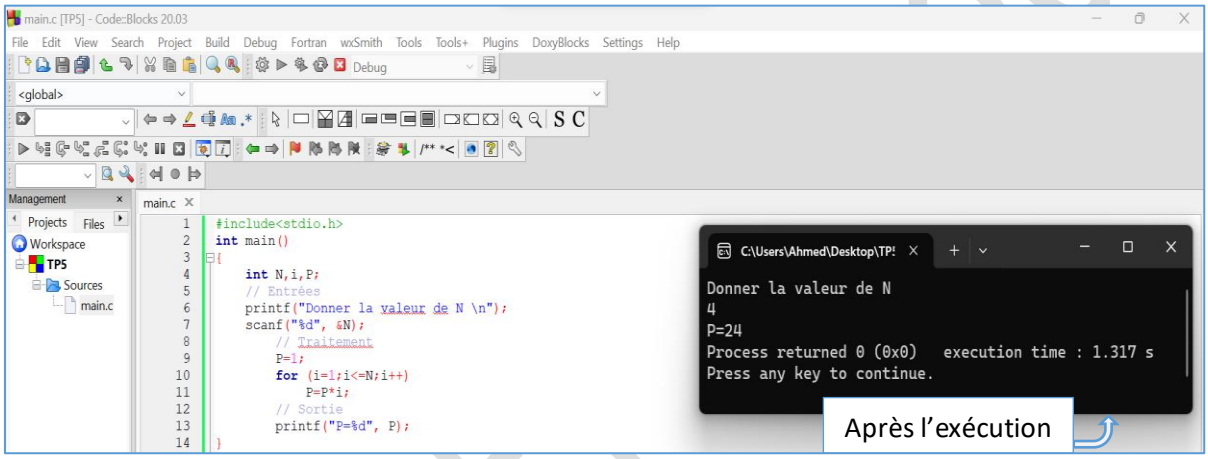


2) Calculer le produit $P = 1 \times 2 \times 3 \times \ldots \times N$

Algorithme	**Programme C**
Algorithme exo2_2; **Variables** N, i, P: **entier** ; **Début**	**#include<stdio.h>** **int main()** **{** **int** N,i,P; // Entrées

```
    // Entrées
    Ecrire("Donner la valeur de N : ");
    Lire(N);
    // Traitement
    P ← 1;
    Pour i ← 1 à N faire
        P ← P* i;
    Fin-pour;
    // Sortie
    Ecrire("P =", P);
Fin.
```

```
        printf("Donner la valeur de N \n");
        scanf("%d", &N);
            // Traitement
            P=1;
            for (i=1;i<=N;i++)
                P=P*i;
            // Sortie
            printf("P=%d", P);
}
```



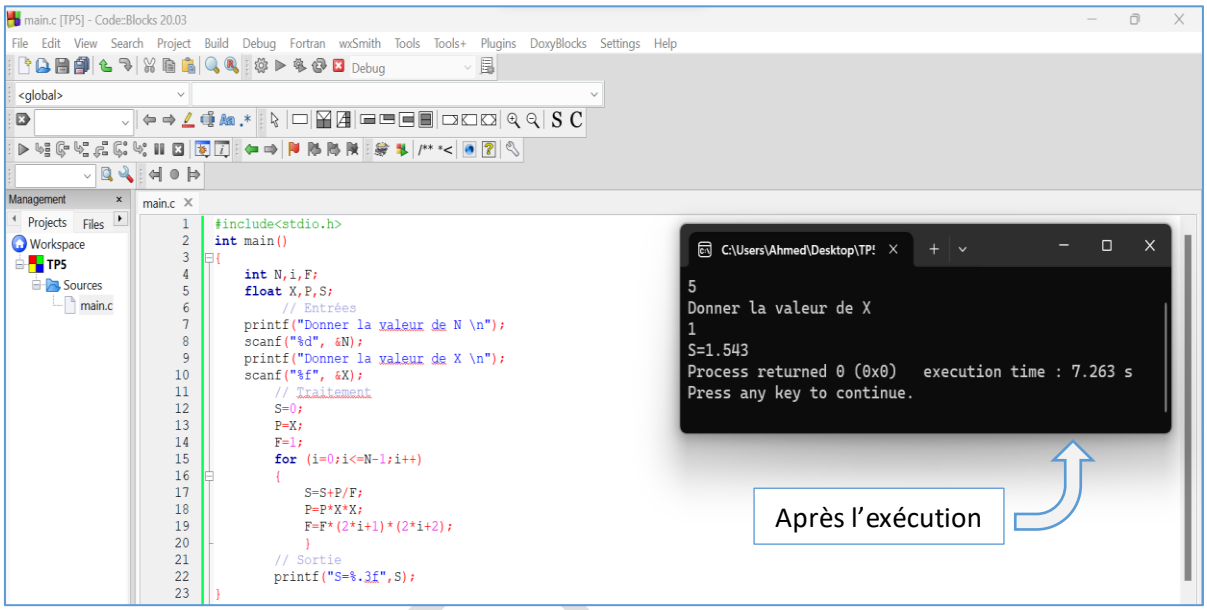
3) Calculer la somme $S = X + \frac{X^3}{2!} + \frac{X^5}{4!} + \frac{X^7}{6!} + \cdots$ ($N^{ème}$ terme)

Algorithme	**Programme C**

Algorithme exo2_3;
Variables
N, i, F : **entier**;
X, P, S : **réel**;
Début
// Entrées
Ecrire("Donner la valeur de N : ");
Lire(N);
Ecrire("Donner la valeur de X : ");
Lire(X);
// Traitement
S ← 0;
P ← X;
F ← 1;
Pour i ← 0 **à** (N-1) **faire**
	#include<stdio.h>
	int main() {
	int N,i,F;
	float X,P,S;
	// Entrées
	printf("Donner la valeur de N \n");
	scanf("%d", &N);
	printf("Donner la valeur de X \n");
	scanf("%f", &X);
	// Traitement
	S=0;
	P=X;
	F=1;
	for (i=0;i<=N-1;i++)
	{
	S=S+P/F;

```
        S ← S + P/F;
        P ← P*X*X;
        F ← F*(2*i+1)* (2*i+2);
    Fin-Pour;
  // Sortie
  Ecrire("S = ", S);
Fin.
```

```
            P=P*X*X;
            F=F*(2*i+1)*(2*i+2);
        }
        // Sortie
        printf("S=%.3f",S);
}
```

