République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Abderrahmane Mira - Bejaia

Faculté de Technologie

Département de Technologie



Polycopié de cours

Première Année Technologie

Structure des Ordinateurs et Applications

Enseignant :

Dr. OUARET Ahmed

Année Universitaire : **2026/2027**

A propos de ce cours

Ce support de cours est destiné au module **Structure des ordinateurs et applications**, proposé aux étudiants de première année Licence (L1) dans le cadre du système LMD, ainsi qu'aux étudiants de première année du cycle Ingénieur. Il présente de manière progressive et structurée les fondements de la science informatique : notion de traitement automatique de l'information, évolution historique de l'informatique et des ordinateurs, architecture simplifiée d'un ordinateur, composition matérielle d'un système informatique, codage de l'information, ainsi que les expressions arithmétiques, relationnelles et logiques.

La seconde partie de ce cours est consacrée à l'initiation à la programmation à travers l'algorithmique, étape fondamentale pour tout futur programmeur, puis à la mise en œuvre pratique via le langage de programmation C. Les thèmes abordés incluent : la notion d'algorithme, les étapes de conception algorithmique, les entrées et sorties, les constantes, variables et types de données scalaires, les instructions de base, les instructions conditionnelles et les structures itératives, ainsi que leur traduction en langage C.

À l'issue de ce cours, l'étudiant sera capable d'écrire ses propres algorithmes et programmes en C pour résoudre des problèmes de complexité moyenne.

Pré-requis

Pour suivre ce cours dans de bonnes conditions, l'étudiant doit disposer de connaissances de base en :

- Culture générale en informatique
- Résolution de problèmes et raisonnement logique
- Notions élémentaires sur les langages de programmation

Les compétences visées

À l'issue de cet enseignement, l'étudiant sera capable de :

- Comprendre les concepts fondamentaux de l'architecture des ordinateurs
- Maîtriser les bases de l'algorithmique et de la conception de programmes
- Écrire et exécuter des programmes simples en langage C
- Développer une approche méthodique et rigoureuse dans la résolution de problèmes informatiques

Contenu et organisation de l'enseignement

Cours : 1h30 par semaine

TP : 1h30 par semaine

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

Table des matières

Introduction 1

Chapitre I : Introduction à l'informatique

I.1. Objectif du chapitre 3
I.2. Définition de l'informatique 3
I.2.1. Role de l'informatique 3
I.3. Evolution de l'informatique et des ordinateurs 4
I.3.1. Evolution de l'informatique 4
I.3.2. Evolution des ordinateurs 5
I.3.2.1. Définition de l'ordinateur 5
I.3.2.2. Générations des ordinateurs 5
I.4. Architecture et fonctionnement d'un ordinateur 6
I.5. Composition d'un ordinateur (Matériel & Logiciel) 8
I.5.1. Partie matérielle (Hardware) 8
I.5.2. Partie logicielle (Software) 11
I.6. Les systèmes de codage des informations 12
I.6.1. Les systèmes de numérotation 12
I.6.2. Conversion d'un nombre d'un système à un autre 14
I.6.3. Le code D.C.B. (Decimal coded Binary – Décimal codé binaire) 18
I.6.4. La codification Alphanumériques 18
I.7. Les expressions arithmétiques, relationnelles et logiques 19
I.7.1. Expressions arithmétiques 19
I.7.2. Expressions relationnelles 20
I.7.3. Expressions logiques (booléennes) 20
I.7.4. Les fonctions 21
I.7.5. La priorité dans les expressions 22
I.7.6. Evaluation des expressions 22
Série de TP N°01 24
Corrigé de la série de TP N°01 26
Exercices supplémentaires sur la série de TP N°01 30

Chapitre II : Notion d'algorithme et de programme

II.1. Objectif du chapitre 32

II.2. Concept d'un algorithme 32
II.3. La démarche et analyse d'un problème 33
II.4. Représentation des algorithmes 33
II.4.1. Représentation par organigramme 33
II.4.1.1. Symboles des organigrammes 34
II.4.2. Représentation par pseudocode 35
II.5. Structure d'un algorithme/programme 35
II.5.1. L'Entête 36
II.5.2. La partie déclarative 37
II.5.3. Le coprs de programme 40
II.6. Types d'instructions 40
II.6.1. Instructions d'Entrées/Sorties (Lecteure/Ecriture) 40
II.6.1.1. Entrées (Lecteure) 40
II.6.1.2. Sorties (Ecriture) 41
II.6.2. Instruction d'afectation 42
II.6.3. Structures de contrôles 51
II.6.3.1. Structures de contrôle conditionnelle 51
II.6.3.2. Structure de contrôle répétitives 55
Série de TP N°02 61
Corrigé de la série de TP N°02 62
Exercices supplémentaires sur la série de TP N°02 66
Série de TP N°03 67
Corrigé de la série de TP N°03 69
Exercices supplémentaires sur la série de TP N°03 80
Série de TP N°04 81
Corrigé de la série de TP N°04 82
Exercices supplémentaires sur la série de TP N°04 93
Bibliographie 94

Introduction

L'informatique, contraction d'information et automatique, est la science du traitement automatique de l'information. Depuis son émergence au milieu du 20ème siècle, elle n'a cessé d'évoluer à un rythme soutenu, étendant progressivement son champ d'application bien au-delà du simple calcul. Elle englobe aujourd'hui des domaines aussi variés que l'automatisation des processus, le contrôle et la commande des systèmes, la communication et le partage de l'information.

Tout système informatique repose sur deux composantes fondamentales et complémentaires : le matériel (hardware), qui désigne l'ensemble des éléments physiques du système tels que l'unité centrale, la mémoire et les organes d'entrées-sorties, et le logiciel (software), qui correspond à l'ensemble des instructions et des programmes définissant le comportement de la machine.

Ce cours intitulé **« Structure des ordinateurs et application »** vise à fournir une compréhension solide de l'architecture des ordinateurs et des principes fondamentaux de la programmation. Il est structuré en plusieurs unités d'apprentissage couvrant les composants matériels d'un ordinateur, les systèmes de codage de l'information, la conception d'algorithmes ainsi que la méthodologie d'analyse et de résolution de problèmes.

Le langage de programmation C est utilisé comme support pratique tout au long de ce cours. Reconnu pour sa proximité avec le matériel, sa performance et sa portabilité, il représente un outil particulièrement adapté pour illustrer concrètement les concepts abordés et initier l'étudiant à une programmation rigoureuse et structurée.

L'ensemble de ces composantes est illustré par la figure 1, qui place les algorithmes au cœur du processus informatique, en interaction avec les données, les machines, les langages de programmation et les interfaces avec le monde extérieur.

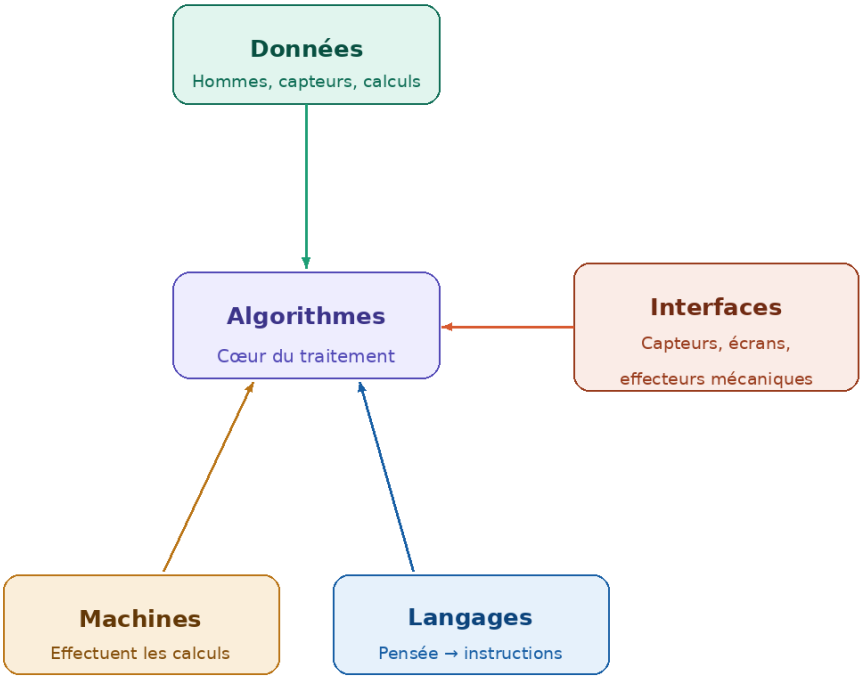


Figure 1 : Les composantes de l'informatique

CHAPITRE I

Introduction à l'informatique

Chapitre I : Introduction à l'informatique

I.1. Objectif du chapitre

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

- Se familiariser avec l'environnement informatique et ses concepts fondamentaux
- Identifier et décrire les principaux composants matériels et logiciels d'un ordinateur
- Comprendre et appliquer les systèmes de codage de l'information
- Évaluer et manipuler les expressions arithmétiques, relationnelles et logiques

I.2. Définition de l'Informatique

L'informatique est la science qui s'occupe du traitement automatique de l'information à l'aide de systèmes électroniques programmables. Le terme « informatique » est formé par la contraction des mots « information » et « automatique », introduit dans les années 1960 pour désigner cette nouvelle discipline.

Elle englobe un ensemble de théories, de méthodes et de techniques liées à la collecte, au stockage, au traitement et à la transmission de l'information, et trouve des applications dans de nombreux domaines tels que la médecine, l'industrie, l'éducation, la communication et la recherche scientifique.

I.2.1. Rôle de l'informatique

L'informatique joue un rôle fondamental dans le monde moderne. Elle intervient principalement à trois niveaux :

- La conception et la construction des ordinateurs : elle couvre le développement des composants matériels et logiciels qui constituent les systèmes informatiques.
- Le fonctionnement et la maintenance des ordinateurs : elle assure le bon fonctionnement des systèmes, leur fiabilité, leur sécurité et leur mise à jour continue.
- L'exploitation des ordinateurs : elle englobe l'utilisation des outils informatiques dans les différents domaines d'activités tels que la gestion, l'industrie, la médecine, l'enseignement et la recherche scientifique.

I.3. Evolution de l'informatique et des ordinateurs

I.3.1. Evolution de l'informatique

L'histoire de l'informatique est marquée par la volonté constante des hommes d'automatiser certaines tâches autrefois réalisées manuellement, en particulier les opérations de calcul. Cette quête d'automatisation a conduit, au fil des siècles, à des avancées technologiques majeures qui ont profondément transformé notre façon de traiter et d'échanger l'information.

Le tableau I.1 ci-dessous présente les principales étapes de cette évolution.

Tableau I.1 : Evolution de l'informatique

Date	Inventeur	Machine	Fonction
1642	Blaise Pascal	Pascaline	Première machine à calculer mécanique, capable d'effectuer des additions et des soustractions sur des nombres de six chiffres.
1822-1832	Charles Babbage	Machine à Différences *(Difference Engine)*	Conçue pour effectuer des calculs numériques complexes et générer automatiquement des tables numériques.
1887	Hermann Hollerith	Machine mécanographique (Calculateur à cartes perforées)	Développée pour accélérer le traitement des données lors du recensement américain de 1890, marquant l'invention de la carte perforée.
1945 (inauguré en 1946)	John P. Eckert & John W. Mauchly	**ENIAC** (Electronic Numerical Integrator and Computer)	Premier calculateur numérique électronique programmable. Composé de 18 000 tubes électroniques et pesant 30 tonnes, il était capable de multiplier deux nombres de 10 chiffres en moins d'une milliseconde.

1945	John Von Neumann	**EDVAC** (Electronic Discrete Variable Automatic Computer)	Premier ordinateur conçu sur le principe du programme enregistré en mémoire, posant les bases de l'architecture moderne des ordinateurs.

I.3.2. Evolution des ordinateurs

I.3.2.1. Définition de l'ordinateur

Un ordinateur est une machine automatique programmable, utilisée pour le traitement de l'information.

I.3.2.2. Générations des ordinateurs

- **1ère Génération (1945 - 1957) :** C'est les ordinateurs construits sur la base des "*tubes électroniques*" (aussi appelés "*tubes à vide*". **Exemple** : ENIAC (1945) et IBM 701 (1952).
- **2ème Génération (1958 - 1963) :** Elle est définie par l'invention du "*Transistor"* en 1947 (le transistor est un composant électronique capable de réguler le courant). Ces ordinateurs sont 100 fois plus rapides que ceux de la 1ère génération et consomment moins d'énergie électrique et sont moins volumineux (occupent moins d'espace). **Exemples**: PDP I (1er ordinateur de 2ème génération (1960)), IBM 7030 (1961).
- **3ème Génération (1964 - 1971) :** Elle est définie après l'apparition du "*Circuit Intégré*" en 1958, les ordinateurs de cette génération utilisent les transistors et les circuits intégrés. **Exemple**: IBM 360 (1964), DEC PDP8 (1964).
- **4ème Génération (1971 - 1980) :** Elle survient après l'invention du "*microprocesseur*". Le premier microprocesseur est fabriqué par la société INTEL en 1971. On parle, désormais, de microordinateurs. **Exemple**: MICRAL 8008 (1973), ALTAIR 8800 (1975).
- **5ème Génération (1980 - 2000) :** Les ordinateurs entrent dans les foyers, on parle alors d'informatique familiale (c'est la naissance des ordinateurs personnels (PC)). **Exemple** : IBM PC (1981), Intel Pentium (1993).
  Aussi, les systèmes d'exploitation MS-DOS et Mac OS font leur apparition.

- **6ème Génération (2000 – aujourd'hui) :** On assiste, dans cette génération, à l'apparition des ordinateurs portables et des réseaux sans fil.
  Actuellement l'évolution des ordinateurs tend à exploiter le laser, les fibres optiques, la biochimie et même l'informatique moléculaire.

I.4. Architecture et fonctionnement d'un ordinateur

L'architecture et le fonctionnement d'un ordinateur constituent des notions fondamentales pour la compréhension d'un système informatique.

L'architecture d'un ordinateur désigne l'organisation générale ainsi que la structure interne de ses différents composants matériels, tels que le processeur, la mémoire centrale, les unités de stockage et les périphériques d'entrée/sortie. Cette organisation permet d'assurer le traitement, le stockage et l'échange des informations nécessaires à l'exécution des différentes tâches informatiques.

Le fonctionnement d'un ordinateur repose sur l'interaction entre ces composants, qui coopèrent afin d'exécuter les instructions des programmes et de traiter les données de manière automatique et rapide.

La Figure I.1 illustre l'architecture générale d'un ordinateur.

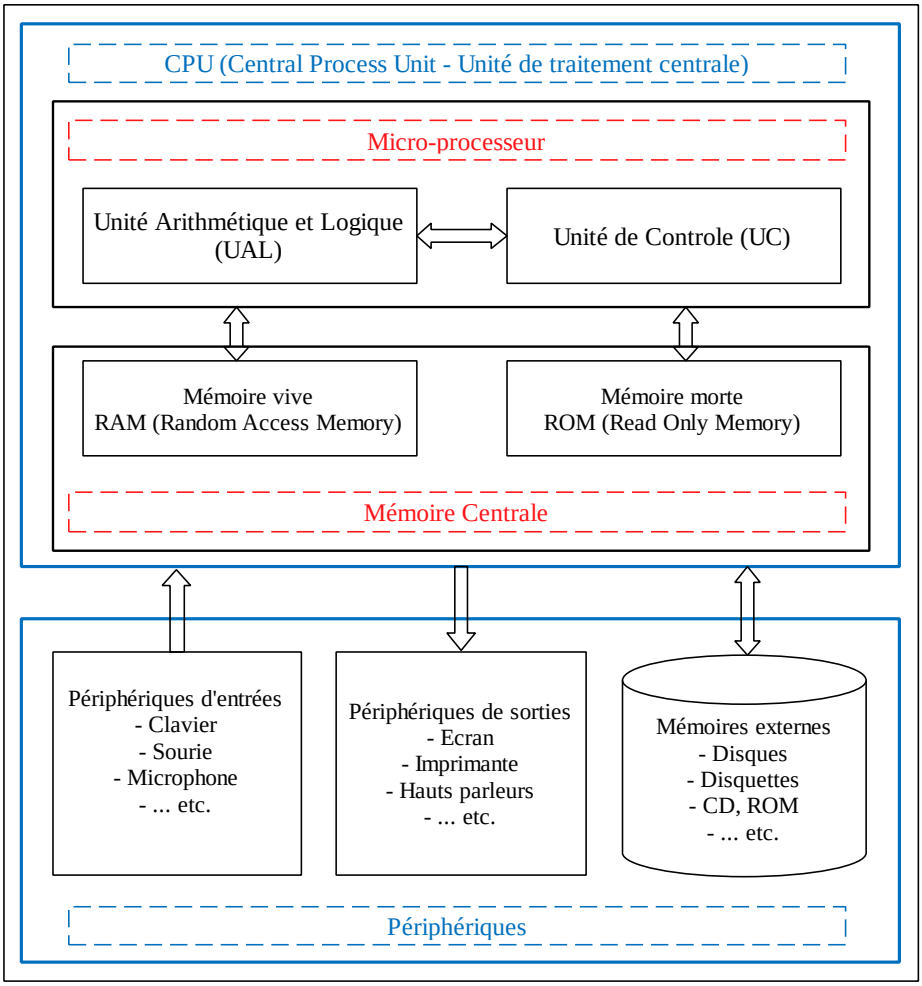


Figure I.1 : Architecture générale d'un ordinateur

CPU : Central Process Unit (Unité de traitement centrale)
UAL : Unité Arithmétique et Logique

UC : Unité de Contrôle ou de commande
MC : Mémoire Centrale

Un ordinateur est constitué, d'une manière générale, d'une Unité Centrale de Traitement, d'une mémoire centrale et d'un ensemble de périphériques.

a) L'Unité Centrale de Traitement / Processeur

Le processeur, également appelé microprocesseur ou CPU *(Central Processing Unit)*, constitue le cœur de l'ordinateur. Il est chargé d'interpréter et d'exécuter les instructions des programmes. Il comprend deux unités fondamentales :

- **L'Unité de Contrôle (UC)** : responsable de la lecture et du décodage des instructions stockées en mémoire centrale, elle coordonne et synchronise l'ensemble des opérations du processeur.
- **L'Unité Arithmétique et Logique (UAL)** : également appelée unité de traitement, elle exécute toutes les opérations arithmétiques, logiques et relationnelles. Elle assure également les échanges de données (réception et transmission) avec la mémoire vive.

Ces deux unités communiquent non seulement avec la mémoire centrale, mais également avec les différents périphériques du système.

b) La Mémoire Centrale (ou Principale)

La mémoire centrale est constituée de deux types de mémoires complémentaires :

- **La RAM** *(Random Access Memory — Mémoire Vive)* : elle assure le stockage temporaire des données et des programmes en cours d'exécution. Elle est dite **volatile**, car son contenu est perdu en cas de coupure d'alimentation électrique.
- **La ROM** *(Read Only Memory — Mémoire Morte)* : contrairement à la RAM, elle est **non volatile** et conserve son contenu même en l'absence d'alimentation. Elle ne peut être que lue (ou très rarement modifiée) et contient les programmes indispensables au démarrage du système, avant le chargement du système d'exploitation en RAM.

