الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
جامعة وهران للعلوم والتكنولوجيا محمد بوضياف
كلية الرياضيات و الاعلام الالي
قسم الإعلام الآلي
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Oran Mohamed-Boudiaf
Faculté des Mathématiques et Informatique
Département d'Informatique

Polycopié Pédagogique

Notes de cours et recueil d'exercices corrigés

Algorithmique et Structure de Données 1

Travaux Pratiques, destiné aux étudiants de :

Licence : 1ère Année Mathématiques & Informatique

Dr. Mohamed Amine NEMMICH

Maitre de Conférences

Année Universitaire : 2021/2022

Table des matières

Objectifs

Ce document est rédigé à l'intention des étudiants de première année du premier cycle universitaire (licence), Socle Commun en Mathématiques et Informatique de l'université des Sciences et de la Technologie d'Oran - Mohamed Boudiaf.

L'objectif est de présenter aux étudiants un **rappel du cours algorithmique et structures de données** et de proposer une **série d'exercices** traités en séance de TP **avec solutions**.

Les exercices sont choisis sur une partie du domaine de programmation de sorte à aider les étudiants à développer une solution et un programme informatique (en langage C) qui résout un problème proposé.

Ce document est structuré en six chapitres comme suit :

- Le premier chapitre commence par une **brève histoire de l'informatique** et une **introduction à l'algorithmique**.
- Dans le deuxième chapitre, nous présentons un rappel sur **les notions de base** sur l'algorithmique (et la programmation C), c.-à-d. comment écrire un algorithme et un programme qui résout un problème donné tout, à savoir : utiliser les **instructions de base** (affectation, lecture, écriture, entrées/sorties), définir les notions de **variable** et **constante**, définir le **type** d'une variable**.**
- Dans le troisième chapitre, un résumé sur **les structure conditionnelle simple, les structure conditionnelle composée, les structure conditionnelle à choix multiple, ainsi que le branchement**.
- Le quatrième chapitre est consacré aux **différentes structures de contrôles (boucles)** qui peuvent être utilisées dans un algorithme ou un programme (ex. pour, tant que, répéter, boucles imbriquées, ..).
- Le cinquième chapitre décrit les éléments liés à la manipulation des **tableaux et des chaînes de caractères**, et illustre leurs utilisation.
- Le chapitre six étudie **les types personnalisés (énumérations, enregistrements,…).**

Chacun de ces chapitres se termine par une série d'exercices avec solutions proposées en langage C, servant en partie de base aux séances de travaux pratiques.

Il existe de nombreux **environnement de développement C** (ou IDE, en anglais Integrated Development Environment). Voici une liste non exhaustive d'IDE gratuits : Dév C++, Constructeur C ++, Visual Studio, CodeLite, KDevelop, CLion par JetBrains, L'Éclipse, Cévelop, Studio GNAT et Code::Blocs.

Chapitre 1 : Introduction

1. Bref historique sur l'informatique

Les peuples primitifs ont été les premiers à utiliser le dispositif de comptage. Ils utilisaient des pierres, des os et des bâtons comme outils de comptage. Au fil du temps, l'esprit humain et la technologie ont fait un bond en avant et le développement d'autres dispositifs informatiques a commencé.

Le mot **informatique** a été créé en 1962 par Philippe Dreyfus. Il s'agit d'un néologisme de la langue française fait de la contraction des deux mots « automatique » et « information ». Pour parler du traitement automatique de l'information, les anglo-saxons utilisent les termes de « computer science » ou de « data-processing ».

L'abaque est considéré comme le premier ordinateur et l'histoire de l'informatique commence donc avec lui. Selon la croyance actuelle, le boulier a été mis au point il y a environ 4000 ans par les Chinois. Il s'agit d'un support en bois contenant des tiges métalliques sur lesquelles sont montées des perles. L'opérateur du boulier déplaçait les perles pour effectuer des calculs arithmétiques en suivant un ensemble de règles. Certains pays comme le Japon, la Russie et la Chine utilisent encore le boulier.