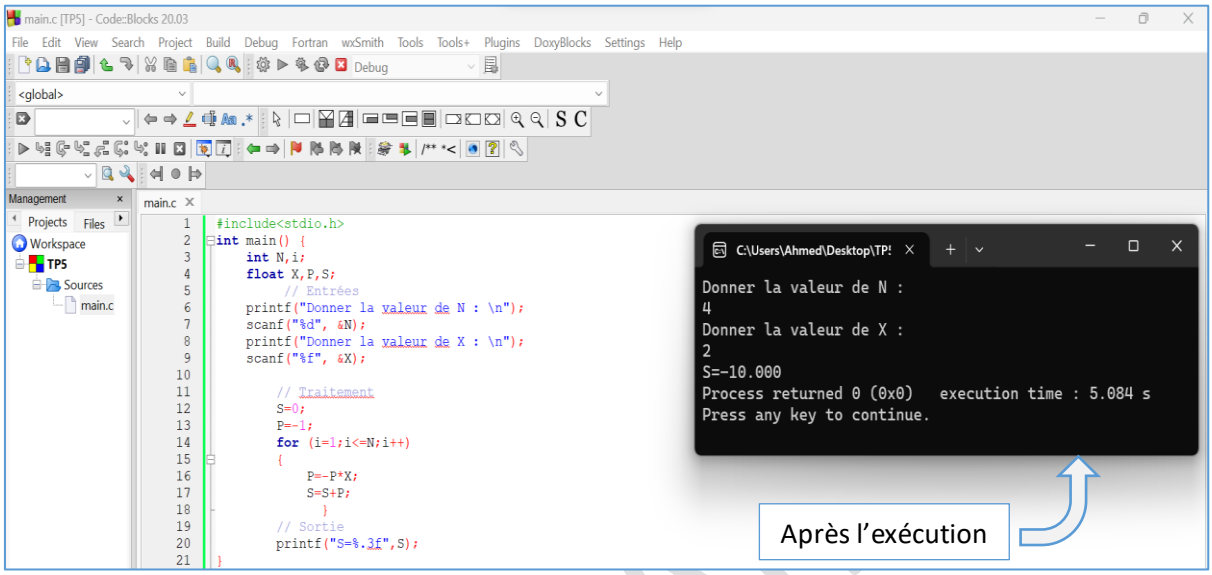


4) Calculer la somme $S = X - X^2 + X^3 - \cdots \pm X^N$

Algorithme	**Programme C**

Algorithme exo2_4;
Variables
N, i : **entier**;
X, P, S : **réel**;
Début
// Entrées
Ecrire("Donner la valeur de N : ");
Lire(N);
Ecrire("Donner la valeur de X : ");
Lire(X);
// Traitement
S ← 0;
P ← -1;
Pour i ← 1 **à** N **faire**
P ← -P*X;
S ← S+P;
	#include<stdio.h>
	int main() {
	int N,i;
	float X,P,S;
	// Entrées
	printf("Donner la valeur de N : \n");
	scanf("%d", &N);
	printf("Donner la valeur de X : \n");
	scanf("%f", &X);
	// Traitement
	S=0;
	P=-1;
	for (i=1;i<=N;i++)
	{
	P=-P*X;
	S=S+P;
	}

Fin-Pour; // Sortie **Ecrire**("S= ", S); **Fin.**	// Sortie **printf**("S=%.3f",S); **}**



Exercice N°03 :

Soit A et B deux entier tel-que A < B. Introduire N valeurs entières entre A et B, et réaliser la somme de valeurs pairs non-nuls et le produit des valeurs impaires.

Algorithme	**Programme C**

Algorithme Exo3;
Variables
A, B, N, i, Vi, S, P : **entier**;
Début
 	
// Entrées
Répéter
Ecrire("Donner la valeur de A et B : ") ;
Lire(A, B) ;
Jusqu'à (A<B);
Ecrire("Donner la valeur de N : ") ;
Lire(N) ;
S ← 0;
P ← 1;
Pour i ← 1 **à** N **faire**
Répéter
Lire(Vi)
Jusqu'à (Vi>=A) **ET** (Vi<=B);
Si Vi mod 2 = 0 **Alors**
S ← S + Vi;
Sinon
P ← P * Vi;
Fin-Si;
Fin-Pour;
 	
// Sortie
Ecrire("S = ", S, " P = ", P) ;
Fin.
	#include<stdio.h>
	int main()
	{
	int A, B, N, i, Vi, S, P ;
	
	// Entrées
	do{
	printf("Donner la valeur de A et B : \n");
	scanf("%d %d", &A, &B);
	} while(A>B);
	printf("Donner la valeur de N : \n");
	scanf("%d", &N);
	S=0;
	P=1;
	for (i=1;i<=N;i++)
	{
	do{
	scanf("%d", &Vi);
	} while (Vi<A || Vi>B);
	if (Vi % 2==0)
	S=S + Vi;
	else
	P=P * Vi;
	}
	
	// Sortie
	printf("S = %d et P= %d", S,P) ;
	}

Exercices supplémentaires sur la série de TP N°3

Exercice Sup-01 :

Ecrire un programme en langage C qui calcule et affiche la **somme** et le **produit**, des 20 premiers entiers (de 1 à 20).

Exercice Sup-02 :

Ecrire un algorithme permettant de calculer la somme de tous les nombres impairs entre deux valeurs N et M.

Exercice Sup-03 :

Ecrire un algorithme permettant de tester si un nombre N est premier en utilisant une boucle **Tant que** puis une boucle **Répéter**.

Exercice Sup-04 :

Ecrire un programme en langage C qui permet de calculer la somme S suivante : (avec X un nombre réel donné)

$$S = \frac{X^2}{2} - \frac{X^4}{4} + \frac{X^6}{6} - \frac{X^8}{8} + \frac{X^{10}}{10} - \frac{X^{12}}{12} + \cdots$$

Exercice Sup-05 :

Ecrire un programme en langage C qui permet d'afficher tous les multiples de 7 positifs strictement et inférieur à 100. Le programme doit aussi calculer et afficher la somme, le produit et la moyenne de ces multiples de 7.

On veut aussi avoir sur l'écran à l'exécution du programme, les affichages sous la forme suivante :

```
Les multiples de 7 inférieurs à 100 sont :
7 14 21 ……
La somme de ces nombres est : ……
Le produit de ces nombres est : …...
La moyenne de ces nombres est : …..
```

Exercice Sup-06 :

Un hôtel propose des chambres familiales pour ses clients. Le tarif, par nuitée, pour une personne adulte, dans une de ces chambres, est TP. À l'occasion des vacances, l'hôtel fait une réduction des prix pour les enfants mineurs (moins de 18 ans), ces réductions sont en fonction de leur âge :

- Si l'enfant a moins de 3 ans, il bénéficiera d'une réduction de 90%.
- S'il a entre 3 et 10 ans, il bénéficiera d'une réduction de 50%.
- S'il a plus de 10 ans, il bénéficiera d'une réduction de 30%.

Écrire un programme en langage C qui calcule et affiche le tarif total par nuitée pour n'importe quelle famille désirant passer des vacances dans cet hôtel.

N.B : On considère que toutes les entrées sont strictement positives, il n'est pas nécessaire de les contrôler.

Exercice Sup-07 :

Ecrire un programme en langage C faisant calculer et afficher le **factoriel** d'un entier naturel N donné. Sachant que (pour N>0) : N ! = N x (N-1) x (N-2) x ..... 3 x 2 x 1.

CHAPITRE II

Les tableaux

Les Tableaux

II.1. Objectif de ce chapitre

À l'issue de ce chapitre, l'apprenant sera capable de :

- Comprendre la notion de tableaux en langage C et leur rôle dans la manipulation de collections de données.
- Distinguer les tableaux unidimensionnels (vecteurs) des tableaux bidimensionnels (matrices) et comprendre leur organisation en mémoire.
- Déclarer, initialiser et parcourir des vecteurs et des matrices en utilisant des structures de contrôle appropriées.
- Manipuler les éléments des tableaux (lecture, écriture, modification) à l'aide d'indices.
- Implémenter des algorithmes simples sur les vecteurs et matrices (recherche, somme, maximum, minimum, etc.).
- Résoudre des problèmes concrets en langage C faisant intervenir des tableaux unidimensionnels et bidimensionnels.

II.2. Introduction

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

Exemple :

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

Solution :

Algorithme Note ;
Variables
N1, N2, N3, N4, N5 : réel ;
Début
Ecrire ('Enter la 1ère Note') ;
Lire(N1) ;
Ecrire ('Enter la 2ème Note') ;

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

Fin.

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

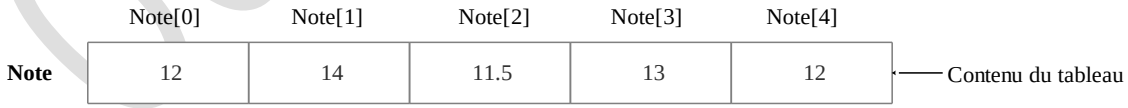
Comme les variables ont des noms différents, on ne peut pas utiliser de boucle, ce qui allonge considérablement le code et le rend très répétitif.

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

II.3. Définition

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

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



Nous disposons alors de cinq variables. Pour les nommer, on indique le nom du tableau suivi de son indice entre crochets ou entre parenthèses : La première s'appelle Note[0], la deuxième Note[1], etc. jusqu'à la dernière Note[4].