La capacité de la mémoire centrale se mesure actuellement en milliers de mégaoctets. Les principales unités de mesure sont les suivantes :

Unité	**Équivalence**
1 bit	Unité de base (valeur : 0 ou 1)
1 octet	8 bits
1 Kilo-octet (Ko)	2^{10} octets = 1 024 octets
1 Méga-octet (Mo)	2^{20} octets = 1 024 Ko
1 Giga-octet (Go)	2^{30} octets = 1 024 Mo

Remarque : Le **bit** est l'abréviation de *"binary digit"* (chiffre binaire). C'est l'unité élémentaire d'information dans tout système numérique, pouvant prendre deux valeurs : **0** ou **1**.

c) Les Périphériques

Les périphériques désignent l'ensemble des équipements connectés à l'unité centrale permettant les échanges d'informations entre l'ordinateur et le monde extérieur. Ils se classent en trois catégories :

- **Périphériques d'entrée** : ils permettent de transmettre des données vers l'ordinateur. Exemples : le clavier, la souris, le microphone, la webcam, le scanner, etc.
- **Périphériques de sortie** : ils permettent à l'ordinateur de restituer des données vers l'extérieur. Exemples : l'écran, l'imprimante, les haut-parleurs, le vidéoprojecteur, etc.
- **Périphériques d'entrée/sortie** : ils permettent à la fois de transmettre et de recevoir des données. Également appelés **périphériques de stockage** ou **mémoires auxiliaires**, ils assurent la conservation permanente des données. Exemples : la clé USB, le disque dur externe, le CD-ROM, la carte mémoire, etc.

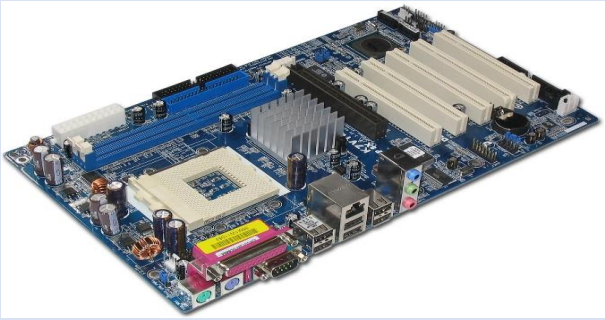
I.5. Composition d'un ordinateur (Matériel et Logiciel)

Un ordinateur est constitué de deux parties essentielles et complémentaires : le **matériel** *(hardware)* et le **logiciel** *(software)*. Le matériel désigne l'ensemble des composants physiques de la machine, tandis que le logiciel regroupe l'ensemble des programmes et instructions qui permettent de les exploiter.

I.5.1. Partie matérielle (Hardware)

Le matériel d'un ordinateur désigne l'ensemble des composants physiques qui constituent la machine, incluant les éléments électroniques, mécaniques et électriques nécessaires à son fonctionnement. On distingue généralement les principales parties suivantes :

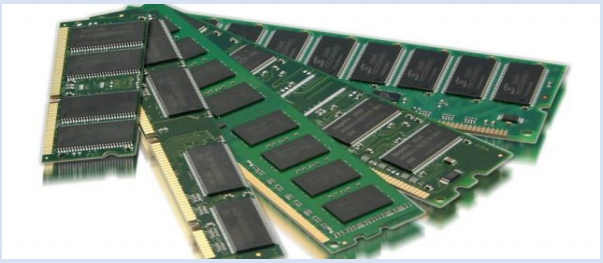
- Carte Mère :

Carte principale sur laquelle on retrouve tous les composants nécessaires au fonctionnement d'un ordinateur, dont le processeur, la mémoire centrale, les bus et les connecteurs d'extension destinés à recevoir des cartes d'extension (carte graphique externe, carte son, carte réseau, etc.).	

- Processeur :

Processeur dont tous les éléments sont miniaturisés et rassemblés sur une puce en un seul ou en plusieurs circuits intégrés. Le processeur remplit les fonctions d'unité centrale dans un micro-ordinateur. Un ventilateur est utilisé pour réduire sa température souvent élevée et réduire les risques de surchauffe.	

- Mémoire Vive :

Mémoire du travail du processeur, tout programme exécuté doit être chargé sur la mémoire vive, dans laquelle les instructions peuvent lire ou écrire des données.	

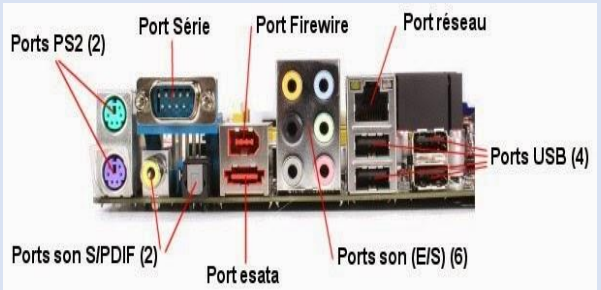
- Disque Dur :

Support de mémoire de masse d'accès rapide qui est composé d'un ou de plusieurs disques magnétiques généralement solidaires d'une mécanique, et ayant une très grande capacité de stockage.	

- Carte graphique :

Carte d'extension servant à stocker et à interpréter les images reçues par l'ordinateur et à mettre à jour continuellement l'écran du moniteur. Elle n'est pas obligatoire sur certains ordinateurs qui en possèdent une incorporée dans la carte mère. Mais pour de meilleurs performances il est conseillé d'en installer une.	

- Connecteurs et Ports :

Ces ports permettent de relier des périphériques externes (clavier, souris, etc.) au boîtier de l'unité centrale. Il y a plusieurs type de ports : Port USB, port parallèle, port série, etc.	

- Écran :

Périphérique de sortie, il permet d'afficher les toute sorte de données. Il y a deux mode d'affichage : mode texte et mode graphique. Il y a des écran tactile permettant d'émettre des commandes via de simples touches de doigts, dans ce cas, l'écran représente aussi un périphérique d'entrée.	

- Clavier :

Périphérique permettant d'entrer des données sous forme textuelle vers le CPU. Donc il représente un périphérique de sortie.	

- **Sourie :**

Permet de lancer des commandes à travers de simples cliques : clique gauche pour la sélection, double clique gauche pour exécuter un programme ou ouvrir un fichier, clique droit affiche le menu contextuel, etc.	

- **Imprimante :**

Périphérique de sortie permettant d'afficher des données (textuelles, images ou graphes) sur du papier.	

La figure I.2 résume la partie hardware de l'ordinateur

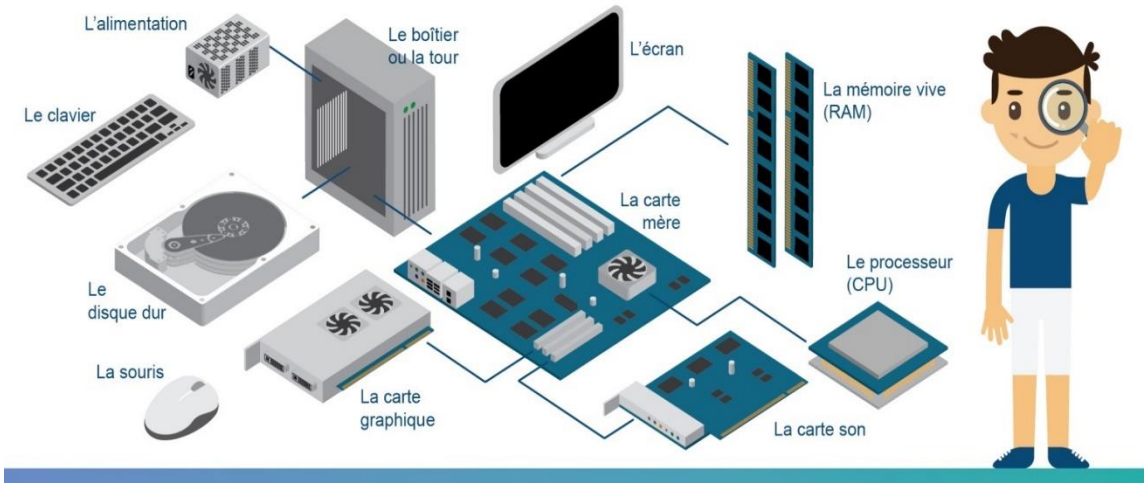


Figure I.2 : Partie hardware de l'ordinateur

I.5.2. Partie logicielle (Software)

Le logiciel désigne l'ensemble des programmes, instructions et données qui permettent à l'ordinateur de fonctionner et d'accomplir des tâches spécifiques. Il constitue la composante immatérielle du système informatique, par opposition au matériel. On distingue trois grandes catégories de logiciels :

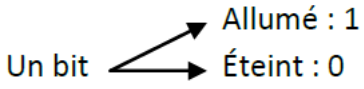
- **Logiciels de base (Système d'exploitation)** : Le système d'exploitation est le logiciel fondamental qui assure la gestion des ressources matérielles de l'ordinateur et sert d'interface entre l'utilisateur et la machine. Sans lui, aucun autre logiciel ne peut fonctionner. Exemples : Windows, Linux, macOS, Android, etc.
- **Langages de programmation** : Ce sont des outils formels utilisés par les développeurs pour concevoir et écrire des programmes informatiques. Chaque langage possède sa propre syntaxe et est adapté à des domaines d'application spécifiques. Exemples : C/C++, Java, Python, Pascal, Fortran, MATLAB, etc.
- **Logiciels d'application** : Ce sont des programmes conçus pour répondre à des besoins spécifiques de l'utilisateur final. Ils s'exécutent au-dessus du système d'exploitation et couvrent un large éventail de fonctionnalités : traitement de texte, tableur, retouche d'image, conception assistée par ordinateur, navigation sur internet, lecture multimédia, jeux, etc. Exemples : Microsoft Office, Adobe Photoshop, AutoCAD, VLC Media Player, etc.

I.6. Les systèmes de codage des informations

Toute information traitée par un ordinateur, qu'elle soit numérique, textuelle, sonore, visuelle ou vidéo, est représentée sous forme de séquences de deux chiffres : **0** et **1**. Ces chiffres sont appelés **bits** *(BInary digiT)* et constituent la base de toute représentation numérique de l'information.

En effet, un ordinateur fonctionne sur le principe de deux états électriques distincts : la présence de courant (état 1) et l'absence de courant (état 0). C'est la seule forme d'information qu'un ordinateur est capable d'interpréter directement. Le bit représente ainsi l'unité élémentaire d'information en informatique, c'est-à-dire la plus petite unité de mesure existante dans un système numérique.

Par conséquent :



I.6.1. Les systèmes de numérotation

Définition : Un système de numérotation est un ensemble de règles et de symboles permettant de représenter tout entier naturel N dans une base b donnée. Tout système de numérotation est défini par deux éléments fondamentaux :

1. **La base (b) :** le nombre de symboles distincts utilisés dans le système.
2. **Les symboles :** l'ensemble des chiffres et/ou des lettres qui composent le système et permettent de représenter toute valeur numérique.

En informatique, quatre systèmes de numérotation sont principalement utilisés, présentés dans le tableau ci-dessous :

Système	Base	Symboles	Nombre de symboles du système
Décimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
Binaire	2	0, 1	2
Octal	8	0, 1, 2, 3, 4, 5, 6, 7	8
Hexadécimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	16

❖ **Remarque 1 :** Dans le système hexadécimal, des lettres sont utilisées pour représenter les valeurs supérieures à 9 : $(A)_{16} = (10)_{10}$, $(B)_{16} = (11)_{10}$, $(C)_{16} = (12)_{10}$, $(D)_{16} = (13)_{10}$, $(E)_{16} = (14)_{10}$, $(F)_{16} = (15)_{10}$.

❖ **Remarque 2 :** Le nombre de symboles d'un système de numérotation est toujours égal à sa base : Nombre de symboles = b.

- **Notation**

Un nombre quelconque Nb exprimé dans une base b sera noté comme suit :

$$Nb = (a_{n-1}\ a_{n-2} \dots\ a_1\ a_0)_b$$

avec :

b : la base du système de numérotation

a_i : symbole du système de numérotation, *i=0,…,n-1 et* $a_i < b$

Exemples :

$Nb_1 = (1995)_{10}\ avec\ a_3{=}1, a_2 = 9, a_1 = 9, a_0 = 5$

$Nb_2 = (243)_8\ avec\ \ a_2 = 2, a_1 = 4, a_0 = 3$

$Nb_3 = (1011010)_2\ avec\ a_6 = 1, a_5 = 0, a_4 = 1, a_3{=}1, a_2 = 0, a_1 = 1, a_0 = 0$

$Nb_4 = (BAC)_8\ avec\ \ a_2 = B, a_1 = A, a_0 = C$ Cette notation est erronée, car le nombre contient un symbole supérieur ou égale à la base

$Nb_5 = (248)_8$ Cette notation est erronée, car le nombre contient un symbole supérieur ou égale à la base

❖ **Remarques :**

- Quand la base n'est pas mentionnée, on considère qu'on est en base 10.
- Dans une base b, nous avons b chiffres : 0, 1, 2, …., (b-1).

I.6.2. Conversion d'un nombre d'un système à un autre

a) Conversion de la base b (base 2, 8, 16,…etc) vers la base 10

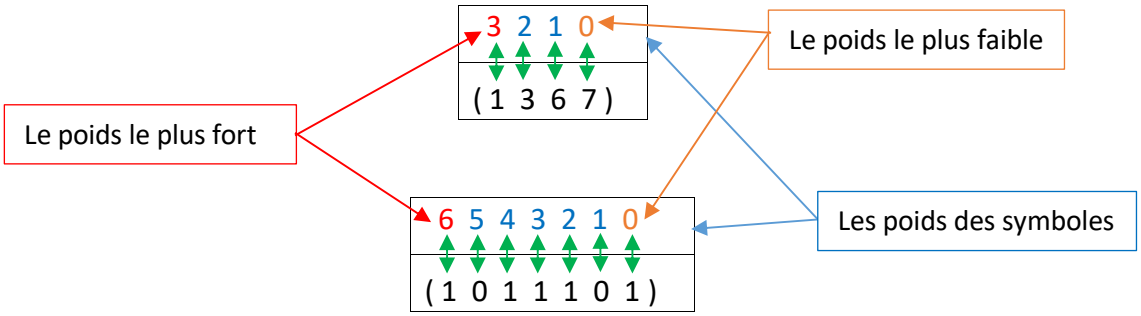
Pour convertir un nombre $Nb = (a_{n-1}\ a_{n-2} \dots\ a_1\ a_0)_b$ de la base b vers la base 10, on effectue le calcul suivant :

$$(Nb)_b = (a_{n-1} * b^{n-1} + a_{n-2} * b^{n-2} \dots + a_1 * b^1 + a_0 * b^0)_{10}$$

$$(Nb)_b = \sum_{i=0}^{n-1} a_i * b^i$$

Elle correspond à la somme des produits de chaque symbole du nombre par le poids correspondant.

Le poids des symboles : il s'obtient en numérotant les symboles à partir de la droite vers la gauche, en commençant du 0.



Exemple 1 :

$(\mathbf{1367})_{\mathbf{8}} = (?)_{\mathbf{10}}$

$(1367)_8 = (1 * 8^3 + 3 * 8^2 + 6 * 8^1 + 7 * 8^0)_{10}$

$(1367)_8 = (759)_{10}$

Exemple 2 :

$(\mathbf{1011101})_{\mathbf{2}} = (?)_{\mathbf{10}}$

$(1011101)_2 = (1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0)_{10}$

$(1011101)_2 = (93)_{10}$

Exemple 3 :

$(\boldsymbol{2B3})_{\boldsymbol{16}} = (?)_{\boldsymbol{10}}$

$(2B3)_{16} = (2 * 16^2 + B * 16^1 + 3 * 16^0)_{10}$

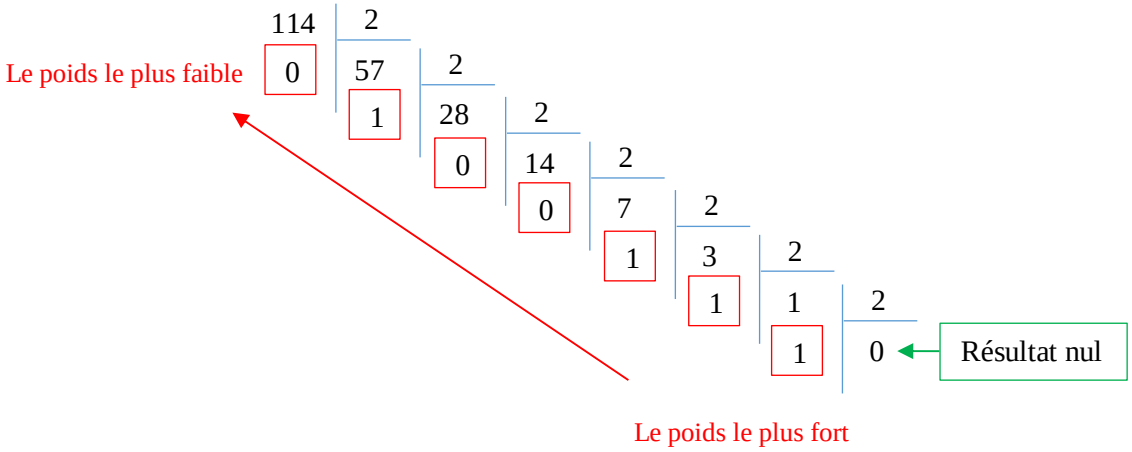
$(2B3)_{16} = (691)_{10}$

b) Conversion de la base 10 vers la base b (base 2, 8, 16,…etc)

Soit Nb un nombre exprimé dans la base 10, pour trouver son équivalent en base b, on applique la méthode des **divisions successives sur b**, jusqu'à l'obtention d'un résultat nul. Puis, on récupère les restes des divisions dans le sens inverse, i.e. le dernier reste trouvé représentera le poids le plus fort et le premier reste trouvé sera le poids le plus faible.

Exemple 1 :

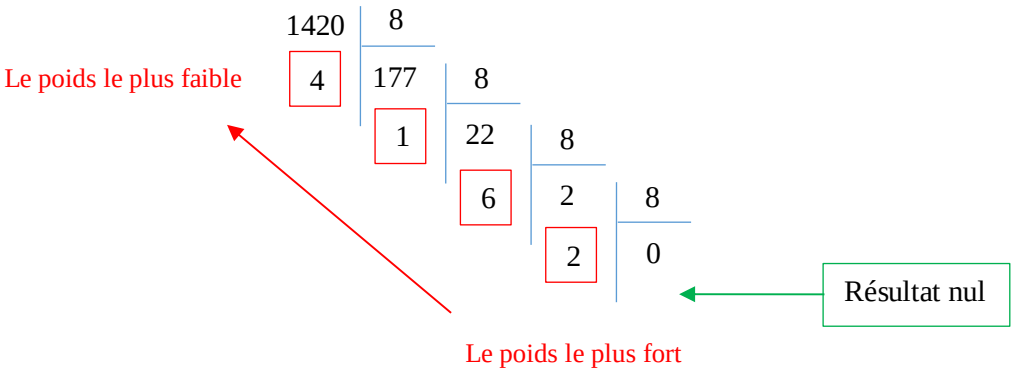
- $(\boldsymbol{114})_{\boldsymbol{10}} = (?)_{\boldsymbol{2}}$



$(114)_{10} = (1110010)_2$

Exemple 2 :

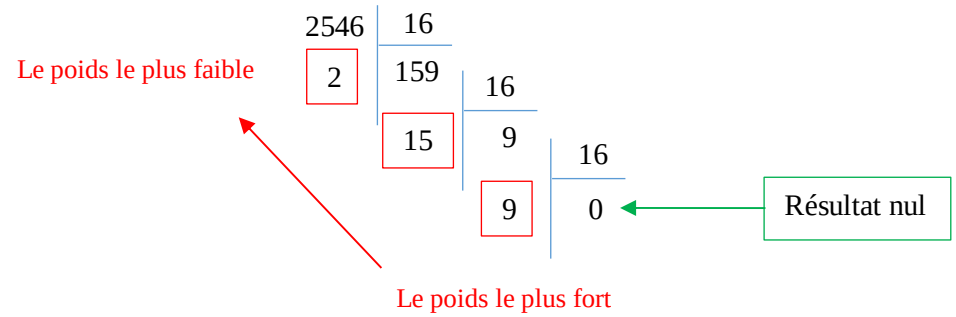
- $(\boldsymbol{1420})_{\boldsymbol{10}} = (?)_{\boldsymbol{8}}$



$(1420)_{10} = (2614)_8$

Exemple 3 :

- $(\mathbf{2546})_{\mathbf{10}} = (?)_{\mathbf{16}}$



$(2546)_{10} = (9F2)_{16}$

c) Conversion de la base 8 vers la base 2

Pour convertir un nombre Nb exprimé en base 8 vers la base 2, nous procédons comme suit:

$$8 = 2^3$$

Il faut donc utiliser **3** bits pour exprimer un seul chiffre octal en binaire.