Note[3] représente le 4ème élément du tableau **Note** et vaut 13.

II.4. Tableau à une dimension

II.4.1. Déclaration

La syntaxe à suivre pour déclarer un tableau est la suivante :

Syntaxe algorithmique	Syntaxe en C
constante MAX = <valeur_entière> ; **Variable** <id_tab> : **tableau**[0..MAX-1] de <type)> ;	**const** **int** MAX = <valeur_entière> ; <type> <id_tab> [MAX] ;

Remarques :

- Ce n'est pas obligatoire d'utiliser une constante pour la taille maximale de tableau ; cependant, c'est une bonne pratique dans la programmation.
- La plage d'indice *0 .. MAX-1* représente l'indice du premier élément *0* et l'indice du dernier élément *MAX-1*.
- C'est pas obligatoire d'utiliser toutes les composantes du tableau, pour cela on déclare une variable entière N qui représente la taille du tableau à utiliser. La valeur de cette variable sera introduite au cours de l'exécution (par lecture)

Exemple :

Syntaxe algorithmique	Syntaxe en C
constante MAX = 50; **Variable** V : **tableau**[0..MAX-1] de réel ;	**const int** MAX = 50 ; float V[MAX] ;

V est un tableau de 50 composantes réelles (c'est comme si on a déclaré 50 variables réelles)

II.4.2. Initialisation d'un tableau unidimensionnel

Un tableau peut être initialisé en utilisant deux méthodes :

– En utilisant des affectations ;

– En utilisant la lecture à partir d'un fichier de données ou à partir de clavier.

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

Exemple :

V[0] = 24.00; *V[1] = 38.25*; *V[2] = 28.00*; *V[3] = 70.25*; *V[4] = 63.00* ; *V[5] = 96.25*;

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

Exemple 1 : Exemple d'algorithme / programme C permettant de lire et d'écrire (afficher) le tableau précédent :

```
algorithme LireEcrireTableau ;
variables
   v:tableau[0..5] de réel
   i:entier
Début
    écrire('Introduire V :');
    pour i←0 à 5 faire
          lire(v[i]);
    fin-Pour;
    écrire('Affichage de V :');
    pour i←0 à 5 faire
          écrire(v[i]);
    fin-Pour;
Fin;
```

```
#include <stdio.h>
int main()
{
      float v[6];
      int i;

   printf("Introduire V : \n");
   for (i=0; i<6; i++)
      scanf ("%f", &V[i]);
   printf ("Affichage V : \n") ;
   for (i=0; i<6; i++)
         printf ("%.2f  ",V[i]);
}
```

Exemple 2 : Lecture et écriture d'un tableau de *N* éléments.

```
algorithme LireEcrireTableau
   variables
            V:Tableau[0..99] de réel
            N, i:entier

Début
  écrire('Introduire nbre d'éléments N : ')
  lire (N)
  écrire('Introduire V : ')
  pour i←0 à N-1 faire
     lire (V[I]);
  finPour;

  écrire ('Affichage de V : ') ;
  pour i←0 à N-1 faire
    écrire (V[I]);
  finPour;
Fin.
```

```
#include <stdio.h>

int main()
{
   float V[100];
   int N, i;
    printf("Introduire nbre d'éléments N : \n");
   scanf ("%d" , &N);

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

   printf("Affichage du V : \n") ;
   for (i=0; i<N ; i++)
      printf ("%.2f", V[i]);
}
```

II.4.3. Manipulation de Tableaux unidimensionnels (Vecteurs)

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

Exemples : Nous allons voir quelques exemples d'algorithmes pour les tableaux à une seule dimension (Vecteurs).

a) Somme de deux Vecteurs V1 et V2

```
Algorithme Somme
 Variables
   V1,V2,V:Tableau[0..99] de réel;
   N, i:entier;
Début
   Lire (N);
   pour i←0 à N-1 faire
       Lire (V1[i]);
   FinPour;

   pour i←0 à N-1 faire
       Lire (V2[i]);
   FinPour;

   pour i←0 à N-1 faire
       V[i] ← V1[i] + V2[i];
   FinPour;

   pour i←0 à N-1 faire
       Écrire (V[i]);
   FinPour;
Fin.
```

```
#include <stdio.h>

int main()
{
   float V1[100], V2[100], V[100];
   int N, i;

    printf("Introduire nbre d'éléments N : \n");
   scanf ("%d" , &N);

   printf("Introduire V1 : \n");
   for (i=0; i<N ; i++)
      scanf ("%f", &V1[i]);

   printf("Introduire V2 : \n");
   for (i=0; i<N ; i++)
      scanf ("%f", &V2[i]);

   for (i=0; i<N ; i++)
      V[i] = V1[i] + V2[i];

   printf("Affichage du V : \n") ;
   for (i=0; i<N ; i++)
      printf ("%.2f", V[i]);
}
```

b) Produit scalaire de deux vecteurs V1 et V2

```
Algorithme ProduitScalaire;

Variables
   V1, V2 : Tableau [0..99] de réel;
   N, i:entier;
   PS : réel;
Début
   Lire (N)
   pour i←0 à N-1 faire
     Lire (V1[i]);
   Fin-Pour;
   pour i←0 à N-1 faire
     Lire (V2[i]);
   Fin-Pour;

   PS ← 0;
   pour ←0 à N-1 faire
     PS ← PS + V1[i] * V2[i];
   FinPour;

   Écrire(PS);
Fin
```

```
#include <stdio.h>

int main()
{
   float V1[100], V2[100], ps;
   int N, i;

   printf("Introduire nbre d'éléments N : \n");
   scanf ("%d" , &N);

   printf("Introduire V1 : \n");
   for (i=0; i<N ; i++)
      scanf ("%f", &V1[i]);

   printf("Introduire V2 : \n");
   for (i=0; i<N ; i++)
      scanf ("%f", &V2[i]);

   Ps = 0;
   for (i=0; i<N ; i++)
      ps += V1[i]*V2[i];

   printf ("Produit scalaire %.2f", Ps);
}
```

c) Recherche d'un élément dans un tableau

Chercher un élément val dans un tableau à une dimension (vecteur). S'il existe on récupère son rang.

```
Algorithme Recherche;
 Variables
   V:Tableau[0..99] de reel;
   Val : reel;
   N, I, R:entier;
   Trouve : boolean;
Début
   Lire (N);
   pour i←0 à N-1 faire
       Lire (V1[i]);
   FinPour
   Lire (Val);

   i ← 1 ;
   Trouve ← Faux;
   TantQue (i<=N) et (Trouve=Faux) faire
       Si V[i] = Val Alors
          Trouve ← Vrai;
          R ← i;
       FinSi
       i ← i + 1;
   Fin-Tant-Que;

   Si Trouve = Vrai Alors
     Écrire('La valeur ',Val,' exist.');
     Écrire('Son Rang est : ', R);
   Sinon
     Écrire(Val,' n''existe pas dans V.');
   FinSi;
Fin;
```

```
#include <stdio.h>
#include <stdbool.h>

int main()
{
   float V[100], val;
   int n, i, pos;
   bool trouve;

   printf("Donenr la taille de V : \n");
   scanf ("%d" , &n);
   printf("Introduire V1 : \n");
   for (i=0; i<n ; i++)
      scanf ("%f", &V[i]);
   scanf("%f", &val);

   i=0; trouve=false;
   while ((i<n) && (!trouve)){
     if (V[i] == val){
        trouve=true; pos = i;
     }
     i++;
   }

   if (trouve)
    printf("La valeur %f existe à la position = %d", val, pos) ;
   else
    printf("La valeur %f n'existe pas dans V", val);
}
```

d) Recherche du maximum dans un tableau unidimensionnel

Soient deux tableaux NOMS et NOTES contenant respectivement des noms d'étudiants et leurs notes respectives, on veut sélectionner la meilleurs note de l'étudiant correspondant.

```
Algorithme Maximum;
 Variables
   Noms : Tableau[0..99] de chaîne;
   Notes : Tableau[0..99] de réel ;
   N, i, imax : entier;
   max : réel;
Début
   Lire (N);
   pour i←0 à N-1 faire
     Lire (Noms[i]);
     Lire (Notes[i]);
   Fin-Pour;
   max ← Notes[0];   imax ← 0;
   pour i←1 à N-1 faire
      si Notes[i] > max then
           max ← Notes[i];
           imax ← i;
       fin-si;
   Fin-Pour;

   Écrire('L''étudiant : ', Noms[imax]);
   Écrire(' a obtenu la meilleur note :');
   Écrire (max);
Fin.
```

```
#include <stdio.h>
int main()
{
   char Noms[100][100];
   float Notes[100];
   int n, i, imax;
   float max;
   scanf ("%d",  &n);
   for (i=0; i<n;  i++) {
      scanf ("%[^\n]",  Noms[i]);
      scanf ("%f",  &Notes[i]);
   }
   max=Notes[0] ;  imax=0;
   for (i=1; i<n; i++)
   {
      if (Notes[i] > max) {
          max = Notes[i]; imax=i;
       }
   }
   printf("L'étudiant : %s", Noms[imax]);
   printf("a obtenu la meilleur note :%.2f", max) ;
}
```

e) L'algorithme de Tri par sélection (permutations)

– Méthode de Tri Simple (Tri croissant)

```
Algorithme Tri_Selection;
 Variables
   T:Tableau[0..99] de reel;
   I, J, N:entier;
   Z : real;
Début
   Lire (N);
   pour i←1 à N-1 faire
       Lire (T[i]);
   Fin-Pour;

   pour i←1 à N-2 faire
      pour j=(i+1) à N-1 faire
            Si T[I] > T[J] Alors
                  Z ← T[I];
                  T[I] ← T[J];
                  T[J] ← Z;
            Fin-Si;
      Fin-Pour;
   Fin-Pour;

   pour i=1 à N faire
       Ecrire (T[i]);
   Fin-Pour;
Fin;
```

```
#include <stdio.h>
int main()
{
      float T[100];
      int i, j, n;
      float Z:real;

      scanf("%d", &n);
      for (i=0; i<n; i++)
          Read("%f", &T[i]);

      for (i=0; i<(n-1); i++)
        for (j=i+1; j<n; j++)
            if (T[i] > T[j])
            {
              Z = T[i];
              T[i] = T[j];
              T[j] = z;
            }

      for (i=0; i<n; i++)
            printf("%8.2f   ", T[i]);
}
```

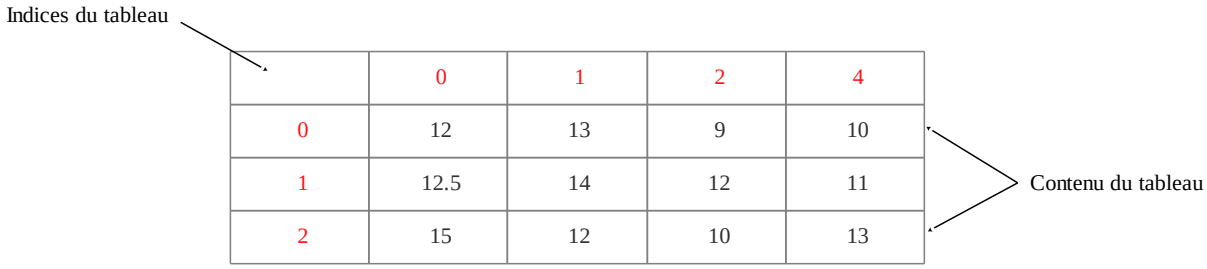
II.5. Tableaux à deux dimensions

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

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

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

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



	0	1	2	4
0	12	13	9	10
1	12.5	14	12	11
2	15	12	10	13

II.5.1. Déclaration

La matrice A définie précédemment est déclarer comme suit :

Syntaxe algorithmique	Syntaxe en C
variable A : **tableau**[0..2,0..3] **de** réel ;	**float** A [3][4] ;

Exemple :

Une chaîne de 28 magasins, chacun comportant 4 rayons. On veut établir l'état des ventes hebdomadaires. Ces données sont ensuite entrées dans un ordinateurs en utilisant un tableau à deux dimensions où le premier indices repère le magasin et le second le rayon. Si VENTES est le nom du tableau, décrire cette représentation des données.

VENTES	***00***	***01***	***02***	***03***
00	2872	805	3211	1560
01	2196	1225	2525	1477
02	3257	1017	3687	1951
….	….	….	….	….
….	….	….	….	….
….	….	….	….	….
….	….	….	….	….
27	2618	913	2333	982

Variables

VENTES : **tableau** [0..27,0..3] **de** entier ;

II.5.2. Lecture et affichage d'une matrice

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

```
Algorithme LectureAffichageMatrice
 Variables
   mat:tableau [0..99, 0..99] de réel;
   N,M, i,j:entier;
Début
   Lire (N, M);
   pour i←0 à N-1 faire
      Pour j←0 à M-1 faire
         Lire (mat[i, j]);
      FinPour;
   FinPour;

   pour i←0 à N-1 faire
      Pour j←0 à M-1 faire
         Écrire (mat[i, j]);
      FinPour;
   FinPour;
Fin.
```

```
#include <stdio.h>
int main()
{
      float mat[100][100];
      int i, j, n, m;

      scanf("%d %d", &n, &m);
      for (i=0; i<n; i++)
         for (j=0; j<m; j++)
            scanf("%f", &mat[i][j]);

      for (i=0; i<n; i++)
         for (j=0; j<m; j++)
            printf("%8.2f", mat[i][j]);
      }
}
```

On doit introduire le nombre de lignes *n* et le nombre de colonnes *m.* par la suite on introduit les éléments de la matrice mat[i,j] / i=1...n et j=1..m

II.5.3. Manipulation de matrice

a) Lire et écrire un tableau bidimensionnels

```
Algorithme LectureEcritureMatrice
 Variables
   mat:tableau [0..99, 0..99] de reel;
   N,M, i,j:entier;
Début
   Lire (N, M);
   pour i←0 à N-1 faire
      Pour j←0 à M-1 faire
         Lire (mat[i, j]);
      FinPour;
   FinPour;

   pour i←0 à N-1 faire
      Pour j←0 à M-1 faire
         Écrire (mat[i, j]);
      FinPour;
   FinPour;
Fin.
```

```
#include <stdio.h>
int main()
{
      float mat[100][100];
      int i, j, n, m;

      printf("Donner n et m :");
      scanf("%d %d", &n, &m);
      for (i=0; i<n; i++)
         for (j=0; j<m; j++)
            scanf("%f", &mat[i][j]);

      printf("Affichage de la matrice mat :\n");
      for (i=0; i<n; i++) {
         for (j=0; j<m; j++)
            printf("%8.2f", mat[i][j]);

         printf("\n");//Saut de ligne
      }
}
```