La représentation des chiffres de la base 8 vers le binaire est comme suit :

- $(7)_8 = (1 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (111)_2$
- $(4)_8 = (1 * 2^2 + 0 * 2^1 + 0 * 2^0)_2 = (100)_2$
- $(3)_8 = (0 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (011)_2$

Chiffre en octal	Chiffre équivalent en binaire (2^2 2^1 2^0)
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Exemple :

- $(743)_8 = (111\ \ 100\ \ 011)_2$

d) Conversion de la base 16 vers la base 2

Pour convertir un nombre Nb exprimé en base 16 vers la base 2, nous procédons comme suit:

$$16 = 2^4$$

Il faut donc utiliser **4** bits pour exprimer un seul chiffre hexadécimal en binaire.

La représentation des chiffres de la base 16 vers le binaire est comme suit :

Chiffre en hexadécimal	Chiffre équivalent en binaire (2^3 2^2 2^1 2^0)
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1

Chiffre en hexadécimal	Chiffre équivalent en binaire (2^3 2^2 2^1 2^0)
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

- $(7)_{16} = (0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (0111)_2$
- $(B)_{16} = (1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (1011)_2$
- $(3)_{16} = (0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (0011)_2$
- $(A)_{16} = (1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0)_2 = (1010)_2$

Exemple :

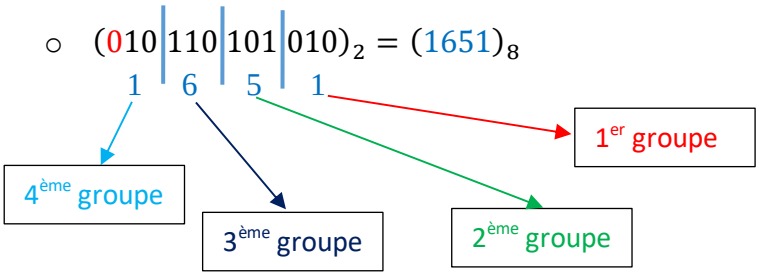
- $(7B3A)_{16} = (0111\ \ 1011\ \ 0011\ \ 1010)_2$

e) Conversion de la base 2 vers la base 8

Pour trouver l'équivalent d'un nombre binaire en octal, il suffit de former des **groupes de 3 bits** chacun (Puisque **8 = 2^3**), en commençant du poids le plus faible (à partir de la droite), si le dernier groupe formé possède moins de 3 bits, il suffit de rajouter des 0, puis calculer l'équivalent en octal de chaque groupe.

Exemple :

- $(10110101010)_2 = (?)_8$
- $(010|110|101|010)_2 = (1651)_8$



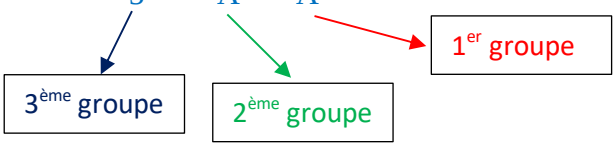
f) Conversion de la base 2 vers la base 16

Pour trouver l'équivalent d'un nombre binaire en Hexadécimal, il suffit de former des **groupes de 4 bits** chacun (Puisque **16 = 2^4**), en commençant du poids le plus faible (à partir

de la droite), si le dernier groupe formé possède moins de 4 bits, il suffit de rajouter des 0, puis calculer l'équivalent en Hexadécimal de chaque groupe.

Exemple :

- $(10110101010)_2 = (?)_{16}$
- $(0101\,|\,1010\,|\,1010)_2 = (5AA)_{16}$



g) Conversion de la base 16 vers la base 8

Pour convertir un nombre Nb exprimé en base 16 vers la base 8 ou vice versa, nous devons **passer par une base intermédiaire** tel que le **décimal** ou le **binaire**, mais le passage par le binaire est beaucoup plus simple.

Exemple :

- $(C9F)_{16} = (1100\ \ 1001\ \ 1111)_2$
- $(110\,|\,010\,|\,011\,|\,111)_2 = (6237)_8$

6 2 3 7

I.6.3. Le code D.C.B. (Decimal Coded Binary – Décimal codé binaire)

Le code DCB consiste à convertir chaque chiffre décimal en un nombre binaire sur 4 positions.

Exemple :

Le nombre décimal 378 est codé en DCB comme suit : 0101 0111 1000

Dans ce code, chaque chiffre décimal est remplacé par 4 chiffre binaires (comme dans le cas de hexadécimal).

I.6.4. La codification Alphanumériques

Les chiffres, lettres, signes de ponctuation, les symboles mathématiques, etc., sont représentées généralement en utilisant le code normalisé à 8 positions binaires. On utilise pour cela le code EBDIC (Extended Binary Coded Decimal International Code) ou le code ASCII (American Standard Code Information Interchange).

Exemples :

Tableau I.2 : Exemples de codification Alphanumériques

Lettre / Signe / Symbole	En EDCDIC	En ASCII
A (Majuscule)	11000001	10100001
D (miniscule)	10000010	11100100
=	01111110	01011101

Exercice : *(Systèmes de numérotation)*

Effectuer les conversions suivantes :

$2022 = (?)_2$;

$(1011001101)_2 = (?)_{10}$;

$(1011001101)_2 = (?)_8 = (?)_{16}$;

$(32103)_4 = (?)_2$;

$(37163)_8 = (?)_2$;

$(379)_{10} = (?)_{16}$;

$(3A2D)_{16} = (?)_{10}$;

$(4D5B)_{16} = (?)_8$

Corrigé de l'exercice : *(Systèmes de numérotation)*

Effectuer les conversions suivantes :

$2022 = (11111100110)_2$;

$(1011001101)_2 = (717)_{10}$;

$(1011001101)_2 = (1315)_8 = (2CD)_{16}$;

$(32103)_4 = (1110010011)_2$;

$(37163)_8 = (01111001110011)_2$ oubien $(11111001110011)_2$;

$(379)_{10} = (17B)_{16}$;

$(3A2D)_{16} = (14893)_{10}$;

$(4D5B)_{16} = (46533)_8$

I.7. Les expressions arithmétiques, relationnelles et logiques

I.7.1. Expressions arithmétiques

C'est une combinaison d'opérande(s) (valeur, constante, variable) et d'opération(s) arithmétiques. Le Tableau I.3 résume ces opérations.

Tableau I.3 : Les opérations arithmétiques

Opération	**Opérateur**
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo (reste de la division entière)	%

I.7.2. Expressions relationnelles

En langage C, les opérateurs relationnels permettent de comparer deux valeurs. Le résultat d'une comparaison est 1 si elle est vraie et 0 si elle est fausse. Le tableau I.4 résume les principaux opérateurs relationnels utilisés en programmation.

Tableau I.4 : Les opérateurs relationnels

Signification	**Opérateur**
Egal à	==
Strictement inférieur à	<
Inférieur ou égal à	<=
Strictement supérieur à	>
Supérieur ou égal à	>=
Différent de	!=

❖ **Remarque importante :**

- = est l'opérateur d'affectation : a = 5;
- == est l'opérateur de comparaison : a == 5

I.7.3. Expressions logiques (booléennes)

En langage C, les expressions logiques (booléennes) permettent de combiner ou de modifier des conditions. Elles renvoient 1 si le résultat est vrai et 0 s'il est faux.

Tableau I.5 : L'opérateur **ET logique (&&)**.

A	B	A **&&** B
False	False	False
False	True	False
True	False	False
True	True	True

Tableau I.6 : L'opérateur **OU logique (||)**

A	B	A || B
False	False	False
False	True	True
True	False	True
True	True	True

Tableau I.7 : L'opérateur **NON logique (!)**

A	! A
False	True
True	False

I.7.4. Les fonctions

Le Tableau I.8 présente une liste non-exhaustive de fonctions standards (ou prédéfinies) applicables sur des entiers ou des réels.

Tableau I.8 : Quelques fonctions standards en langage C

Fonction	**Bibliothèque**	**Description**	**Exemple**
abs(x)	<stdlib.h>	Valeur absolue d'un entier	abs(-5) → 5
sqrt(x)	<math.h>	Racine carrée	sqrt(25) → 5
pow(x,y)	<math.h>	Puissance x^y	pow(2,3) → 8
sin(x)	<math.h>	Sinus de x (en radians)	sin(x)
cos(x)	<math.h>	Cosinus de x (en radians)	cos(x)
tan(x)	<math.h>	Tangente de x (en radians)	tan(x)
exp(x)	<math.h>	Exponentielle e^x	exp(1)
log(x)	<math.h>	Logarithme népérien	log(x)
log10(x)	<math.h>	Logarithme décimal	log10(100) → 2
ceil(x)	<math.h>	Plus petit entier supérieur ou égal à x	ceil(3.2) → 4
floor(x)	<math.h>	Plus grand entier inférieur ou égal à x	floor(3.8) → 3
fabs(x)	<math.h>	Valeur absolue d'un réel	fabs(-3.5) → 3.5
fmod(x,y)	<math.h>	Reste de la division de deux réels	fmod(7.5,2) → 1.5

❖ **Remarque :** les fonctions de la bibliothèque <math.h> nécessitent généralement l'inclusion de cette bibliothèque au début du programme :

I.7.5. La priorité dans les expressions en langage C

Lorsqu'une expression contient plusieurs opérateurs, le langage C respecte un **ordre de priorité** pour déterminer dans quel ordre les opérations doivent être effectuées.

Tableau I.9 : Priorité des opérateurs

Priorité	Opérateurs	Description	Associativité
1	`() ; fonctions`	Parenthèses ; fonctions	Gauche → droite
2	`!`	NON logique	Droite → gauche
3	`* / %`	Multiplication, division, modulo	Gauche → droite
4	`+ -`	Addition, soustraction	Gauche → droite
5	`< <= > >=`	Opérateurs relationnels	Gauche → droite
6	`== !=`	Égalité, différence	Gauche → droite
7	`&&`	ET logique	Gauche → droite
8	`||`	OU logique	Gauche → droite
9	`= += -= *= /= %=`	Affectation	Droite → gauche

I.7.6. Évaluation des expressions

L'évaluation d'une expression consiste à calculer, au fur et à mesure, les résultats des calculs jusqu'à obtenir un résultat finale. Cela se fait en plusieurs étapes :

- Écrire l'expression sous forme linéaire (Il faut noter qu'en algorithmique, les expressions s'écrivent sous forme linéaire: $\frac{(x+z)}{(y*2)} \rightarrow (x+z)/(y*2)$;
- Remplacer les identifiants (c'est à dire les noms) des variables et des constantes par leurs valeurs ;
- Évaluer (Calculer) étape par étape chacune des sous-expressions en commençant par les sous-expressions qui sont dans les parenthèses les plus internes.
- Indiquer à chaque calcul, le rang d'évaluation.

❖ **Remarque :** Si les opérateurs ont le même rang de priorité, l'évaluation se fait de gauche à droite.

Exercice N°02 :(Evaluation des expressions)

Evaluer les expressions suivantes en respectant l'ordre de priorité des opérateurs:

Expression1 : E1=a + b*c - d/e ; ***avec***: a=10, b=4, c=3, d=20, e=5;

Expression2: E2= ((a + b) – c) % 3 == 0 && c* a – b < c || ! (a >= c) ; ***avec:*** a =6;b= 10;c= 4.

Exercice N°02 :(Evaluation des expressions)
Evaluer les expressions suivantes en respectant l'ordre de priorité des opérateurs:

- ***Expression 1 :*** E1=a+b*c-d/e ; ***avec***: a=10, b=4, c=3, d=20, e=5;

E1=a + b*c - d/e

= 10 + 4*3 - 20/5

=10 + 12 – 20 / 5

= 10 + 12 – 4

= 22 – 4 = 18

- ***Expression 2 :*** E2= ((a + b) – c) % 3 == 0 && c* a – b < c || ! (a >= c) ; ***avec:*** a =6;b= 10;c= 4.

E2 = ((a + b) – c) % 3 == 0 && c*a – b < c || ! (a >= c)

E2 = ((6 + 10) – 4) % 3 == 0 && 4*6 – 10 < 4 || ! (6>= 4)

E2 = (16 – 4) % 3 == 0 && 4*6 – 10 < 4 || ! (6>= 4)

E2 = 12 % 3 == 0 && 4*6 – 10 < 4 || ! (6>= 4)

E2 = 12 % 3 == 0 && 4*6 – 10 < 4 || !True

E2 = 12 % 3 == 0 && 4*6 – 10 < 4 || False

E2 = 0 == 0 && 4*6 – 10 < 4 || False

E2 = 0 == 0 && 24 – 10 < 4 || False

E2 = 0 == 0 && 14 < 4 || False

E2 = 0 == 0 && False || False

E2 = True && False || False

E2 = False || False

E2 = False

Série de TP N°01: Conversions et évaluation d'expressions arithmétiques

Objectif: L'objectif de ce TP est de familiariser les étudiants avec les conversions entre différents systèmes de numération (binaire, octal, décimal et hexadécimal) ainsi qu'avec l'évaluation d'expressions arithmétiques en appliquant les règles de priorité et d'associativité des opérateurs.

Exercice N°01 : *(Systèmes de numérotation)*

Effectuer les conversions suivantes :

$223 = (?)_2$

$(101100101)_2 = (?)_{10}$

$(110011010)_2 = (?)_8 = (?)_{16}$

$(3716)_8 = (?)_2$

$(239)_{10} = (?)_{16}$

$(3DB)_{16} = (?)_{10}$

$(3DA)_{16} = (?)_8$

$(32103)_4 = (?)_2$

Exercice N°02 :(Evaluation des expressions)

Evaluer les expressions suivantes en respectant l'ordre de priorité des opérateurs:

Expression1 : E1=a + b*c - d/e ; ***avec***: a=10, b=4, c=3, d=20, e=5;

Expression2: E2= ((a + b) – c) % 3 == 0 && c* a – b < c || ! (a >= c) ; ***avec:*** a =6;b= 10;c= 4.

Exercice N°03 : *(Expressions arithmétiques en Algorithme/Langage C)*

Réécrire les expressions mathématiques en Algorithme/Langage C

Expression mathématique	**Algorithme / C Language**
$b^2 - 4ac$	
$\frac{-b - \sqrt{d}}{2a}$	
$e^{3a} + \lvert b \rvert$	
$4a < \frac{b}{c}\ AND\ (5c \leq 7)\ OR\ (a \neq b)$	

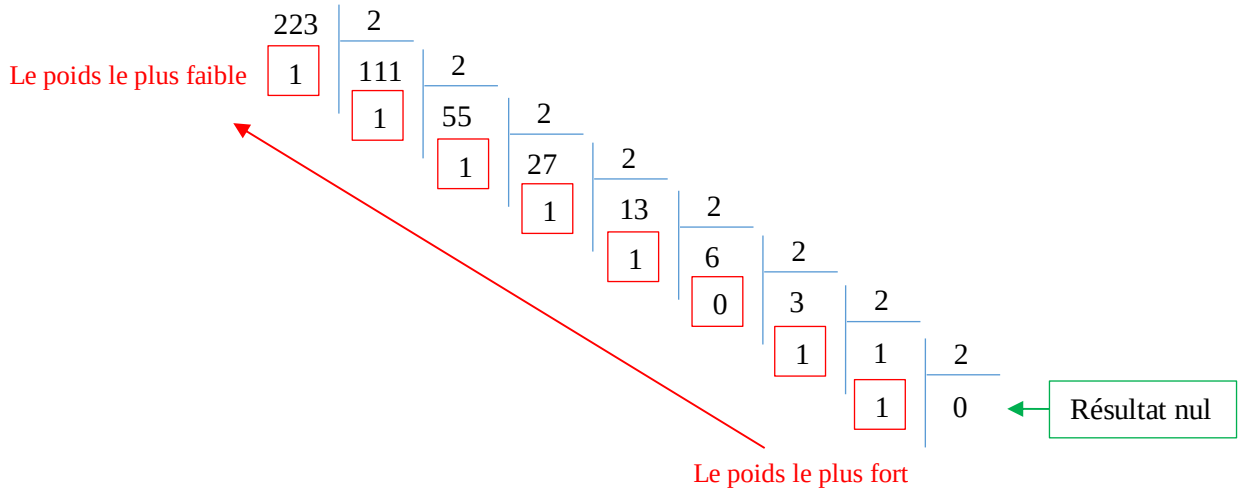
$\frac{a^2 + \sqrt{e^a + c}}{3\sqrt{b}}$	
$\frac{-\sqrt{a} + e^{3b} + \sqrt{a^2 + ab}}{2a + \lvert b \rvert}$	

Corigé de la série de TP N°01 :

Corrigé de l'exercice N°02 : Effectuer les conversions suivantes :

$223 = (?)_2$

Method: Successive divisions by 2



$223 = (11011111)_2$

$(101100101)_2 = (?)_{10}$

Appliquez la formule des poids positionnels :

$(101100101)_2 = (1 * 2^8 + 0 * 2^7 + 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0)_{10}$;

$(101100101)_2 = (357)_{10}$;

$(110011010)_2 = (?)_8 = (?)_{16}$

→ **Binaire vers Octal :** Regrouper les bits par **3** de droite à gauche :

$(110|011|010)_2 = (632)_8$
6 3 2

→ **Binaire vers Hexadécimal :** Regrouper les bits par **4** de droite à gauche (compléter avec des zéros à gauche si nécessaire) :

$(0001|1001|1010)_2 = (19A)_{16}$
1 9 A

Chiffre en base 16	Equivalent en binaire (2^3 2^2 2^1 2^0)
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Chiffre en base 8	Equivalent en binaire (2^2 2^1 2^0)
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

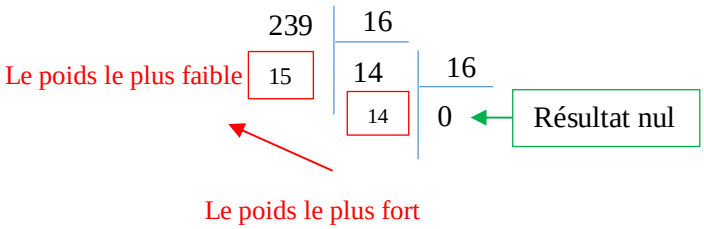
$(3716)_8 = (?)_2$

Méthode : Remplacer chaque chiffre octal par son équivalent binaire sur **3 bits** :

$(3716)_8 = (011111001110)_2$ *or* $(11111001110)_2$;

$(239)_{10} = (?)_{16}$

Méthode : Divisions successives par 16



$(239)_{10} = (EF)_{16}$

Vérification:

$E\times16^1+F\times16^0=14\times16+15\times1=224+15=239$

$(3DB)_{16} = (?)_{10}$

Méthode : Appliquer la formule des poids positionnels :

$(3DB)_{16} = (3 * 16^2 + 13 * 16^1 + 11 * 16^0)_{10}$

$(3DB)_{16} = (987)_{10}$

$(3DA)_{16} = (?)_8$

Méthode : Convertir d'abord hexadécimal → binaire (4 bits par chiffre), puis binaire → octal (3 bits par groupe) :

$$(3DA)_{16} = (001\,|\,111\,|\,011\,|\,010)_2 = (1732)_8$$
$$\quad\quad\quad

• ***Expression 2 :*** E2= ((a + b) – c) % 3 == 0 && c* a – b < c || ! (a >= c) ; ***avec:*** a =6;b= 10;c= 4.