b) Produit d'une matrice par un vecteur

(le nombre de colonne de la matrice = le nombre de composantes du vecteur)

```
Algorithme ProduitMatVect;
 Variables
   mat : tableau [0..49, 0..49] de reel;
   vect : tableau [0..49] de reel;
   p : tableau [0..49] de reel;
   N,M, i,j:entier;
Début
   Lire (N, M);
   pour i←0 à N-1 faire
      Pour j←0 à M-1 faire
        Lire (mat[i, j]);
      FinPour;
   FinPour;
   pour i←0 à M-1 faire
      Lire(vect[i]);
   finPour
   pour i←0 à N-1 faire
      p[i] = 0;
      Pour j←0 à M-1 faire
        P[i] ← p[i] + mat[i, j]*vect[j];
      FinPour;
   FinPour;
   pour i←0 à N-1 faire
      Écrire(p[i]);
   finPour;
Fin.
```

```
#include <stdio.h>

int main()
{
   float mat[10][10],vect[10], p[10];
   int i, j, n, m;

   printf("Introduire la taille de MAT : ");
   scanf("%d %d", &n, &m);
   printf("Introduire les valeurs de MAT : \n");
   for (i=0; i<n; i++)
      for (j=0; j<m; j++)
         scanf("%f", &mat[i][j]);
   printf("Introduire les valeurs de VECT : \n");
   for (j=0; j<m; j++)
      scanf("%f", &vect[j]);

   for (i=0; i<n; i++) {
      p[i] = 0;
      for (j=0; j<m; j++)
         p[i] += mat[i][j] * vect[j];
   }

   printf("Produit P = MatxVect : \n");
   for (i=0; i<n; i++)
      printf("%8.2f", p[i]);
}
```

c) Compter le nombre d'éléments négatifs, positifs et nuls dans une matrice

```
Algorithme compteurPNN
 Variables
   mat : tableau [1..10, 1..10] de reel;
   N,M, i,j:entier
   nbP, nbN, nbNul : entier;
Début
   Lire (N, M);
   pour i←0 à N-1 faire
      Pour j←0 à M-1 faire
        Lire (mat[i, j]);
      FinPour;
   FinPour;
   nbP ← 0; nbN ← 0; nbNul ← 0;
   pour i←0 à N-1 faire
      Pour j=←0 à M-1 faire
        Si mat[i, j] > 0 Alors
          nbP ← nbP + 1;
        Sinon
          Si mat[i,j] < 0 Alors
            nbN ← nbN + 1;
          Sinon
            nbNul ← nbNul + 1;
          FinSi;
        FinSi;
      FinPour;
   FinPour;

   Écrire (nbP, nbN, nbNul);
Fin.
```

```
#include <stdio.h>

int main()
{
   float mat[10][10];
   int i, j, n, m;
   int nbP, nbN, nbNul;

   printf("Introduire la taille de MAT : ");
   scanf("%d %d", &n, &m);
   printf("Introduire les valeurs de MAT : \n");
   for (i=0; i<n; i++)
      for (j=0; j<m; j++)
         scanf("%f", &mat[i][j]);

   nbP = 0; nbN=0; nbNul=0;

   for (i=0; i<n; i++)
      for (j=0; j<m; j++)
         if (mat[i][j] > 0)
            nbP++;
         else if (mat[i][j] < 0)
            nbN++;
         else
            nbNul++;

   printf("Nb valeurs positives %d : \n", nbP);
   printf("Nb valeurs négatives %d : \n", nbN);
   printf("Nb valeurs nulles %d : \n", nbNul);
}
```

nbP est le compteur des nombres positifs, **nbN** est le compteur des nombre négatifs et **nbNul** est le compteur des nombre nuls.

d) Produit de deux matrices

Le nombre de colonne de la première matrice = le nombre de ligne de la deuxième matrice

```
Algorithme compteurPNN
 Variables
  m1,m2 : tableau [0..49, 0..49] de reel
  N,M, L, i,j, k:entier
  Result:tableau [0..49, 0..49] de reel
Début
   Lire (N, M, L);
   pour i←0 à N-1 faire
      Pour j←0 à M-1 faire
        Lire (m1[i, j]);
      FinPour;
   FinPour;

   pour i←0 à M-1 faire
      Pour j←0 à L-1 faire
        Lire (m2[i, j]);
      FinPour;
   FinPour;

   pour i←0 à N-1 faire
      Pour j←0 à L-1 faire
        Result[i, j] ← 0;
        pour k←0 à M-1 faire
          Result[i,j]←Result[i,j]+
                      m1[i,k]*m2[k,j];
        finPour;
      FinPour;
   FinPour;

   pour i←0 à N-1 faire
      Pour j←0 à L-1 faire
        Écrire (Result[i, j]);
      FinPour;
   FinPour;
Fin.
```

```
#include <stdio.h>
int main()
{
  float m1[10][10], m2[10][10];
  int N,M, L, i,j, k;
  float Result[10][10];

  printf("Donnez les dim. de m1 : ");
  scanf ("%d %d", &N, &M);
  printf('Donnez le nbre de colonnes de m2 : ');
  scanf ("%d", &L);

  for (i=0; i<N ; i++)
    for (j=0; j<M ; j++)
      scanf("%f", &m1[i][j]);

  for (i=0; i<M ; i++)
    for (j=0; j<L ; j++)
      scanf("%f", &m2[i][j]);

  for (i=0; i<M ; i++)
    for (j=0; j<L ; j++)
    {
        Result[i, j] = 0;
        for (k=0; k<M ; k++)
          Result[i,j] += m1[i,k]*m2[k,j];
    }

  printf("La matrice Resultat=M1xM2 est :\n");
  for (i=0; i<M ; i++)
  {
    for (j=0 ; j<L ; j++)
      printf("%8.2f", Result[i,j]);
    printf("\n"); //Saut de ligne
  }
}
```

Pour réaliser le produit des deux matrices **m1** et **m2**, il faut que le nombre de colonne de **m1** soit égale au nombre de ligne de **m2**. Le résultat est une matrice Result dont le nombre de ligne est le même que celui de **m1** et le nombre de colonne est le même que celui de **m2**.

Result(N, L)=m1(N,M)*m2(M,L)

- Result est une matrice de N lignes et L colonnes
- m1 est une matrice de N lignes et M colonnes
- m2 est une matrice de M lignes et L colonnesf
- L'élément Result[i,j] est calculé comme suit :

  Result[i,j]=m1[i,1]*m2[1,j]+m1[i,2]*m2[2,j]+m1[i,3]*m2[3,j]+….+m1[i,M]*m2[M,j]

  $=\sum_{k=1}^{M} m1[i,k] * m2[k,j]$

Série de TP N°4 – 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 :

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 */
	#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;

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.
	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 = 6 et T=[14, 3, 8, 22].

```c
#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;
}
```

```
MyC V1.20.5 (Exécution) (
Donner la taille du vecteur T :4
Donner les composantes du vecteur T :
14 3 8 22
La somme S= 44
```

Après l'exécution

3- Dérouler l'algorithme pour : N = 6 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).

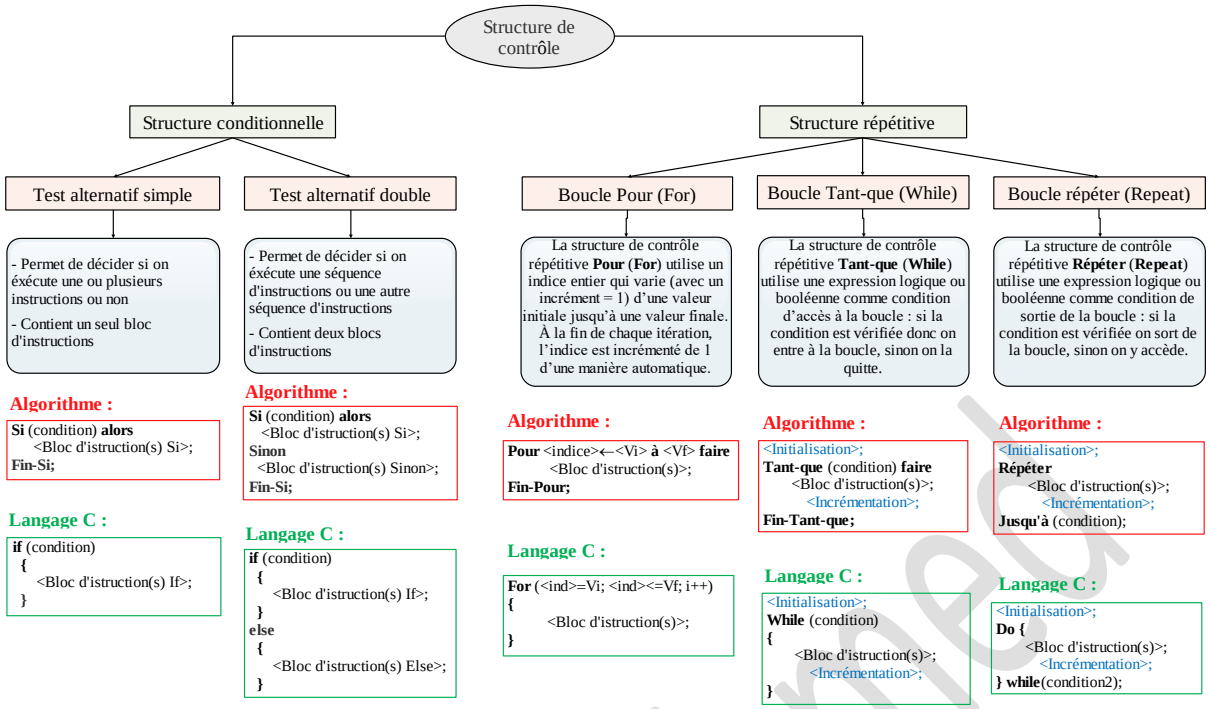
Instructions	Variables				Affichage
	N	i	T	S	
Ecrire('Donner la taille du vecteur T : ');	/	/	/	/	Donner la taille du vecteur T :
Lire(N)	4	/	/	/	//
Ecrire('Donner les composantes du vecteur T : ');	//	/	/	/	Donner les composantes du vecteur T :
Pour i←0 **à** N-1 **faire** Lire(T[i]) **Fin-Pour**	//	0 1 2 3	0 1 2 3 [14, 3, 8, 22]	/	//
S ← 0	//	/	//	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	//	0	//	14	//
Pour i=1 **Si** T[1] mod 2 = 0 **Alors** **3 mod 2 = 0** **Faux**	//	1	//	//	//
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	//	2	//	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	//	3	//	44	//
Ecrire('La somme S = ', S) ;	//		//	//	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)

Rappel sur les boucles :



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 (Repeat)

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 **1** à la valeur **N**. 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 :

Algorithme/Programme C

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]);
	}
	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 vecteur

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

Solution :

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
Ecrire('Donnez les éléments du tableau : ') ;
Pour i ← 0 **à** 9 **faire**
Lire(V[i]) ;
FinPour ;
Som ← 0 ;
Prod ← 1 ;
Pour i ← 0 **à** 9 **faire**
Som ← Som+V[i] ;
Prod ← Prod*V[i] ;
FinPour ;
Moy ← Som/10 ;
 	
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)

Algorithme/Programme C

Algorithme	Programme C

Algorithme Min_PosMin ;
Variables
V : Tableau [0..99] **de réel ;**
i, N, Pos_Min : **entier** ;
Min : **réel** ;
Début
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
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
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)

Algorithme/Programme C

Algorithme	Programme C

Algorithme Max_PosMax ;
Variables
V : Tableau [0..99] **de réel** ;
i, N, Pos_Max : **entier** :
Max : **réel** ;
Début
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
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
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 :

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

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

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

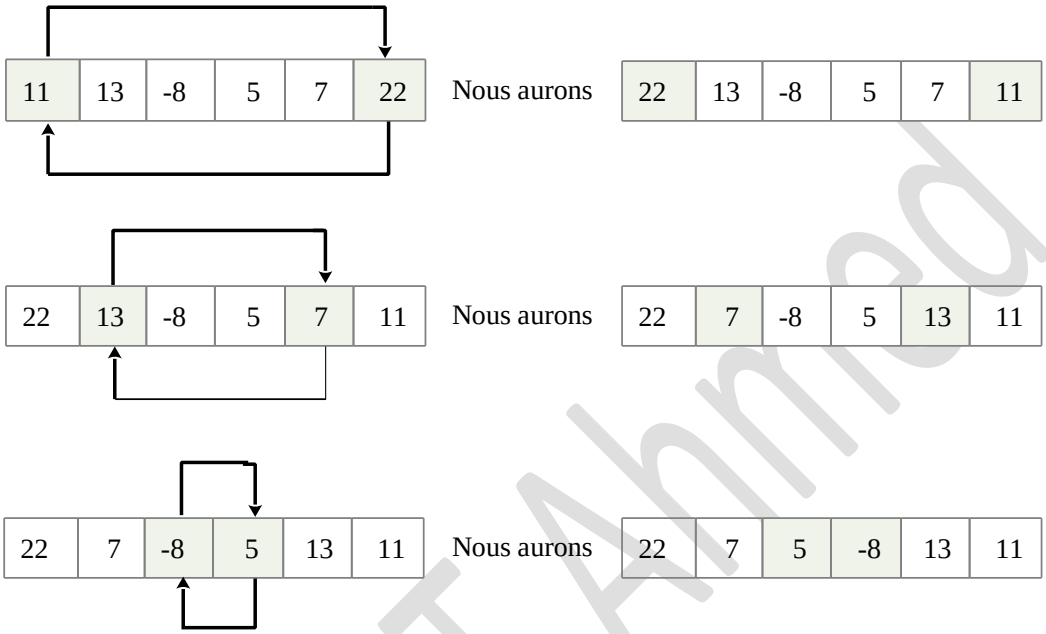
Le même problème que la question 01, sauf que, dans cette deuxième question, nous voulons inverser le vecteur T dans lui-même.

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)
```

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

<u>Exercice N°06 :</u> La recherche d'une valeur dans un vecteur.

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

<u>Solution :</u>

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.

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 ;
	#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 */
    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.
```

```
  /* 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;
}
```

Serie de TP N°5 – Tableaux à deux dimensions - Matrices

Exercice N°01 : Algorithme → Programme C

Soit l'algorithme suivant :

```
Algorithme Matrice ;
Variables
    A : Tableau [0..99, 0..99] de réel;
    i, j, N : entier;
    S, M : réel;
Début
    // Entrées
    Ecrire('Donner la taille de la matrice carrée
            A :');
    Lire(N);
    Ecrire('Donner les composantes de la matrice
            A :');
    Pour i ←0 à N-1 faire
      Pour j ←0 à N-1 faire
        Lire(A[i,j]);
      FinPour;
    FinPour;
   // Traitements
    S ← 0;
    Pour i ← 0 à N-1 faire
      S ← S+A[i,i] ;
    FinPour;
    M ← S/N ;
   // Sorties
    Ecrire('S=', S, 'M=', M);
Fin.
```

Questions :

1- Traduire l'algorithme en Programme C.

2- Compiler et exécuter le programme pour :
N = 3 et

$$A = \begin{bmatrix} 2 & 4 & -4 \\ 4 & 8 & 1 \\ 3.5 & 9 & 4 \end{bmatrix}$$

3- Dérouler le programme pour les valeurs de N et A ci-dessus ?

4- Déduire ce que fait le programme ?

5- Ré-écrire le programme en remplaçant la boucle *Pour* par la boucle *Tantque* **dans la partie des entrées**.