E2 = ((a + b) – c) % 3 == 0 && c*a – b < c || ! (a >= c)

E2 = ((6 + 10) – 4) % 3 == 0 && 4*6 – 10 < 4 || ! (6>= 4)

E2 = (16 – 4) % 3 == 0 && 4*6 – 10 < 4 || ! (6>= 4)

E2 = 12 % 3 == 0 && 4*6 – 10 < 4 || ! (6>= 4)

E2 = 12 % 3 == 0 && 4*6 – 10 < 4 || !True

E2 = 12 % 3 == 0 && 4*6 – 10 < 4 || False

E2 = 0 == 0 && 4*6 – 10 < 4 || False

E2 = 0 == 0 && 24 – 10 < 4 || False

E2 = 0 == 0 && 14 < 4 || False

E2 = 0 == 0 && False || False

E2 = True && False || False

E2 = False || False

E2 = False

Corrigé de l'exercice N°03 : *(Expressions arithmétiques en Algorithme/Langage C)*

Expression mathématique	**Langage C**
$b^2 - 4ac$	$pow(b,2) - 4 * a * c \quad or \quad b * b - 4 * a * c$
$\frac{-b - \sqrt{d}}{2a}$	$(-b - sqrt(d))/(2 * a)$
$e^{3a} + \lvert b \rvert$	$exp(3 * a) + fabs(b)$
$4a < \frac{b}{c}\ ET\ (5c \leq 7)\ OU\ (a \neq b)$	$4 * a < (b/c)\ \&\ \&\ (5 * c <= 7)\ \|\|\ (a! = b)$
$\frac{a^2 + \sqrt{e^a + c}}{3\sqrt{b}}$	$(pow(a,2) + sqrt(exp(a) + c))/(3 * sqrt(b))$
$\frac{-\sqrt{a} + e^{3b} + \sqrt{a^2 + ab}}{2a + \lvert b \rvert}$	$(-sqrt(a) + exp(3 * b) + sqrt(pow(a,2) + a * b))/(2 * a + fabs(b))$

Exercices supplémentaires sur la série de TP N°01

Exercice 01 :

1. Donner l'ordre de priorité des opérateurs arithmétiques et logiques dans les expressions suivantes :

   $E1 = (2 * x + 3 * y)/x + 6 * y - (5\ \%\ y * 2 * x)$

   $E2 = (3\ \%\ a * 2 <= b)\ ||\ !(b >= 8)\ !(a = b)$

2. Traduire les expressions suivantes en langage C

   $E1 = \dfrac{-\sqrt{a} + e^{3b} + \sqrt{a^2 + ab}}{2a + |b|}$, $E2 = \dfrac{5^2 + a^{3b} + \sqrt{e^a + c}}{3\sqrt{b}}$

3. Effectuer les conversions suivantes (les réponses doivent être justifiées):

   $(127)_{10} = (?)_2 = (?)_8$, $(\mathrm{B7C})_{16} = (?)_{10}$

4. **a)** Classer ce qui suit dans les trois catégories suivantes : 1. Systèmes d'exploitation 2. Langages de programmation 3. Logiciels spécialisés.

   Pascal, Microsoft office, Lecteur Media, Windows vista, Google chrome, Mac OS, Adobe, C++, Matlab, Unix, WinRAR.

   b) Citer 3 noms de systèmes d'exploitation pour PC

Exercice 02 :

1. **a)** Classer ce qui suit dans les trois catégories suivantes : 1. Systèmes d'exploitation 2. Langages de programmation 3. Logiciels spécialisés.

   Pascal, Microsoft word, Lecteur Media, Windows vista, Google chrome, Mac OS, C++, Matlab, Linux.

   b) Quelle est la signification des acronymes suivants : CPU, UAL, RAM, ROM

2. Evaluer les expressions suivantes en respectant l'ordre de priorité des opérateurs :

   $E1 = (a/c) - ((d/2 * a + 4)/4 - c)/2b$; avec $a = 6, b = 1, c = 1, d = 4$

   $E2 = (1 > c)\ AND\ NOT(7\ MOD\ a * 2 <= b) OR\ (b >= 8) AND (a = b)$; avec $a = 6, b = 3, c = 1$

3. Traduire les expressions suivantes en langage Pascal

   $E1 = \dfrac{-\sqrt{a} + e^{3b} + \sqrt{a^2 + ab}}{2a + |b|}$, $E2 = \dfrac{a^2 + \sqrt{e^a + c}}{3\sqrt{b}}$

4. Effectuer les conversions suivantes :

   $(120)_{10} = (?)_2 = (?)_8$, $(10110101111)_2 = (?)_{16}$

Exercice 03 :

1. Citer deux périphériques de chacune des trois catégories suivantes :

Entrée	Sortie	Entrée/Sortie

2. Effectuer les conversions suivantes :
   $(90)_{10} = (?)_2 = (?)_{16}$, $(10110101011)_2 = (?)_8$
3. Traduire les expressions suivantes en langage C
   $E1 = \frac{-\sqrt{2a} + e^{3b} + \sqrt{|2a| + b}}{8 + |b|}$, $E2 = \frac{5^2 + a^{3b} + \sqrt{e^a + c}}{\sqrt{b}}$
4. Evaluer l'expression suivante en respectant l'ordre de priorité des opérateurs :
   $E1 = (a/c) - ((d/2 * a + 4)/4 - c)/2 * b$; avec $a = 6, b = 1, c = 1, d = 4$
5. Donner l'expression arithmétique correspondante à l'expression suivante écrite en C :
   $exp(sqrt(x))/(2 * y - 1) + fabs(x) - 1/(sqr(x) + 3)$

Exercice 04 :

1. Quelles sont les trois structures itératives de base ?
2. Citer deux périphériques d'entrée et deux périphériques de sortie.
3. Quels sont les identificateurs valides et invalides parmi les exemples suivants (les identificateurs non valides doivent être justifiés) : _EX01 ; EX_01 ; EX-01 ; EX01_ .
4. Evaluer l'expression suivante tout en montrant l'ordre des opérations.
   (a > 9 DIV 4) AND (a < > b) OR NOT (c = b). avec a = 4, b = 8, c = 8.
5. Donner l'expression arithmétique correspondante à l'expression suivante écrite en Pascal :
   sqrt(abs(2*x + 1 +x*y)) / (sqr(x) – 2*x*y) + sqrt(4*x + 5*x).
6. Effectuer les conversions suivantes : $(1EA)_{16} = (?)_8$, $(240)_{10} = (?)_2$.

CHAPITRE II

Notion d’algorithme et de programme

Notions d'algorithme et de programme

II.1. Objectif du 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.

II.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.

En résumé : un algorithme représente une solution à un problème, exprimée à travers une série d'instructions qui manipulent des données dans un ordre bien défini.

Une fois qu'un algorithme a été conçu, quelle que soit la langue naturelle utilisée (français, anglais, arabe, etc.), il doit être traduit dans un langage de programmation pour devenir un programme exécutable. Ce processus de traduction comprend :

1. La rédaction du code source dans le langage de programmation choisi,
2. La compilation (traduction) de ce code source en instructions compréhensibles par la machine,
3. L'exécution du programme obtenu sur un ordinateur.

Tout au long de ce cours, le langage de programmation utilisé sera le **langage C**.

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

Comme mentionné précédemment, un algorithme représente une solution à un problème donné. Pour parvenir à cette solution algorithmique, un processus structuré d'analyse et de résolution doit être appliqué. Ce processus se déroule en plusieurs étapes, illustrées dans la figure 1.

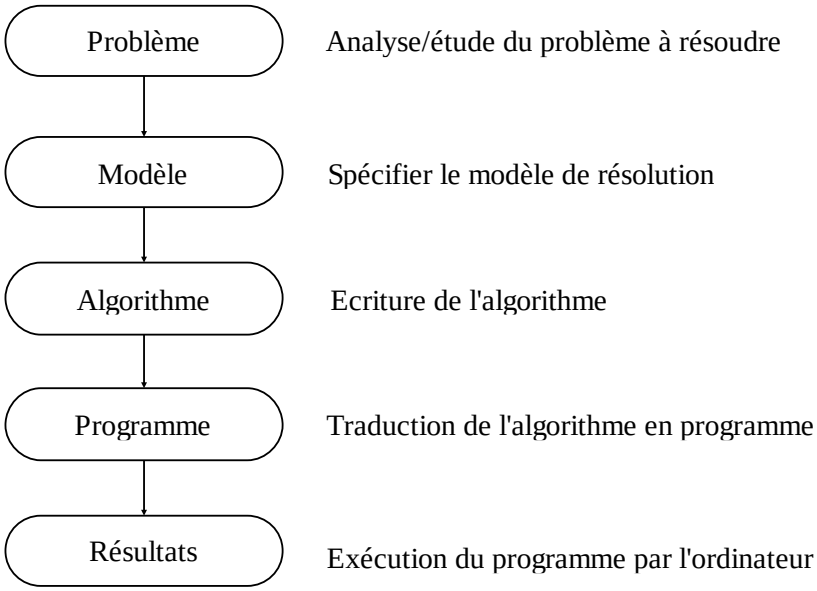


Figure 1 : Etapes d'analyse d'un problème

II.4. Représentation des Algorithmes

Un algorithme peut être représenté selon plusieurs notations. Les deux plus couramment utilisées dans l'enseignement de l'informatique sont :

Notation	**Description**
Organigramme	Une représentation graphique utilisant des formes standardisées : les ovales (début/fin), les rectangles (traitement), les losanges (décision) et les parallélogrammes (entrée/sortie).
Pseudocode	Une description structurée en français (ou en anglais) utilisant des mots-clés tels que SI, TANTQUE, POUR, DÉBUT, FIN.

II.4.1. Représentation par Organigramme

Un organigramme est une **représentation graphique** de la solution à un problème informatique. Il est conceptuellement équivalent à un algorithme : chaque type d'action défini

dans un algorithme possède un symbole standardisé correspondant dans un organigramme, rendant ainsi le déroulement logique de la solution visuellement explicite.

Cependant, la **représentation algorithmique (pseudocode) est généralement préférée** à la représentation par organigramme, notamment lorsque le problème est complexe ou comporte de nombreuses structures imbriquées.

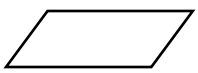
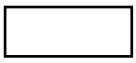
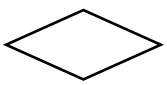
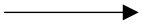
Les principaux **inconvénients** des organigrammes sont :

- Lorsque l'organigramme est long et s'étend sur plusieurs pages, la lisibilité diminue considérablement.
- Les flèches représentant le flux d'exécution peuvent se croiser et se chevaucher, rendant le diagramme confus et difficile à suivre.
- Les organigrammes sont généralement plus difficiles à lire, à rédiger et à maintenir qu'un algorithme bien structuré.

II.4.1.1. Symboles des Organigrammes

Tout organigramme est construit à partir d'un ensemble de symboles graphiques standardisés (ISO 5807), chacun représentant un type spécifique d'opération. Les symboles utilisés dans les organigrammes sont illustrés et décrits dans le Tableau II.1 ci-dessous.

Tableau II.1 : Symboles Standards des Organigrammes

Symbole	**Nom**	**Description**
Ovale / Ellipse 	Terminal	Marque le **Début** et la **Fin** de l'organigramme. Tout organigramme doit avoir exactement un Début et une Fin.
Parallélogramme 	Entrée / Sortie	Représente les opérations d'**entrée** (Lire — recevoir des données de l'utilisateur) et les opérations de **sortie** (Écrire — afficher les résultats).
Rectangle 	Traitement	Représente les **calculs**, les **affectations** et toutes les étapes de traitement des données (ex. x ← a + b).
Losange 	Décision	Représente les **tests et les décisions**. La condition est écrite à l'intérieur du losange. Il possède exactement **deux sorties** : une pour VRAI et une pour FAUX.
Flèche 	Ligne de flux	Indique l'**ordre d'exécution** et relie les symboles pour montrer la direction du flux logique.

Cercle 	Connecteur	Relie deux parties distantes ou séparées de l'organigramme, notamment lorsque le diagramme s'étend sur **plusieurs pages**. Il est identifié par une lettre ou un numéro commun aux deux points reliés.

❖ **Remarque 1 :** Ces symboles sont définis par la norme internationale **ISO 5807**. Leur utilisation cohérente garantit que tout lecteur, quel que soit son parcours ou son langage de programmation, peut interpréter correctement l'organigramme.

❖ **Remarque 2 :** Le symbole **Connecteur** (cercle) est particulièrement important dans les grands organigrammes : il évite les croisements de flèches en remplaçant une longue ligne par une paire de cercles étiquetés correspondants, l'un au point de sortie et l'autre au point d'entrée de la section reliée.

❖ **Remarque 3 :** En pratique, pour les problèmes de complexité moyenne à élevée, **l'algorithme en pseudocode est préféré** à l'organigramme, car il est plus compact, plus facile à modifier et se rapproche davantage du code de programme réel. Les organigrammes restent utiles pour **l'enseignement introductif** et pour illustrer des processus simples et linéaires.

II.4.2. Représentation par pseudocode

Le pseudocode est une notation intermédiaire entre le langage naturel et le langage de programmation. Il permet de décrire un algorithme de manière structurée et lisible, à l'aide de mots-clés simples et conventionnels tels que : **DÉBUT**, **FIN**, **SI**, **SINON**, **TANT QUE**, **POUR**, **LIRE**, **ÉCRIRE**, sans se soucier de la syntaxe stricte d'un langage de programmation particulier.

❖ **Remarque :** Le pseudocode est indépendant de tout langage de programmation et peut être facilement traduit en C, Python, Java, etc. Il est généralement préféré à l'organigramme pour sa concision et sa lisibilité, en particulier pour les algorithmes complexes.

II.5. 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, la structure suivante doit être respectée :

Tableau II.2 : Structure d'un algorithme

Algorithme	Langage C
Algorithme <nomAlgorithme> ; *// Section de déclaration* **Variable** <identifiant> : <TypeDonnée> **Constante** <identifiant> = <Valeur> **Début** <Instruction 1> ; <Instruction 2> ; …… **Fin**.	**#include <stdio.h>** **int main ()** **{** *// Section de déclaration* <TypeDonnée> <identifiant> ; **Const** <identifiant> = <Valeur> ; <Instruction 1> ; <Instruction 2> ; …… **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.

II.5.1. L'entête

L'entête sert à donner un nom à l'algorithme en utilisant un identificateur. Ce dernier est précédé par le mot clé "**Algorithme**". Alors qu'est-ce qu'un identificateur ?

- **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 quidoit 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 II.3 : 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)

❖ **Remarque :** Pour résumer, un identifiant valide doit :

- Commencer par une lettre ou tiré 8 (_)
- Contenir uniquement des lettres, des chiffres ou des tirés 8
- Ne pas être un mot-clé réservé du langage
- Ne contenir ni espaces ni caractères spéciaux

II.5.2. La partie déclarative

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

II.5.2.1. Constantes

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

En algorithme:

Const <identifiant> = <valeur> ;

En langage C:

const <type> <identifiant> = <valeur> ;

Exemple :

Tableau II.4 : Exemples de constantes

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

II.5.2.2. Variables

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

En algorithme:

<identifiant> : <TypeDonnée>;

En langage C:

<TypeDonnée> <identifiant>;

Une variable appartient à exactement un type de données. Il existe cinq types de données fondamentaux :

Tableau II.5 : Types de Données Fondamentaux et leur Déclaration en Algorithme et en C

Type de Données	Algorithme	Langage C	Exemples
Entier	x : Entier;	int x;	…, −2, −1, 0, 1, 2, …
Réel	x : Réel;	float x; **ou** double x;	3.14, −0.5, 2.718

Caractère	c : Caractère;	char c;	'A', '7', '@'
Chaîne de caractères	s : Chaîne;	char s[50];	"Bonjour", "CS101"
Booléen	b : Booléen;	int b; **ou** bool b;	VRAI / FAUX → 1 / 0

Exemple :

Tableau II.6 : 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

❖ **Remarque :** En plus des constantes et des variables, il est également possible de déclarer de nouveaux types, fonctions et procédures.

Comme vu précédemment, chaque type de données algorithmique possède un équivalent direct en C. Lors de l'implémentation d'un algorithme en C, il est également essentiel de connaître le spécificateur de format associé à chaque type, car ceux-ci sont requis par les fonctions d'entrée/sortie standard printf() et scanf(). Le tableau II.7 résume cette correspondance.

Tableau II.7: Correspondance Algorithme/Langage C

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

❖ **Remarque :** Le type bool nécessite **#include <stdbool.h>**. 1 représente VRAI (TRUE) et 0 représente FAUX (FALSE).

II.5.3. Le corps de programme

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**.

II.6. 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.

II.6.1. Instructions d'Entrées/Sorties (Lecture / Écriture)

II.6.1.1. 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 II.8 : 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 lire des 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 II.9 : Exemples d'entrées

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

II.6.1.2. 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 II.10 : 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 II.11 : 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

II.6.2. 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 II.12 : 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 II.13 : 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

Exercice : *(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 ?

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

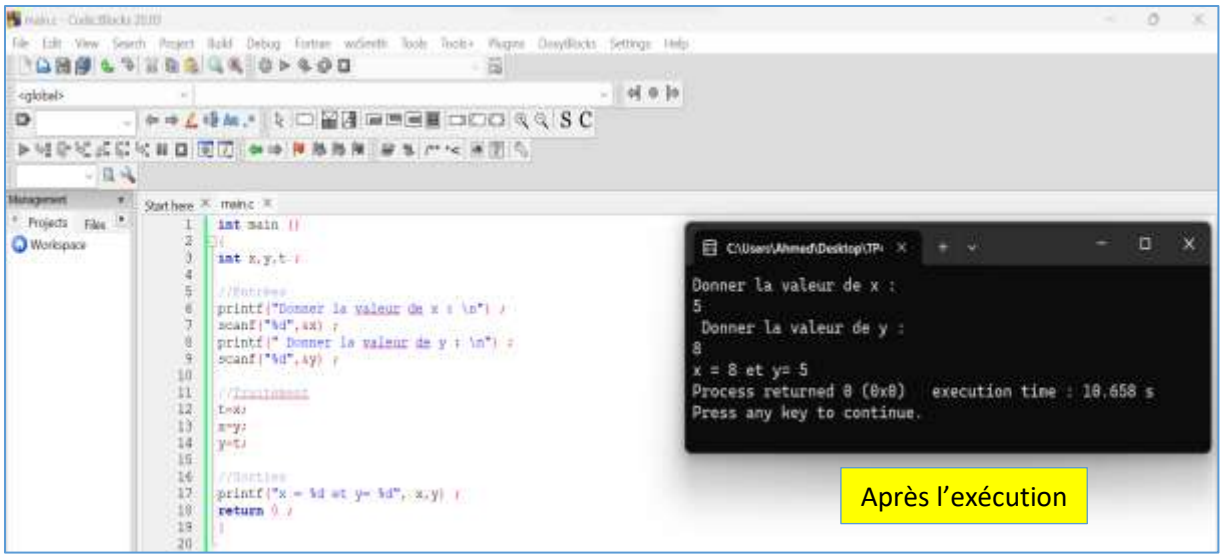
Modifier l'algorithme dans le cas où des coefficients (C1, C2 et C3) sont attribués aux trois matières

Corrigé de l'exercice : *(Enoncé du problème ➔ Algorithme ➔ Programme C)*

1) Permuter entre les deux variables X et Y ?