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

Solution :

1 - Algorithme/programme C :

Algorithme	Programme C

Algorithme Matrice ;
Variables
A : Tableau [0..99, 0..99] **de réel**;
i, j, N : **entier**;
S, M : **réel**;
Début
// Entrées
Ecrire('Donner la taille de la matrice carrée A :');
Lire(N);
Ecrire('Donner les composantes de la matrice A :');
	#include<stdio.h>
	int main()
	{
	float A[100][100];
	int i, j, N;
	float S, M;
	
	// Entrées
	printf("Donner la taille de la matrice carrée A :");
	scanf("%d", &N);
	printf("Donner les composantes de la matrice A : \n");

```
Pour i ←0 à N-1 faire
  Pour j ←0 à N-1 faire
    Lire(A[i,j]);
  FinPour;
FinPour;
// Traitements
S ← 0;
Pour i ← 0 à N-1 faire
  S ← S+A[i,i] ;
FinPour;
M ← S/N ;

// Sorties
Ecrire('S= ', S, ' M=', M);
Fin.
```

```
for (i=0;i<N;i++)  {
      for (j=0;j<N;j++)  {
        scanf("%f", &A[i][j]);
  }
  }
  // Traitements
  S=0;
  for (i=0;i<N;i++)  {
        S=S+A[i][i];
        }
  M=S/N;

  // Sorties
  printf(" S = %.2f, M= %.2f", S,M);

return 0;
}
```

2 - Compiler et exécuter le programme pour : N = 3 et

$$A = \begin{bmatrix} 2 & 4 & -4 \\ 4 & 8 & 1 \\ 3.5 & 9 & 4 \end{bmatrix}$$

```c
#include<stdio.h>
int main() {
    float A[100][100];
    int i,j,N;
    float S,M;
    /* Entrées */
    printf("Donner la dimension de la matrice A :");
     scanf("%d", &N);
    printf("Donner les composantes de la matrice A : \n");
     for (i=0;i<N;i++) {
          for (j=0;j<N;j++)  {
      scanf("%f", &A[i][j]);
     }
     }
     /* Traitements */
     S=0;
      for (i=0;i<N;i++)  {
     S=S+A[i][i];
     }
     M=S/N;
   /* Sorties */
     printf(" S = %.2f, M= %.2f", S,M);
return 0;
}
```

```
MyC V1.20.5 (Exécution) C:\Users\Ahmed\Deskto...
Donner la dimension de la matrice A :3
Donner les composantes de la matrice A :
2 4 -4
4 8 1
3.5 9 4
 S = 14.00, M= 4.67
```

Après l'exécution

3 - Dérouler le programme pour les valeurs de N et A ci-dessus ?

Instructions	Variables						Affichage
	N	i	j	A	S	M	
printf("Donner la dimension de la matrice A :");	/	/	/	/	/	/	Donner la dimension de la matrice A :
scanf("%d", &N);	3	/	/	/	/	/	
printf("Donner les composantes de la matrice A : \n");	3	/	/	/	/	/	Donner les composantes de la matrice A :

for(i=0;i<N;i++){
for (j=0;j<N;j++){
scanf("%f", &A[i][j]);
}}
	3	0
		1
		2
			1
			2
			3
				j=0 j=1 j=2
				i=0 2 4 -4
				i=1 4 8 1
				i=2 3.5 9 4
					/	/	//

S=0;	3	/	/	//	0	/	//
For i=0 S=S+A[i,i]; S=0+A[0,0]; S=0+2=2;	3	0		//	2	/	//
For i=1 S=S+A[i,i]; S=2+A[1,1]; S=2+8=10;	3	1	/	//	10	/	//
For i=2 S=S+A[i,i]; S=10+A[2,2]; S=10+4=14;	3	2	/	//	14	/	//
M=S/N; M=14/3=4.67;	3			//	14	4.67	//
printf(" S = %.2f, M= %.2f", S,M);	3			//	14	4.67	S=14.00 M=4.67

4 - Déduire ce que fait le programme ?

Le programme calcul la somme des éléments de la diagonale de la matrice A et leur moyenne.

5 - Ré-écrire le programme en remplaçant la boucle *Pour* par la boucle *Tantque* dans la partie des entrées.

Programme C (avec la boucle For)	Programme C (avec la boucle While)

#include<stdio.h>
int main() **{**
float A[100][100];
int i, j, N;
float S, M;
// Entrées
printf("Donner la dimension de la matrice A :");
scanf("%d", &N);
printf("Donner les composantes de la matrice A : \n");
for (i=0;i<N;i++) {
for (j=0;j<N;j++) {
scanf("%f", &A[i][j]);
}
}
// Traitements
S=0;
for (i=0;i<N;i++)
{
	#include<stdio.h>
	int main() **{**
	float A[100][100];
	int i, j, N;
	float S, M;
	// Entrées
	printf("Donner la dimension de la matrice A :");
	scanf("%d", &N);
	printf("Donner les composantes de la matrice A : \n");
	i=0;
	while(i<N) {
	j=0;
	while(j<N) {
	scanf("%f", &A[i][j]);
	j=j+1;
	}
	i=i+1;
	}

```
            S=S+A[i][i];
        }
      M=S/N;

    // Sorties

    printf(" S = %.2f, M= %.2f", S,M);
return 0;
}
```

```
    // Traitements
     S=0;
     for (i=0;i<N;i++)  {
           S=S+A[i][i];
     }
     M=S/N;
    // Sorties
        printf(" S = %.2f, M= %.2f", S,M);
return 0;
}
```

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

Programme C (avec la boucle For)

```
#include<stdio.h>
int main()
{
    float A[100][100];
    int i, j, N;
    float S, M;
    // Entrées
    printf("Donner la dimension de la matrice A
            :");
     scanf("%d", &N);
    printf("Donner les composantes de la matrice
            A : \n");
    for (i=0;i<N;i++)  {
        for (j=0;j<N;j++)  {
            scanf("%f", &A[i][j]);
     }
     }
      // Traitements
     S=0;
     for (i=0;i<N;i++)
            {
            S=S+A[i][i];
            }
      M=S/N;

  // Sorties
  printf(" S = %.2f, M= %.2f", S,M);

return 0;
}
```

Programme C (avec la boucle Repeat)

```
#include<stdio.h>
int main() {
    float A[100][100];
    int i, j, N;
    float S, M;
    // Entrées
    printf("Donner la dimension de la matrice A :");
    scanf("%d", &N);
    printf("Donner les composantes de la matrice A : \n");
    i=0;
            do {
                    j=0;
                do {
            scanf("%f", &A[i][j]);
            j=j+1;
     } while(j<N) ;
       i=i+1;
    } while(i<N);
     // Traitements
     S=0;
     for (i=0;i<N;i++)  {
            S=S+A[i][i];
            }
            M=S/N;
   // Sorties
   printf(" S = %.2f, M= %.2f", S,M);
return 0;
}
```

Exercice N°02 : Transposée d'une matrice

Ecrire un algorithme/programme C qui permet de calculer la matrice B transposée d'une matrice réelle A d'ordre N x M.

Solution :

Le transposé d'une matrice A d'ordre N x M est un matrice B d'ordre M x N.

Chaque ligne de A devient une colonne de B (ou chaque colonne de A devient une ligne pour B). Chaque case B[i, j] correspond à la case A[j, i] tel que : i=0, …, M-1 et j=0, …, N-1.

Algorithme/programme C :

Algorithme	Programme C

Algorithme Transposee ;
Variables
A, B : Tableau [0..99, 0..99] **de réel** ;
i, j, N, M : **Entier** ;
Début
// Entrées
Ecrire('Donner le nombre des lignes et des colonnes de A:') ;
Lire(N,M) ;
Ecrire('Donner les composantes de la matrice A :') ;
Pour i←0 **à** N-1 **faire**
Pour j←0 **à** M-1 **faire**
Lire(A[i,j]) ;
FinPour ;
FinPour ;
// Traitements
Pour i←0 **à** N-1 **faire**
Pour j←0 **à** M-1 **faire**
B[j, i]← A[i, j] ;
FinPour ;
FinPour ;
// Sorties
Ecrire('La matrice B Transposée de A est : ');
Pour i←1 **à** M **faire**
Pour j←1 **à** N **faire**
Ecrire(B[i,j]) ;
FinPour ;
FinPour ;
Fin.
	#include<stdio.h>
	int main() **{**
	float A[100][100], B[100][100];
	int i, j, N, M;
	// Entrées
	printf("Donner la dimension de la matrice A :");
	scanf("%d %d", &N,&M);
	printf("Donner les composantes de la matrice A : \n");
	for(i=0;i<N;i++) {
	for(j=0;j<M;j++) {
	scanf("%f",&A[i][j]);
	}
	}
	// Traitements
	for(i=0;i<N;i++) {
	for(j=0;j<M;j++) {
	B[j][i]=A[i][j];
	}
	}
	// Sorties
	printf("La matrice B transposée de la matrice A est : \n");
	for(i=0;i<M;i++) {
	for(j=0;j<N;j++) {
	printf("%.2f",B[i][j]);
	}
	printf("\n");
	}
	return 0;
	}
	Une autre méthode :
	for(i=0;i<M;i++)
	for(j=0;j<N;j++)
	B[i][j]=A[j][i];

Exercice N°03 : Somme de deux matrices

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

Solution :

Pour pouvoir réaliser la somme de deux matrices réelles, A et B, une condition nécessaire doit être vérifiée : A et B doivent être de même taille. Ainsi, si A est d'ordre NxM, alors B est aussi d'ordre NxM. Par conséquent, la matrice C, la somme des deux matrices, est aussi d'ordre NxM.