Algorithme	**Programme C**

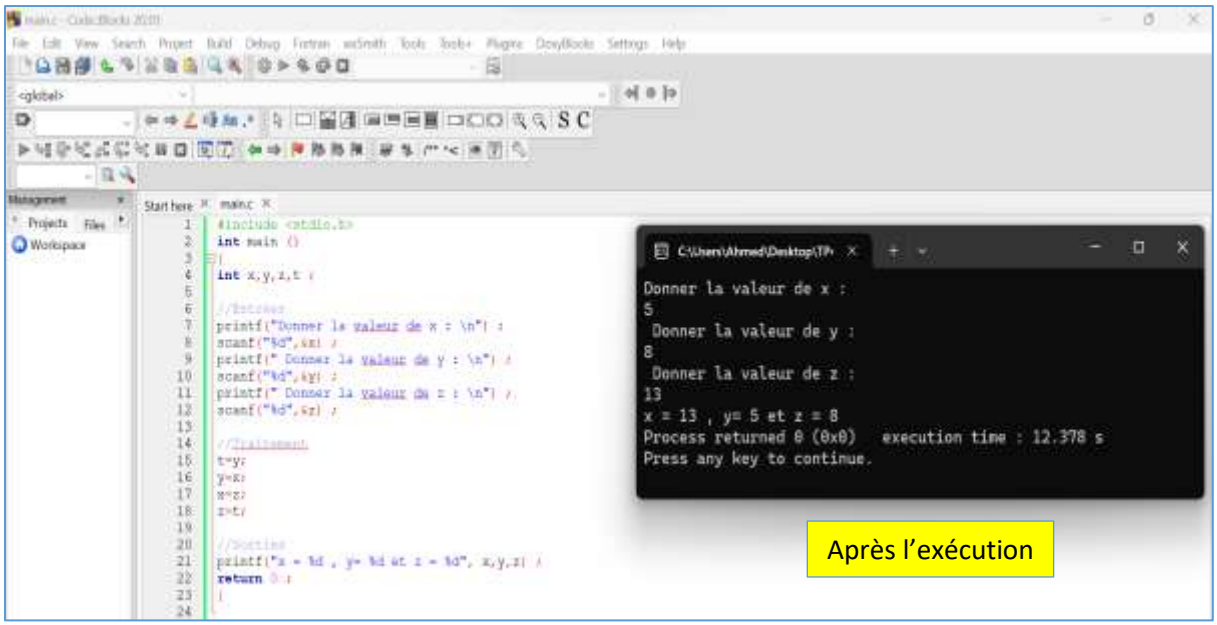
Algorithme Exo2_1;
Variables
x, y, t : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de x : ") ;
Lire(x) ;
Ecrire("Donner la valeur de y : ") ;
Lire(y) ;
 	
//Traitement
t ← x;
x ← y;
y ← t ;
 	
//Sorties
Écrire("x=", x, "y=", y) ;
Fin.
	#include <stdio.h>
	int main ()
	{
	int x,y,t ;
	
	//Entrées
	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**

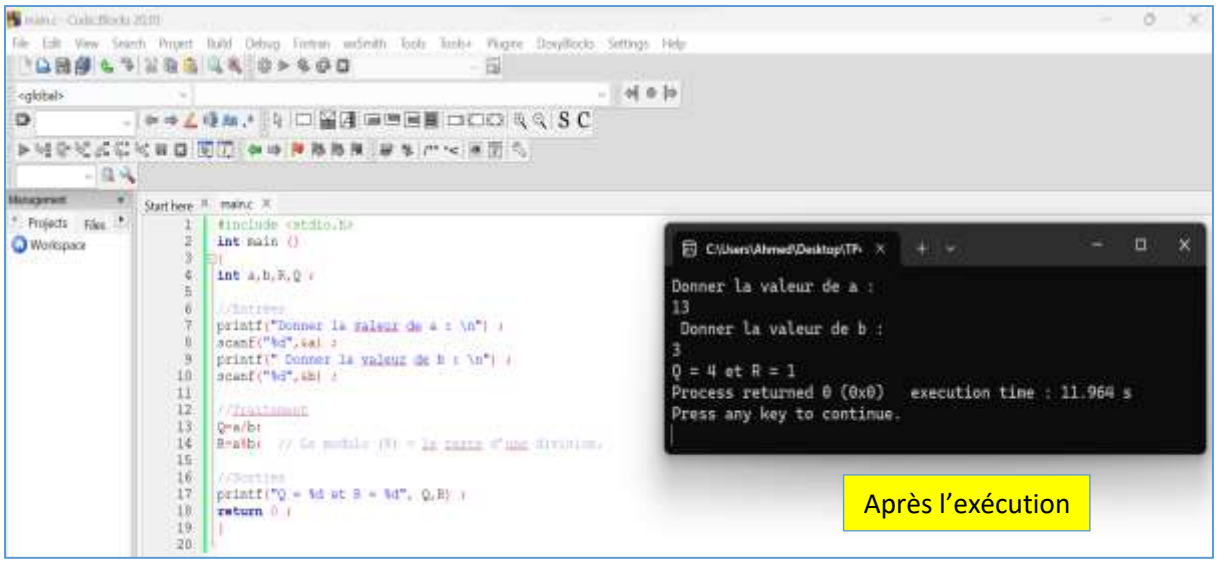
Algorithme Exo2_2;
Variables
x, y, z, t : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de x : ") ;
Lire(x) ;
Ecrire("Donner la valeur de y : ") ;
Lire(y) ;
Ecrire("Donner la valeur de z : ") ;
Lire(z) ;
 	
//Traitement
t ← y;
y ← x;
x ← z ;
z ← t ;
 	
//Sorties
Écrire("x=", x, "y=", y, "z=",z);
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**

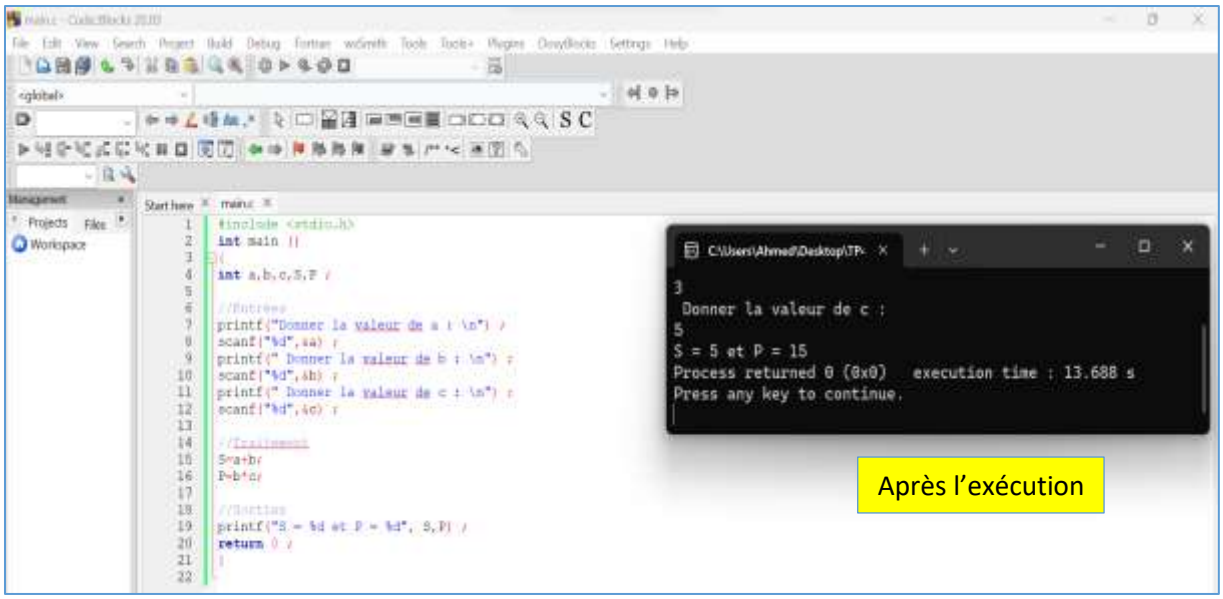
Algorithme Exo2_4;
Variables
a, b, Q, R : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de a : ") ;
Lire(a) ;
Ecrire("Donner la valeur de b : ") ;
Lire(b) ;
 	
//Traitement
Q ← a div b; //div : division entière
R ← a mod b; //mod : reste de division
 	
//Sorties
Écrire("Q=", Q, " et R= ", R) ;
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**

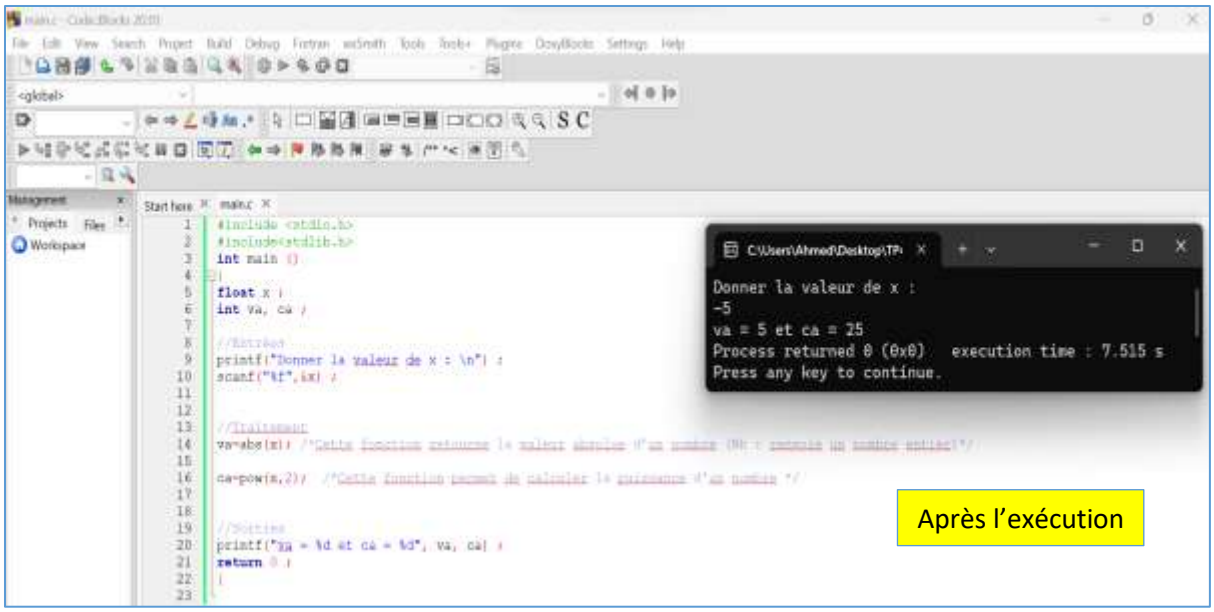
Algorithme Exo2_5;
Variables
a, b, c, S, P : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de a : ") ;
Lire(a) ;
Ecrire("Donner la valeur de b : ") ;
Lire(b) ;
Ecrire("Donner la valeur de c : ") ;
Lire(c) ;
 	
//Traitement
S ← a + b;
P ← b * c;
 	
//Sorties
Écrire("S=", S, "P= ", P) ;
Fin.
	#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**

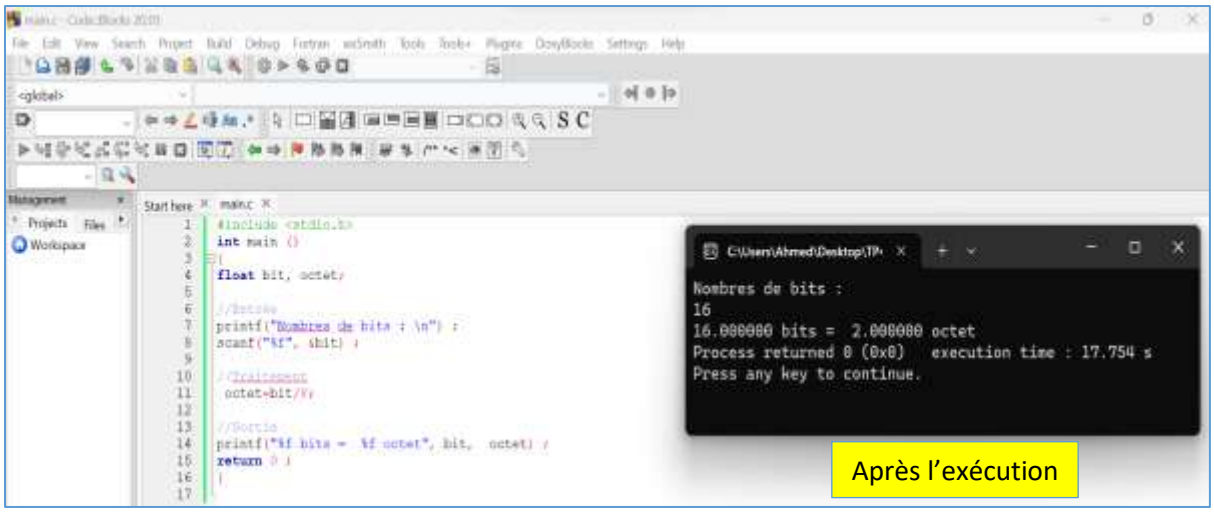
Algorithme Exo2_6;
Variables
x : **réel**;
va, ca : **entier**;
 	
Début
//Entrées
Ecrire("Donner la valeur de x : ") ;
Lire(x) ;
 	
//Traitement
va ← |x|;
ca ← x^2;
 	
//Sorties
Écrire("va=", va, "ca=", ca) ;
Fin.
	#include <stdio.h>
	#include<stdlib.h>
	int main ()
	{
	float x ;
	int va, ca ;
	
	//Entrées
	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**

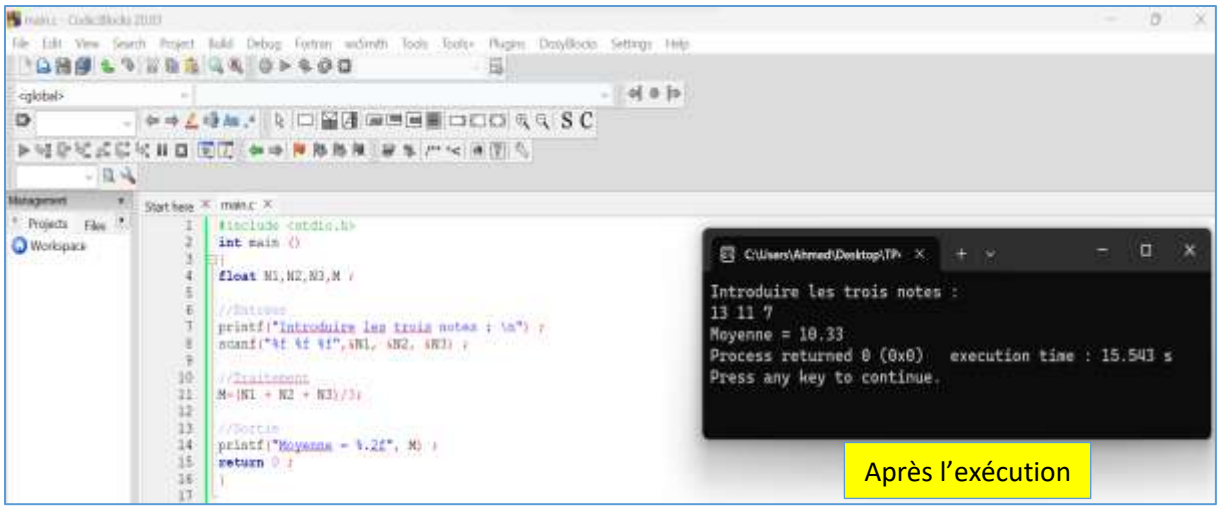
Algorithme Exo2_7;
Variables
bit, octet : **réel**;
 	
Début
//Entrée
Écrire("Nombres de bits =") ;
Lire(bit) ;
 	
//Traitement
octet ← bit/8;
 	
//Sortie
Écrire(bit, " bits =", octet, " octet");
Fin.
	#include <stdio.h>
	int main ()
	{
	float bit, octet;
	
	//Entrée
	printf("Nombres de bits : \n") ;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

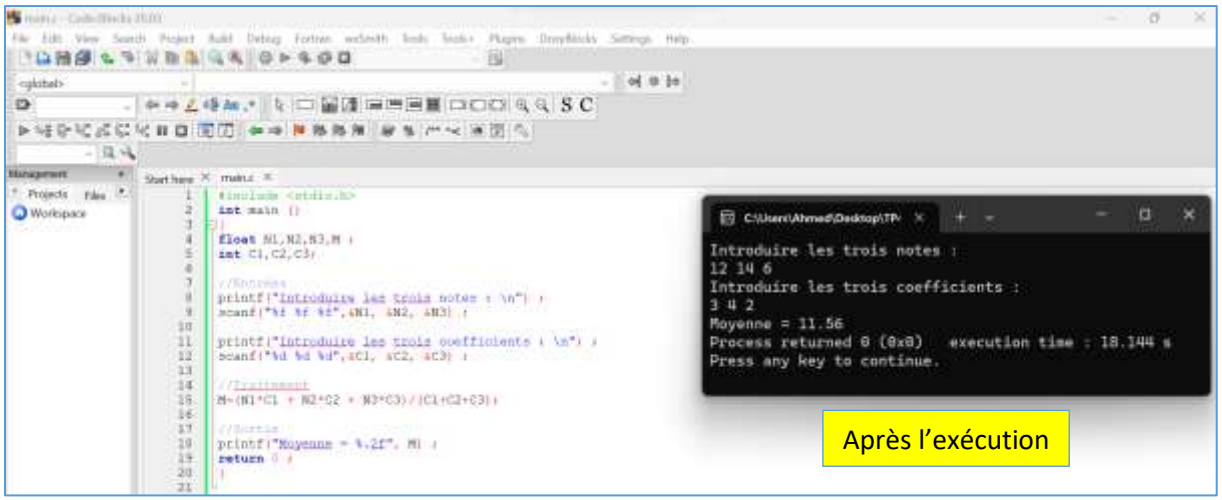
Algorithme Exo2_7_a;
Variables
N1, N2, N3, M : **réel** ;
 	
Début
//Entrées
Écrire ("Introduire les trois notes : ") ;
Lire(N1, N2, N3) ;
 	
//Traitement
M ← (N1 + N2 +
N3)/3;
 	
//Sorties
Écrire("Moyenne =", M:0:2) ;
Fin.
	#include <stdio.h>
	int main ()
	{
	float N1,N2,N3,M ;
	
	//Entrées
	printf("Introduire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



II.6.3. 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 de contrôle conditionnelle (Tests)
- Structures de contrôle répétitives (itératives, boucles).

II.6.3.1. 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

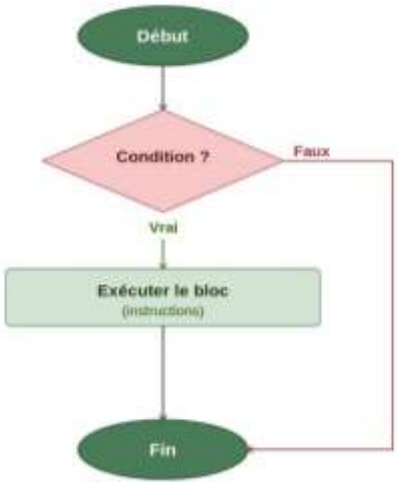
Un test simple contient un seul bloc d'instructions. Selon une condition (expression logique), on décide est ce que le bloc d'instructions est exécuté ou non. Si la condition est vraie, on exécute le bloc, sinon on l'exécute pas.