Algorithme/programme C :

Algorithme	Programme C
Algorithme Somme_deux_matrices ; **Variables** A, B, C : Tableau [0..99, 0..99] **de réel** ; i, j, N, M : **Entier** ; **Début** // Entrées	#include<stdio.h> int main() **{** **float** A[100][100], B[100][100], C[100][100]; **int** i, j, N, M; // Entrées

```
Ecrire('Donner le nombre des lignes et des colonnes :') ;
Lire(N,M) ;
Ecrire('Donner les composantes de la matrice A :') ;
Pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    Lire(A[i,j]) ;
  FinPour ;
FinPour ;

Ecrire('Donner les composantes de la matrice B :') ;
Pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    Lire(B[i,j]) ;
  FinPour ;
FinPour ;


// Traitements
Pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    C[i, j]← A[i,j]+B[i,j]  ;
  FinPour ;
FinPour ;

// Sorties
Ecrire(' L''affichage de la matrice C : ') ;
Pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    Ecrire( C[i,j]) ;
  FinPour ;
FinPour ;
Fin.
```

```
printf("Donner le nombre de lignes et de colonnes :");
scanf("%d %d", &N,&M);
printf("Donner les composantes de la matrice A : \n");
for(i=0;i<N;i++)  {
for(j=0;j<M;j++)  {
        scanf("%f",&A[i][j]);
  }
  }
    printf("Donner les composantes de la matrice B : \n");
for(i=0;i<N;i++)  {
for(j=0;j<M;j++)  {
        scanf("%f",&B[i][j]);
  }
  }
    // Traitements
      for(i=0;i<N;i++) {
        for(j=0;j<M;j++)  {
          C[i][j]=A[i][j]+B[i][j];
  }
  }
  // Sorties
  printf("L'affichage de la matrice C : \n");
for(i=0;i<N;i++) {
  for(j=0;j<M;j++) {
    printf("%.2f  ",C[i][j]);
  }
  printf("\n");
  }
return 0;
}
```

Exercice N°04 : Matrice symétrique

Soit A une matrice carrée de taille N x N et de type réel. Ecrire un programme C qui permet de vérifier si la matrice A est symétrique.

Rappel : Une matrice A est symétrique si A[*i*, *j*] = A[*j*, *i*] pour tout *i* et *j*.

Solution :

Rappel : A est symétrique si A[*i*, *j*] = A[*j*, *i*] pour tout *i* et *j*.

Les étapes à suivre :

- D'abord supposer que A est symétrique (Test = True)
- Ensuite, comparer chaque case A[*j*, *i*] avec la case A[*j*, *i*].
- Si elles sont différentes alors affecter la valeur False à la variable Test.
- A la fin, il suffit de voir la valeur de Test pour savoir si la matrice A est symétrique ou non.

Algorithme/programme C :

Algorithme	Programme C

Algorithme Matrice_Symetrique;
Variables
A : **Tableau** [0..9,0..9] **de réel**;
N, i, j : **entier** ;
Test : **booléen** ;
 	
Début
// Entrées
Ecrire('Donner la dimension de la matrice carrée A :');
Lire(N) ;
Ecrire('Donner les composantes de la matrice A : ');
Pour 0←1 **à** N-1 **faire**
Pour j←0 **à** N-1 **faire**
Lire(A[i, j]) ;
FinPour;
FinPour;
 	
// Traitements
Test ← Vrai ;
 	
Pour i←0 **à** N-1 **faire**
Pour j←0 **à** N-1 **faire**
Si (A[i,j] <> A[j,i]) **alors**
Test ← Faux ;
FinSi;
FinPour;
FinPour;
 	
// Sorties
Si (Test = Vrai) **alors**
Ecrire('La matrice A est symétrique')
Sinon
Ecrire('La matrice A n''est pas symétrique') ;
FinSi ;
Fin.
	#include<stdio.h>
	int main()
	{
	float A[100][100];
	int i, j, N;
	int Test;
	
	// Entrées
	printf("Donner la dimension de la matrice A :");
	scanf("%d", &N);
	printf("Donner les composantes de la matrice A : \n");
	for(i=0;i<N;i++) {
	for(j=0;j<N;j++) {
	scanf("%f",&A[i][j]); }
	}
	
	// Traitements
	Test=1;
	for(i=0;i<N;i++) {
	for(j=0;j<N;j++) {
	if (A[i][j]!=A[j][i]) {
	Test=0;
	}
	}
	}
	
	// Sorties
	if(Test==1) {
	printf("La matrice A est symétrique "); }
	else {
	printf("La matrice A n'est pas symétrique "); }
	return 0;
	}

<u>Exercice N°05 :</u> Produit d'une matrice par un vecteur

Soit A une matrice de type réel et d'ordre N x M. Ecrire un algorithme/programme PASCAL qui permet de calculer le produit de la matrice A par un vecteur V de type réel et de taille M.

Solution :

Algorithme/programme C :

Algorithme	Programme C

Algorithme Produit-Matrice_Vecteur ;
Variables
A : Tableau [0..99, 0..99] **de réel** ;
V, P : Tableau [0..99] **de réel** ;
N, M, i, j : **entier** ;
Début
// Entrées
Ecrire('Donner la dimension de la matrice A :');
Lire (N, M);
Ecrire('Donner les composantes de la matrice A : ');
pour i ←0 **à** N-1 **faire**
Pour j ←0 **à** M-1 **faire**
Lire(A[i, j]) ;
FinPour ;
FinPour ;
Ecrire('Donner les composantes du vecteur V : ');
pour i ← 0 **à** M-1 **faire**
Lire(V[i]) ;
FinPour ;
// Traitement
pour i ← 0 **à** N-1 **faire**
P[i] ← 0 ;
Pour j=0 **à** M-1 **faire**
P[i] ← P[i] + A[i, j]*V[j] ;
FinPour;
FinPour ;
// Sortie
Ecrire('Le résultat de produit :') ;
pour i ← 0 **à** N-1 **faire**
Ecrire(P[i]) ;
FinPour
Fin.
	#include<stdio.h>
	int main()
	{
	float A[100][100];
	float V[100], P[100];
	int N, M, i, j;
	
	// Entrées
	printf("Donner la dimension de la matrice A : ");
	scanf("%d %d", &N, &M);
	printf("Donner les composantes de la matrice A : \n");
	for (i=0;i<N;i++)
	for (j=0;j<M;j++)
	{
	scanf("%f", &A[i][j]);
	}
	printf("Donner les composantes du vecteur V : \n");
	for (i=0;i<M;i++) {
	scanf("%f", &V[i]);
	}
	// Traitement
	for (i=0;i<N;i++) {
	P[i]=0;
	for (j=0;j<M;j++) {
	P[i]=P[i]+A[i][j]*V[j];
	}
	}
	// Sortie
	printf("Le résultat de produit : \n ");
	for (i=0;i<N;i++)
	printf("%.2f ", P[i]);
	return 0;
	}