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

Tableau II.14 : La syntaxe d'un test alternatif simple

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

Organigramme :



Exemple :

Tableau II.15 : 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

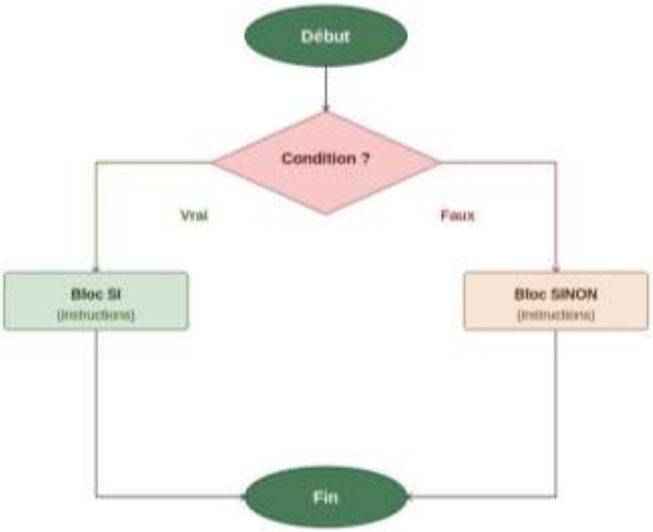
Un test double contient deux blocs d'instructions : on est amené à décider entre le premier bloc ou le seconds. Cette décision est réalisée selon une condition (expression logique ou booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second.

La syntaxe d'un test alternatif double est :

Tableau II.16 : 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>; **}**

Organigramme :



Exemple :

Tableau II.17 : 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.

Exercice N°01:

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

Corrigé de l'exercice N°01 :

Algorithme	**Programme C**

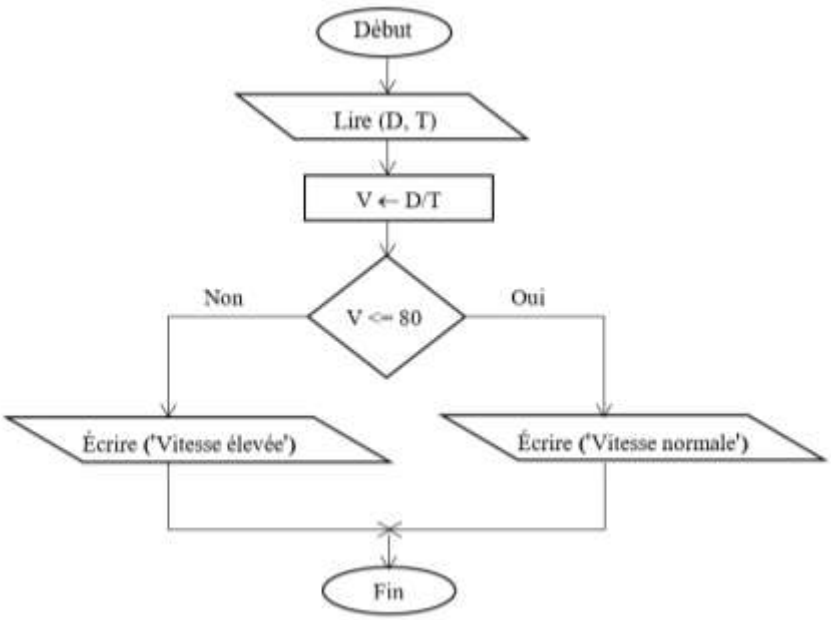
Algorithme Partie ;
Variables
N, R : **entier**;
 	
Début
//Les entrées
Écrire ("Donner un entier : ") ;
Lire (N) ;
 	
//Traitement
R ← N mod 2;
 	
//Les sorties
Si (R = 0) **alors**
Écrire (N, " est pair")
Sinon
Écrire (N, " est impair") ;
Fin-Si ;
Fin.
	#include <stdio.h>
	
	int main()
	{
	int N, R;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

Exercice N°02:

Donner l'organigramme (algorigramme) de l'algorithme ci-dessous :

```
Algorithme Vitesse ;
Variables
  V, D : réel ;
  T : entier ;
Début
    Lire (D, T) ;
    V ← D/T ;
    Si (V <= 80) alors
      Écrire ('Vitesse normale')
    Sinon
      Écrire ('Vitesse élevée') ;
    FinSi
Fin.
```

Corrigé de l'exercice N°02 :



II.6.3.2. 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 II.18 : 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)> ; **}**

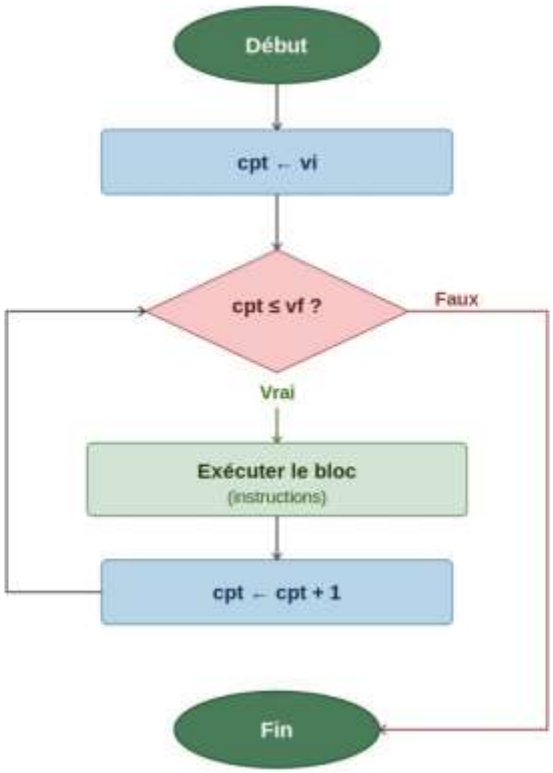
<cpt> : compteur (variable entière)

<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>.

Organigramme :



Example:

Tableau II.19: Exemple de la boucle pour

Algorithme	Equivalent en langage C

// Algorithme — Afficher les entiers de 1 à 5
pour i ← 1 **à** 5 **faire**
Ecrire(i);
FinPour;
	// Langage C
	for (i = 1; i <= 5; i++)
	{
	printf("%d\n", i);
	}
	// Sortie : 1 2 3 4 5

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 :

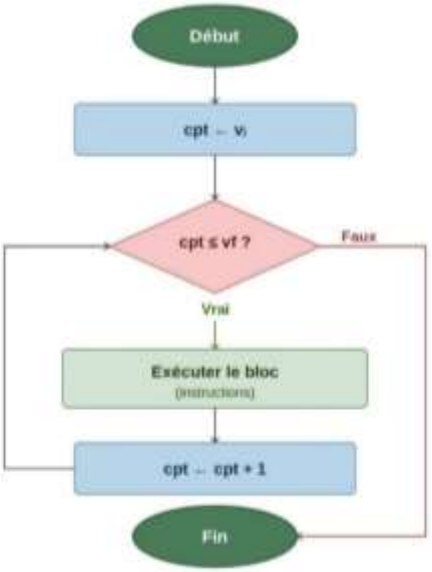
Tableau II.20 : La syntaxe de la boucle **tant-que**

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 **}**).

Organigramme:



Example:

Tableau II.21: Exemple de la boucle Tant-que

Algorithme	Equivalent en langage C

// Algorithme — Afficher les entiers de 1 à 5
i ← 1;
Tant-que i <= 5 **faire**
Ecrire(i);
i ← i + 1;
Fin Tant-que;
	// Languge C
	i = 1;
	while (i <= 5)
	{
	printf("%d\n", i);
	i++;
	}
	// Sortie: 1 2 3 4 5: 1 2 3 4 5

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

Boucle pour	Boucle Tant-que
Pour i ← 1 **à** 5 **faire** Ecrire(i); **FinPour;**	**i ← 1;** **Tant-que** i <= 5 **faire** Ecrire(i); **i ← i + 1;** **Fin Tant-quee;**

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 II.22 : La syntaxe de la boucle **répéter**

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

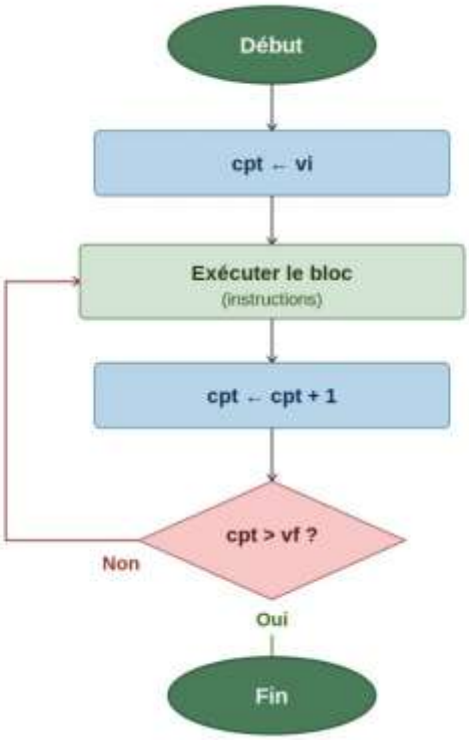
<condition> : expression logique qui peut être vraie ou fausse.
<condition2> : expression logique inverse de <condition>.

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.

Organigramme:



Example:

Tableau II.23: Exemple de la boucle répéter

Algorithme	Equivalent en langage C

// Algorithme — Afficher les entiers de 1 à 5
i ← 1;
Répéter
Write(i);
i ← i + 1;
jusqu'à i > 5;
	// Langage C
	i = 1;
	do
	{
	printf("%d\n", i);
	i++;
	} while (i <= 5); /* condition inverse : Until i>5 → while i<=5 */
	// Sortie: 1 2 3 4 5

Série de TP N°2 : 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

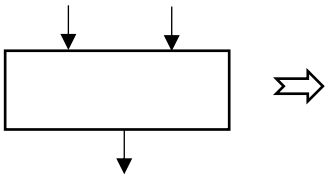
3- Remplacer la dernière instruction :
Écrire("Volume = ", V), par une autre instruction qui permet d'afficher le volume avec seulement deux chiffres après la virgule, puis ré-exécuter le programme.

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 : *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°2 :

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;
}
```

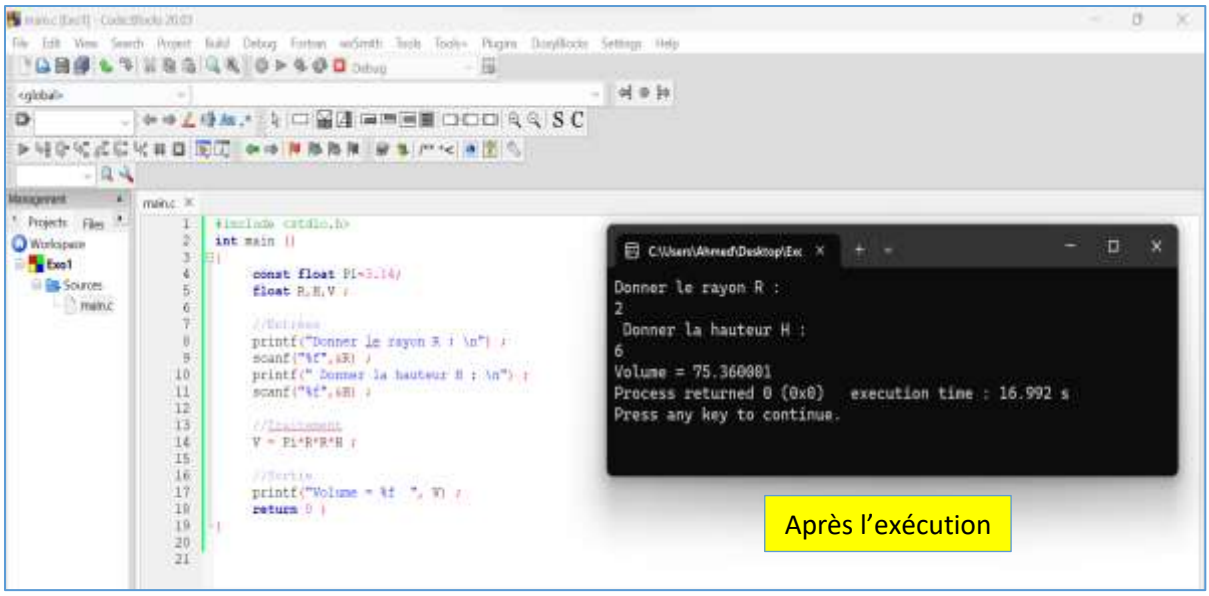
Exercice N°01 : *(Algorithme ➔ Programme C)*

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 :");
Lire(H);
 	
//Traitement
V ← Pi*R*R*H ;
 	
//Sortie
Écrire("Volume = ", V);
Fin.
	#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 = %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 ;

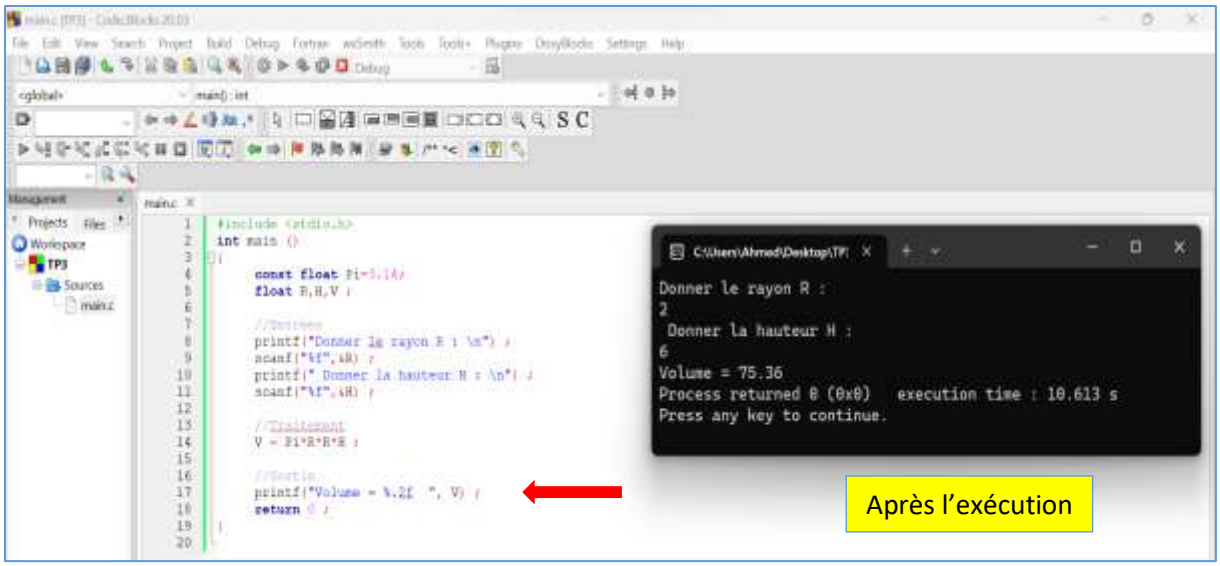
    //Entrées
    printf("Donner le rayon R : \n") ;
    scanf("%f",&R) ;
    printf(" Donner la hauteur H : \n") ;
    scanf("%f",&H) ;

    //Traitement
    V = Pi*R*R*H ;

    //Sortie
    printf("Volume = %.2f  ", V) ;
    return 0 ;
}
```

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

Où 2 représente deux chiffres après la virgule.



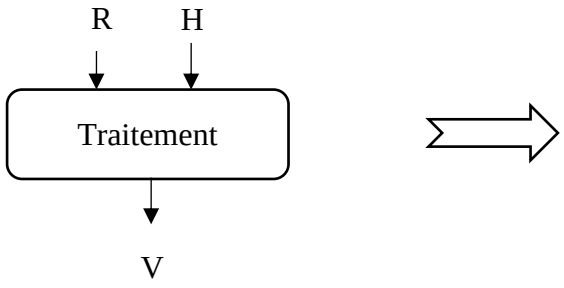
4) Déroulement de l'algorithme pour R=2 et H=6

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 :

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 Exo2;
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;
	}

Après l'exécution

Exercices supplémentaires sur la série de TP N°2

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°3 : Les structures de contrôle conditionnelles

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 ?

Corrigé de la série de TP N°3

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

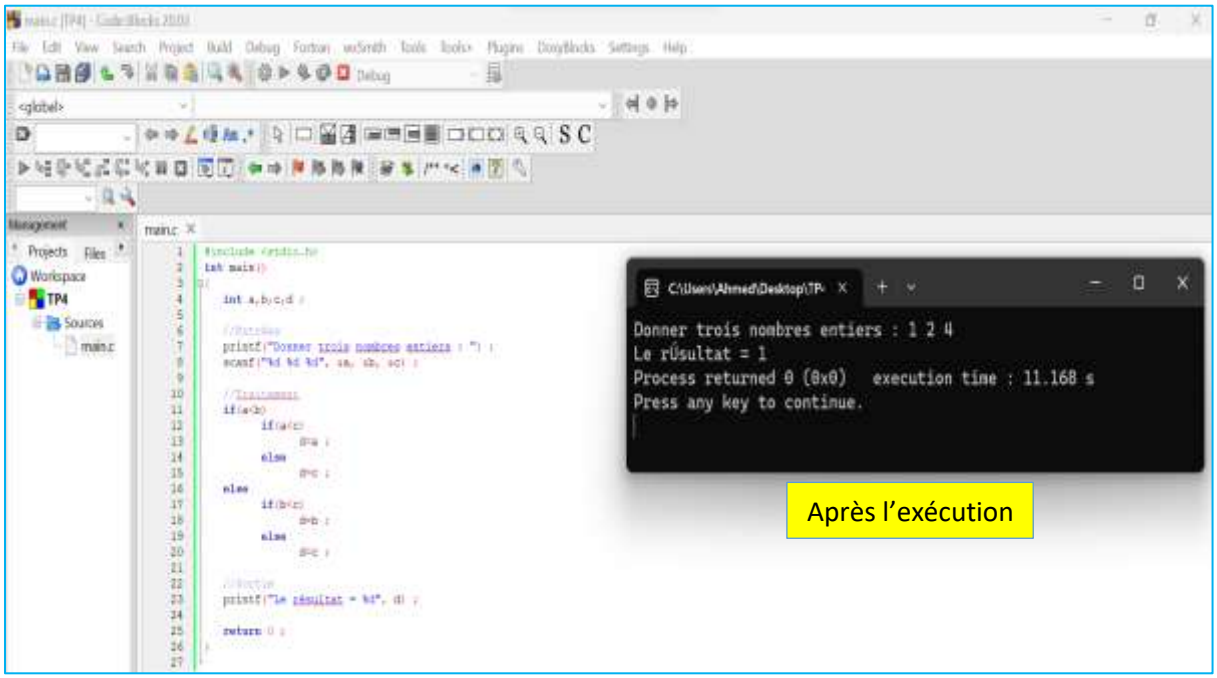
Algorithme	Programme C

Algorithme exo1 ;
Variables
  a, b, c, d : **entier** ;
 	
Début
  //Entrées
  Écrire ("Donner trois nombres entiers : ") ;
  Lire (a,b,c) ;
 	
  //Traitement
  Si (a<b) **alors**
	#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)

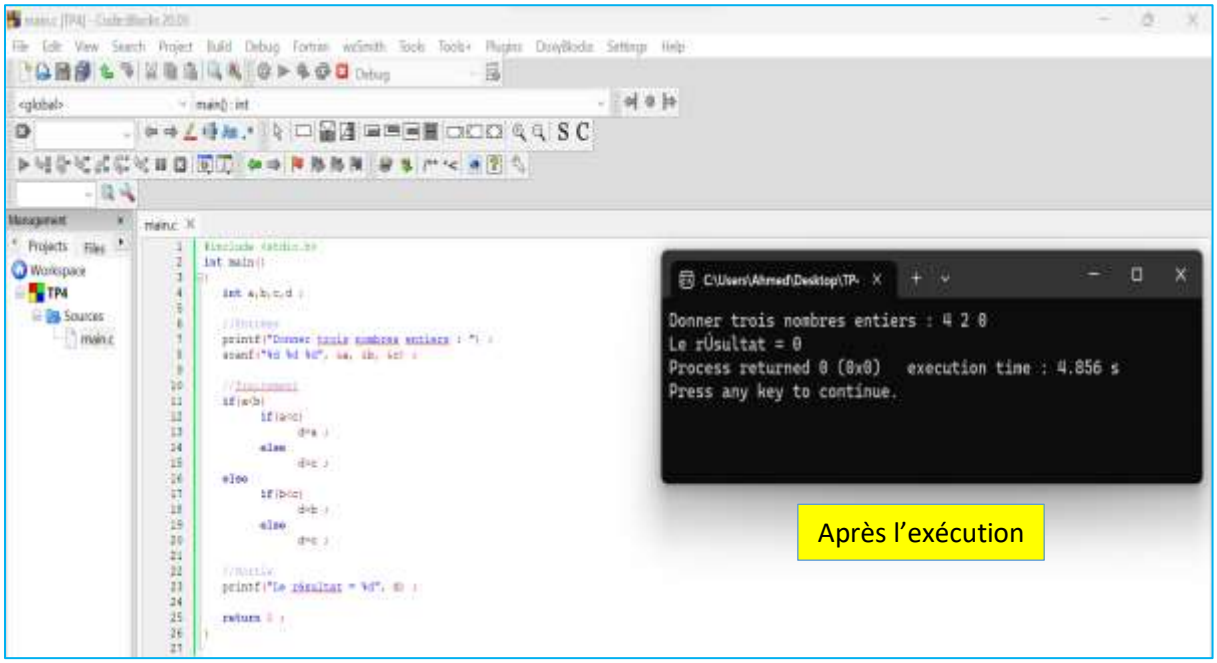
Si (a<c) **alors**
d ← a
Sinon
d ← c ;
Fin-Si
Sinon
Si (b<c) **alors**
d ← b
Sinon
d ← c ;
Fin-Si
Fin-Si
 	
//Sortie
Écrire ("Le résultat = ", d) ;
Fin.
	if(a<c)
	d=a ;
	else
	d=c ;
	else
	if(b<c)
	d=b ;
	else
	d=c ;
	
	//Sortie
	printf("Le résultat = %d", d) ;
	
	return 0 ;
	}

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

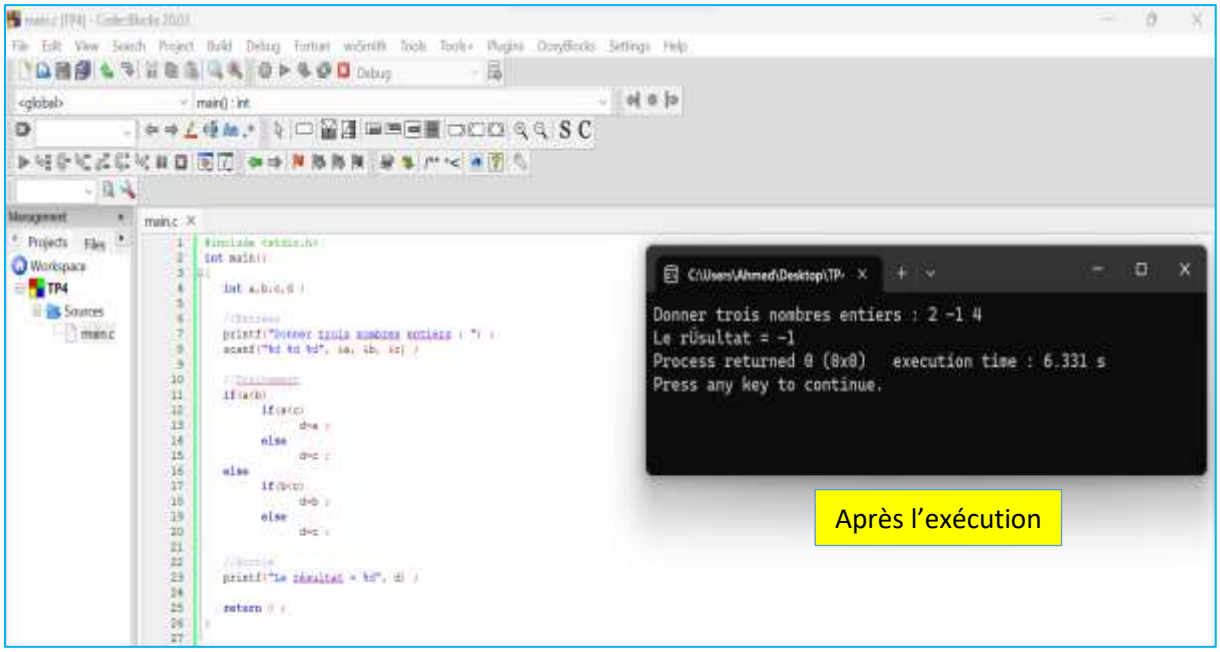
- **a=1, b=2, c=4**



- **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

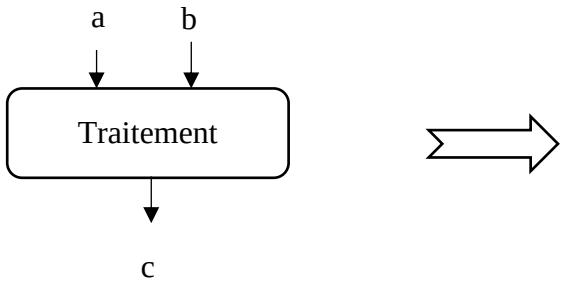
Instructions	Variables				Affichage
	a	b	c	d	
Écrire ("Donner trois nombres entiers : ") ;	/	/	/	/	Donner trois nombres entiers :
Lire (a,b,c) ;	4	2	0	/	
Si (a<b) **alors** **4 < 2** **False** ⇨ On n'exécute pas le bloc **Si** (on passe au bloc **Sinon**) **Si** (b<c) **alors** **4 < 0** **False** ⇨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

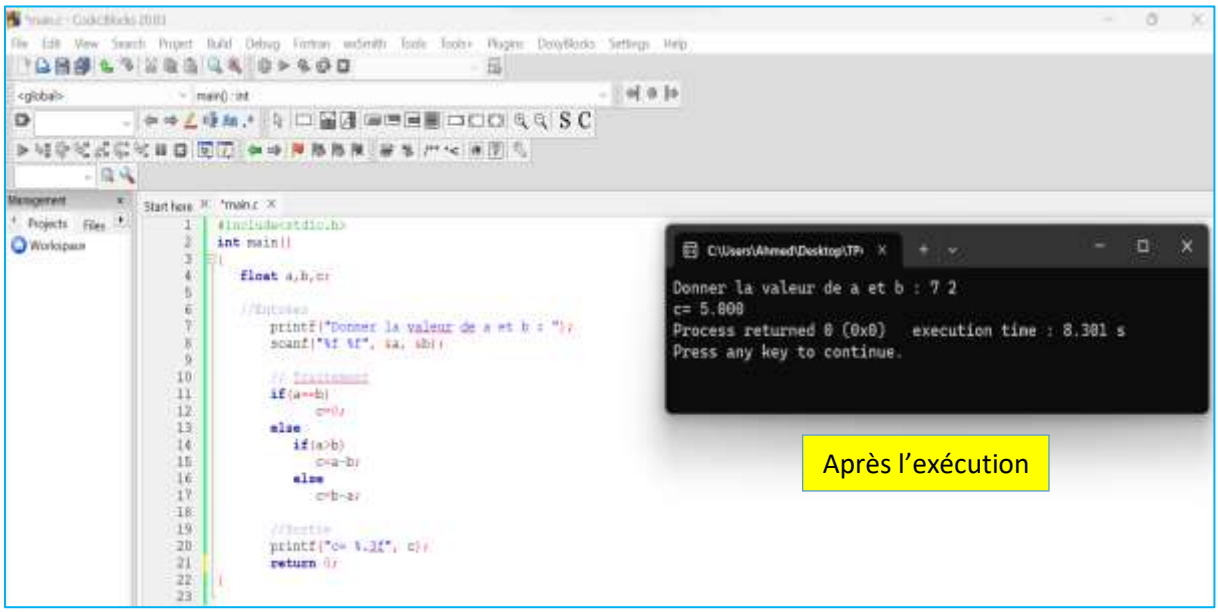
Si a=b **alors**
c ← 0;
Sinon
Si a>b **alors**
c ← a-b;
Sinon
c ← b-a;
Fin-Si
Fin-Si

Sortie :

Écrire (c) ;

Algorithme	**Programme C**

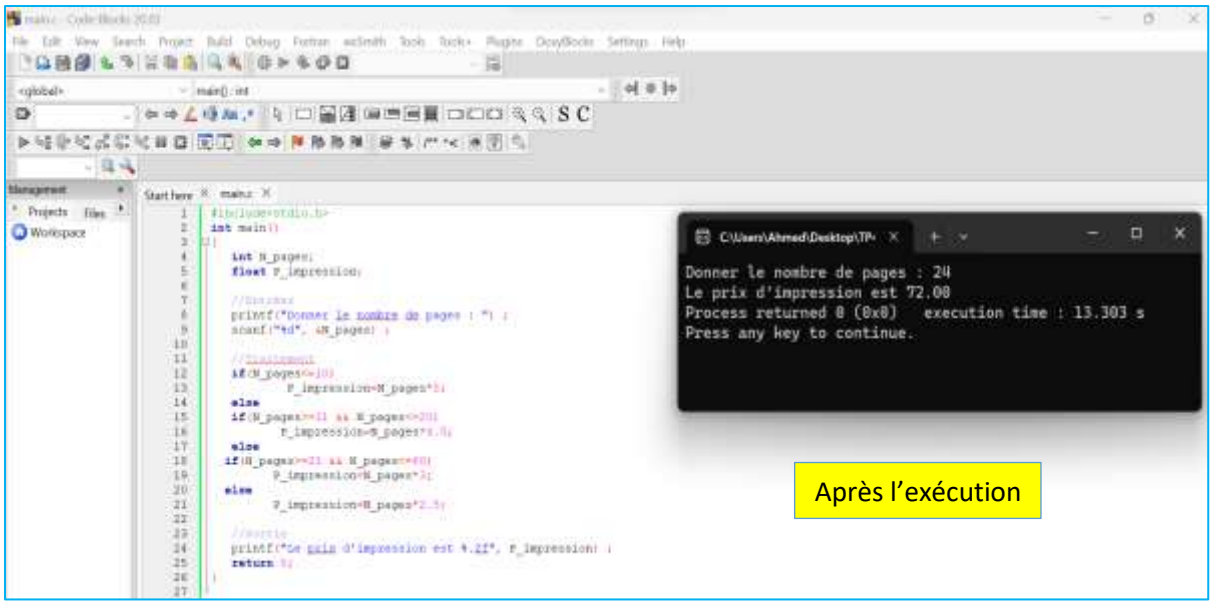
Algorithme exo2;
Variables
a, b, c : **réel**;
Début
//Entrées
Ecrire("Donner la valeur de a et b : ");
Lire (a,b);
 	
//Traitement
Si (a=b) **alors**
c ← 0;
Sinon
Si (a>b) **alors**
c ← a-b;
Sinon
c ← b-a;
Fin-Si
Fin-Si
 	
//Sortie
Écrire ("c = ", c)
Fin.
	#include<stdio.h>
	int main()
	{
	float a,b,c;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

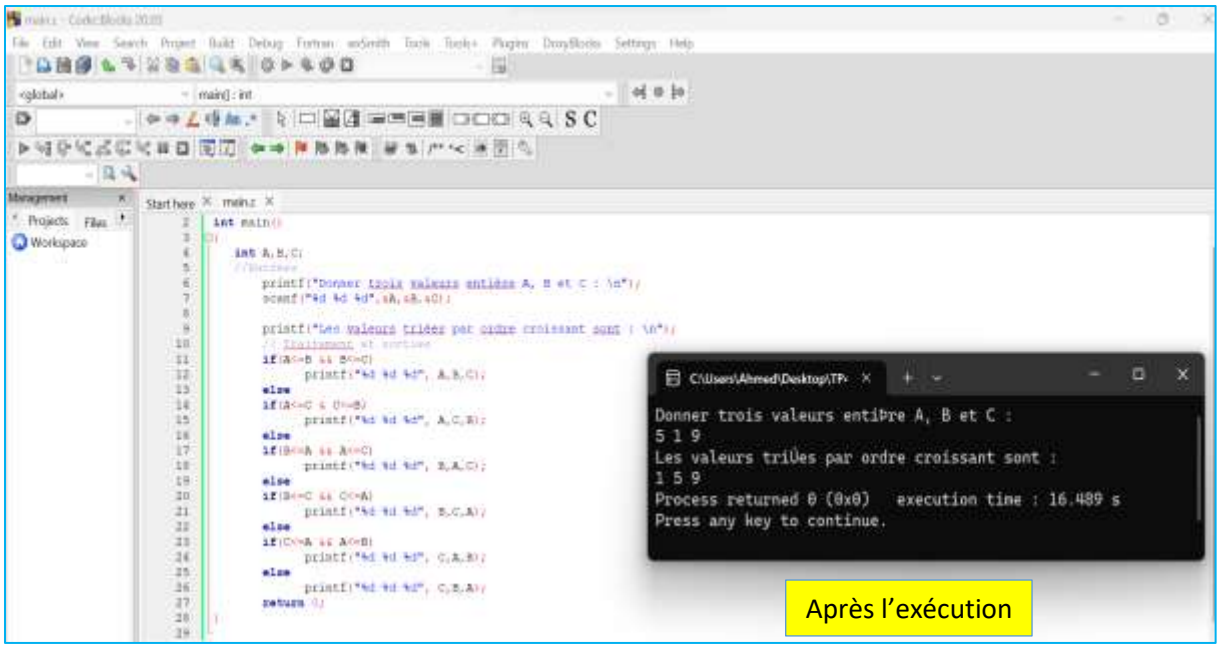
Algorithme Prix_impression;
Variables
N_pages : **entier**;
P_impression : **réel**;
 	
Début
//Entrées
Écrire ("Donner le nombre de pages : ") ;
Lire (N_pages) ;
 	
//Traitement
 	
Si (N_pages <= 10) **alors**
P_impression ← N_pages*5 ;
Sinon
Si (N_pages >= 11 **ET** N_pages<=20)
alors
P_impression ← 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 ;
Fin-Si ;
 	
//Sortie
Écrire("Le prix d'impression est " ,
P_impression);
Fin.
	#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) ;
	
	return 0;
	
	}


Donner le nombre de pages : 24
Le prix d'impression est 72.00
Process returned 0 (0x0) execution time : 13.303 s
Press any key to continue.
Après l'exécution

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.

Exercices supplémentaires sur la série de TP N°3

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°4 : Les structures de contrôle répétitives (boucles)

But du TP :

Le but du TP est de permettre aux étudiants de se familiariser avec les trois structures de contrôle répétitives : POUR, TANT QUE et RÉPÉTER…JUSQU'À, en développant leur capacité à choisir la boucle appropriée selon le problème posé, à maîtriser les conditions d'initialisation, d'arrêt et d'incrément, et à traduire correctement un algorithme itératif en langage C.

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°4

Rappel :

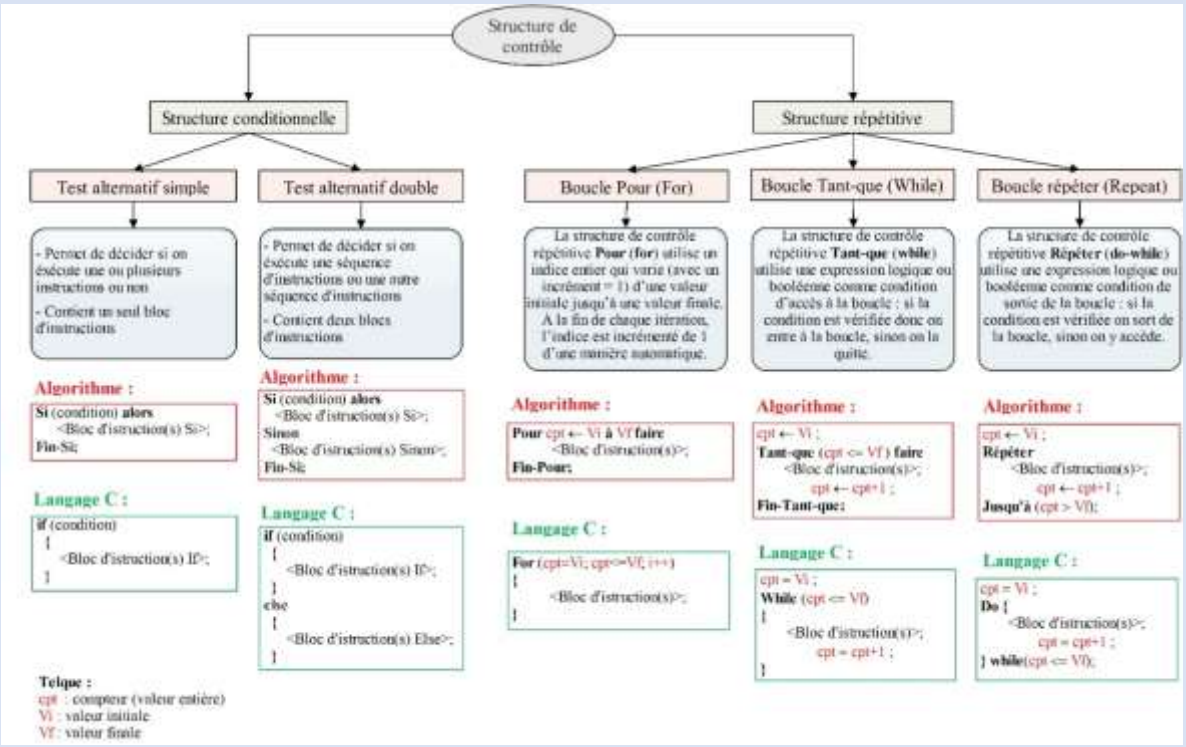
Structures de contrôle répétitives

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.

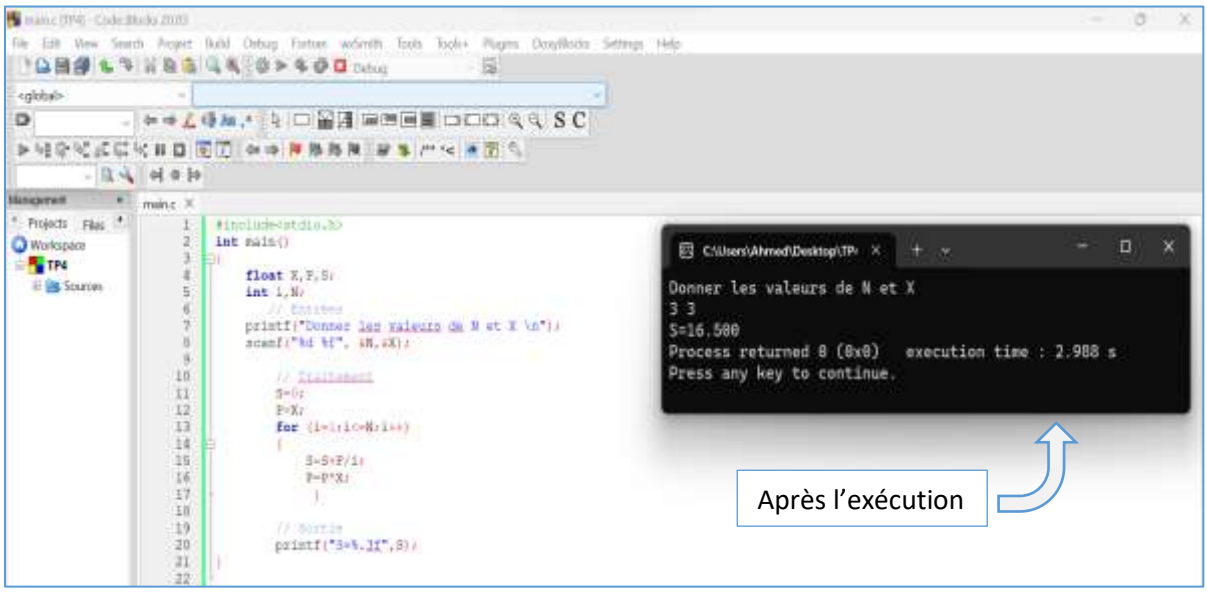
Exercice N°01 : *(Algorithme ➔ Programme en langage C)*

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 : $S = X + \frac{X^2}{2} + \frac{X^3}{3} + \ldots\ldots + \frac{X^N}{N}$

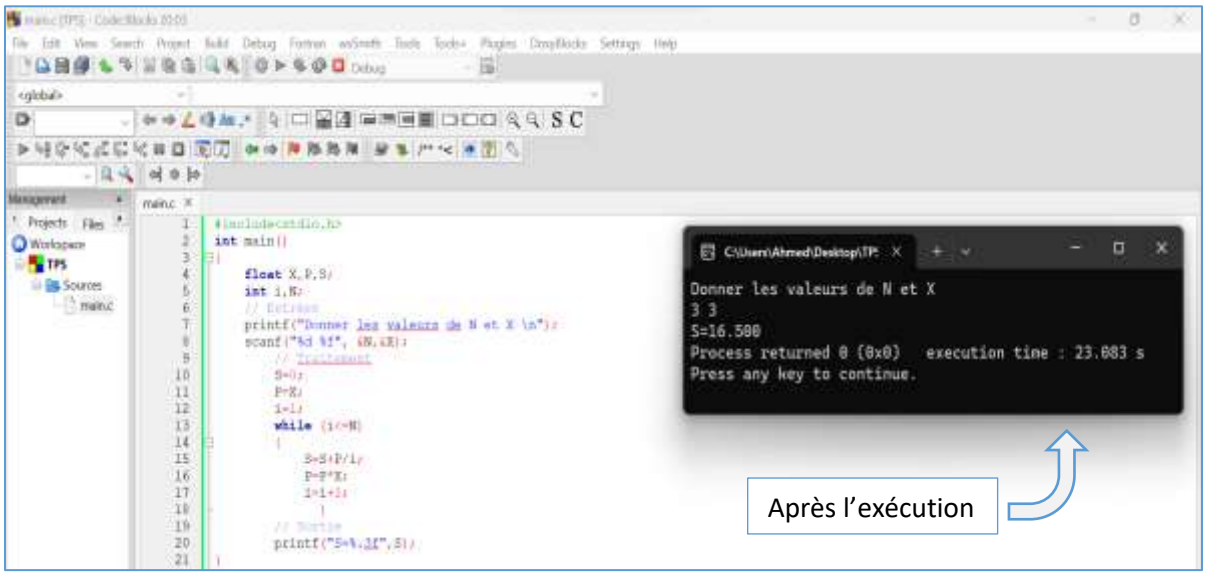
On peut généraliser par la formule suivante :

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

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

Algorithme	**Programme C**

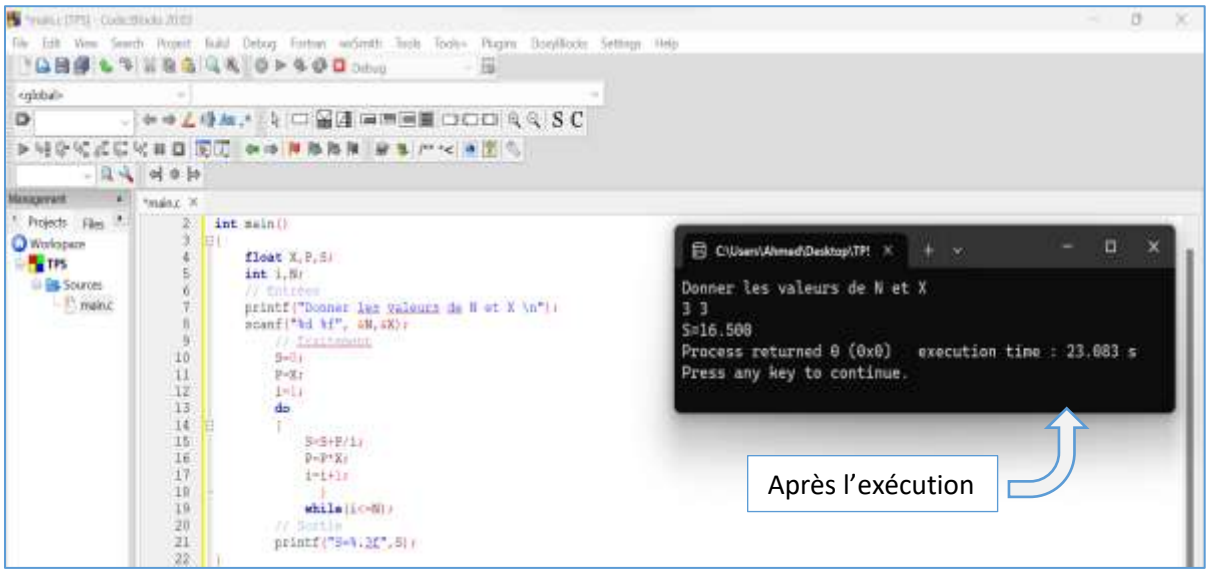
Algorithme Exo1 ;
Variables
X, P, S : **réel** ;
i, N : **entier** ;
Début
 	
// Entrées
Ecrire ("Donner les valeurs de N et X : ") ;
Lire (N, X) ;
 	
// Traitement
S ← 0 ;
P ← X ;
i ← 1 ;
Tant-que i ≤ N **faire**
S ← S+ P/i ;
P ← P*X ;
i ← i+1 ;
Fin Tant-que
 	
// Sortie
Ecrire ("S =", S) ;
Fin.
	#include<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);
	}

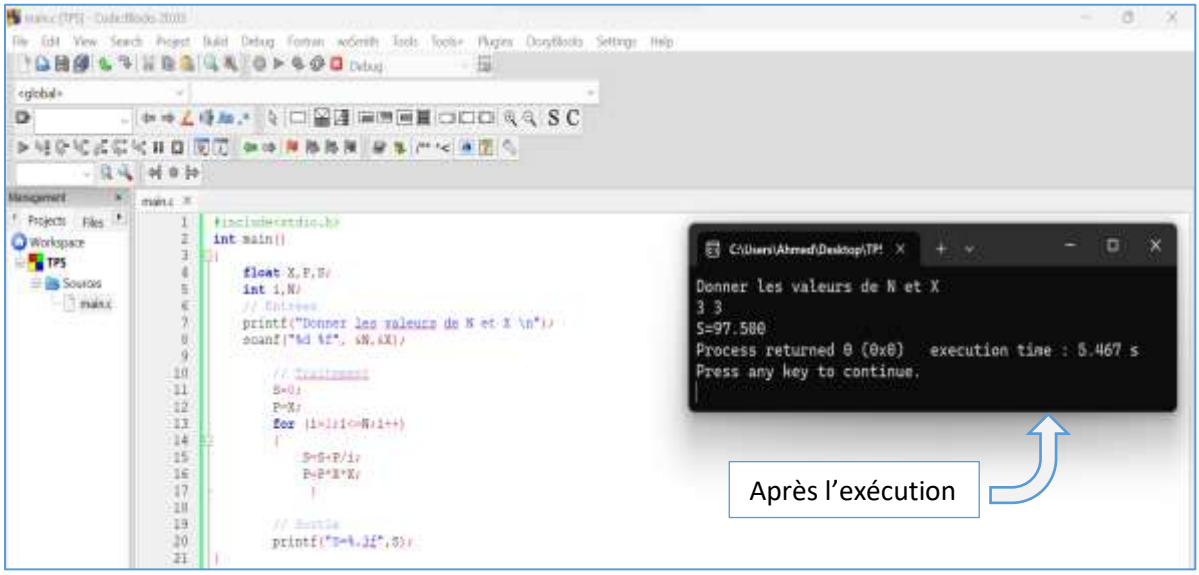


7) Modifier l'algorithme pour calculer la somme S2 :

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

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);
	}

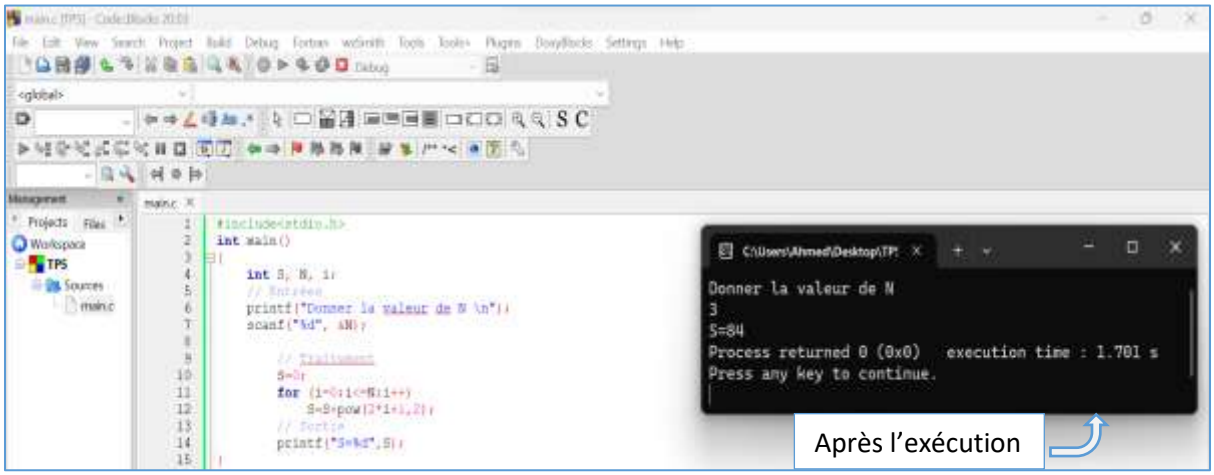


Exercice N°02 : *(Enoncé du problème ➔ Algorithme ➔ Programme en langage C)*

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);
	}

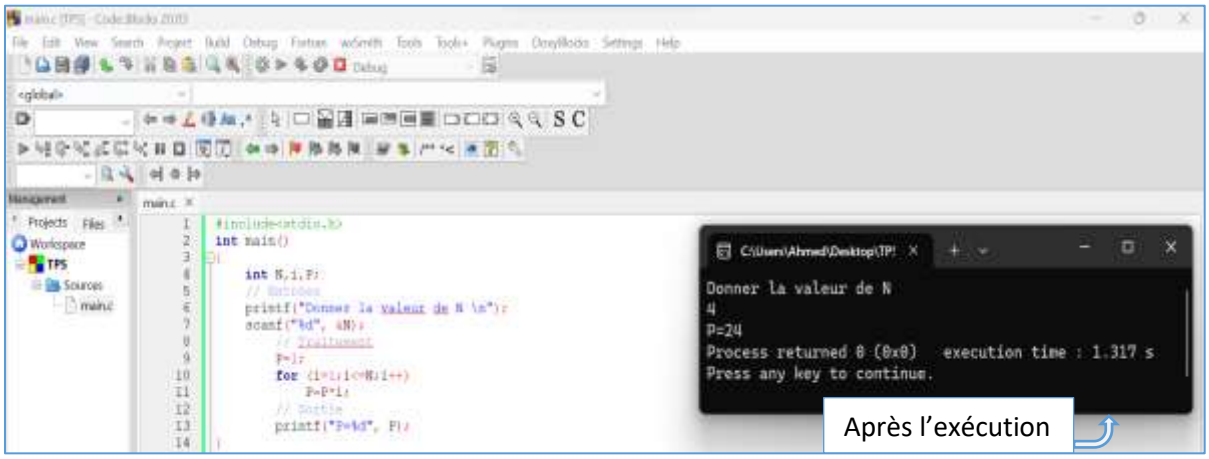


2) Calculer le produit $P = 1 \times 2 \times 3 \times \ldots \times N$

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);
}
```



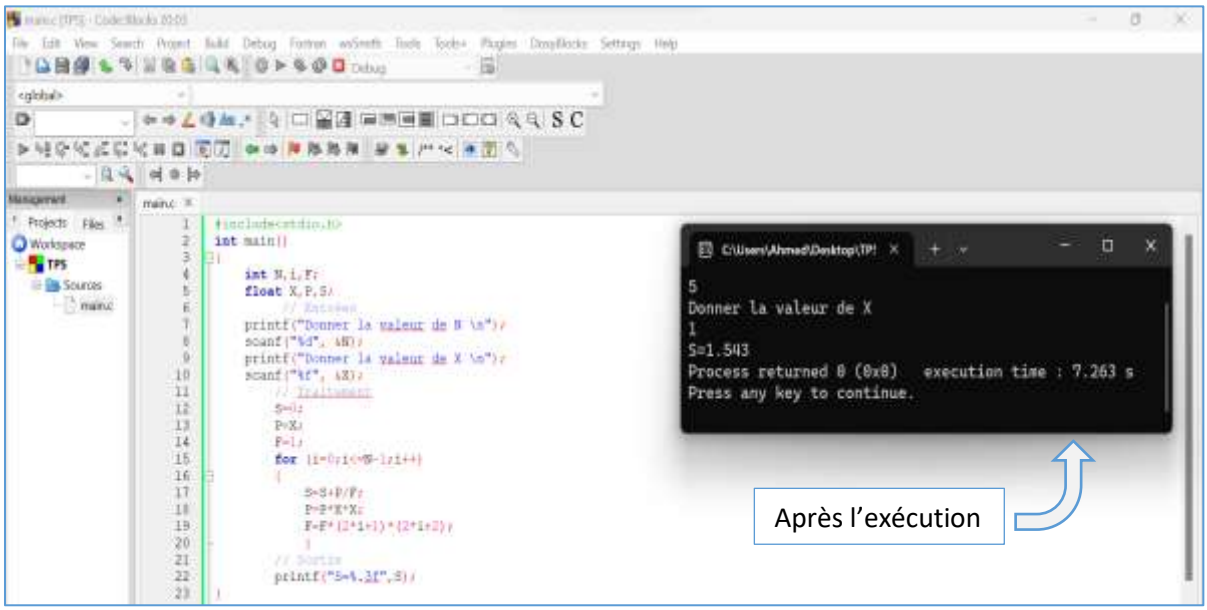
3) Calculer la somme $S = X + \frac{X^3}{2!} + \frac{X^5}{4!} + \frac{X^7}{6!} + \cdots$ ($N^{ème}$ terme)

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);
}
```

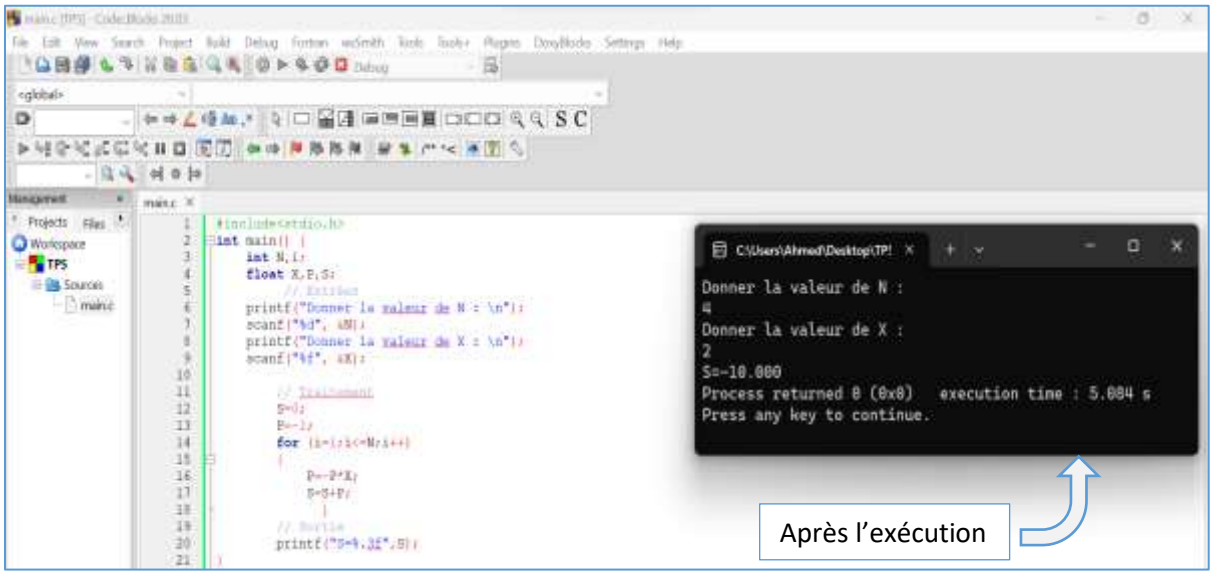


4) Calculer la somme $S = X - X^2 + X^3 - \cdots \pm X^N$

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°4

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.

Bibliographie

Bibliographie

[1] Brookshear, J. G., & Brylow, D. (2024). Computer Science: An Overview (14th ed.). Pearson.

[2] Patterson, D. A., & Hennessy, J. L. (2021). *Computer Organization and Design RISC-V Edition: The Hardware/Software Interface* (2nd ed.). Morgan Kaufmann.

[3] Stallings, W. (2021). *Computer Organization and Architecture: Designing for Performance* (11th ed.). Pearson.

[4] Null, L., & Lobur, J. (2022). *The Essentials of Computer Organization and Architecture* (6th ed.). Jones & Bartlett Learning.

[5] Tanenbaum, A. S., Austin, T., & Bos, H. (2023). *Structured Computer Organization* (7th ed.). Pearson.

[6] Deitel, P., & Deitel, H. (2022). *C How to Program* (9th ed.). Pearson.

[7] King, K. N. (2024). *C Programming: A Modern Approach* (3rd ed.). W. W. Norton & Company.

[8] Reek, K. A. (2023). *The C Programming Language for Engineers and Scientists*. CRC Press.

[9] Forouzan, B. A., & Gilberg, R. F. (2020). *Computer Science: A Structured Programming Approach Using C* (4th ed.). Cengage Learning.

[10] Kelley, A., & Pohl, I. (2021). *A Book on C: Programming in C* (5th ed.). Addison-Wesley.

[11] Downey, A. (2024). *Think C: C Programming for Beginners and Professionals*. Green Tea Press.

[12] Gookin, D. (2022). *Beginning Programming with C For Dummies* (2nd ed.). Wiley.

[13] Sipser, M. (2021). *Introduction to the Theory of Computation* (4th ed.). Cengage Learning.

[14] Vahid, F., & Lysecky, S. (2021). *Computer Organization and Design Fundamentals*. Cypress Publishing.

[15] Schneider, G. M., & Gersting, J. L. (2021). *Invitation to Computer Science* (9th ed.). Cengage Learning.