Une étape historique a été la construction de la pascaline au XVIIe siècle, machine inventée par Blaise Pascal qui effectue les quatre opérations arithmétiques classiques. Une première machine à calculer programmable (technologie inspirée de celle des métiers à tisser) est inventée en 1834, mais jamais réalisée à son époque par Charles Babbage. Les programmes, écrits sur des cartes perforées, technique utilisée jusqu'au milieu des années 1980, sont inventés par la mathématicienne Ada Lovelace, qualifiée de « première programmeuse au monde ». Dès le début du XX e siècle, des entreprises naissantes comme IBM ou Bull, ont créé des machines (mécaniques) enregistreuses, additionneuses et multiplicatrices. Elles furent vendues par milliers aux entreprises et administrations. À la fin du XIXe siècle, l'électricité permet de motoriser ces calculateurs. C'est le début de l'électromécanique. La micro-électromécanique n'a cependant émergé que dans les années 1970. L'ordinateur personnel a connu son essor dix ans plus tard.

Parallèlement à ces avancées technologiques, des idées ont, elles aussi, contribué aux progrès scientifiques. Les algorithmes les plus anciens sont attestés par des tables datant de l'époque d'Hammurabi (env. -1750 av. J.-C.) en Mésopotamie. De nombreux autres algorithmes furent décrits par la suite. C'est toujours le cas aujourd'hui : la compression des images en jpeg s'appuie sur des algorithmes datant de la fin des années 1990.

En logique (domaine des mathématiques lié à l'informatique), George Boole démontre que tout processus logique est décomposable en une suite d'opérations logiques (ET, OU, NON) appliquées sur deux états (VRAI-FAUX). Des questions plus profondes comme celle consistant à savoir si un mécanisme permet d'affirmer si une proposition est vraie ou fausse furent soulevées par David Hilbert dès 1928. Les logiciens Kurt Gödel et Alan Turing ont aussi participé à des avancées dans ce domaine de la logique.

La machine de Turing (1936) est un modèle abstrait permettant de mettre en œuvre n'importe quel algorithme. C'est en quelque sorte le modèle abstrait d'un ordinateur où la technologie permet de répondre à des questions algorithmiques.

De nombreux calculateurs programmables, dont les plus connus sont ceux appelés bombes

(à cause du bruit engendré par leur fonctionnement) et dont le but était d'aider à décrypter les messages allemands durant la seconde guerre mondiale, sont construits entre 1936 et 1956. Alan Turing a participé à la conception de ces bombes.

En 1948, Claude E. Shannon, précurseur de la théorie de l'information, popularise l'utilisation du mot bit comme mesure élémentaire de l'information numérique, information qui devient mesurable et réductible à deux signaux élémentaires (notés 0 et 1).

À partir de 1948 apparurent les premières machines à architecture de Von Neumann. Ce modèle d'architecture utilise une structure unique de stockage pour les données et les instructions ; un ordinateur peut alors modifier les instructions, effectuer des boucles, ce que les programmes venant de cartes perforées ne permettaient pas. Tous les ordinateurs actuels possèdent une architecture issue de celle décrite par John Von Neumann.

L'architecture de Von Neumann (Figure 1) décompose l'ordinateur en quatre parties distinctes :

- l'unité arithmétique et logique qui effectue les opérations de base ;
- l'unité de contrôle, chargée du « séquençage » des opérations ;
- la mémoire qui contient à la fois les données et le programme qui indiquera à l'unité de contrôle quels sont les calculs à effectuer sur ces données. La mémoire se divise en mémoire volatile (programmes et données en cours de fonctionnement) et mémoire permanente (programmes et données de base de la machine) ;
- les dispositifs d'entrée-sortie (périphériques) pour communiquer avec l'extérieur.

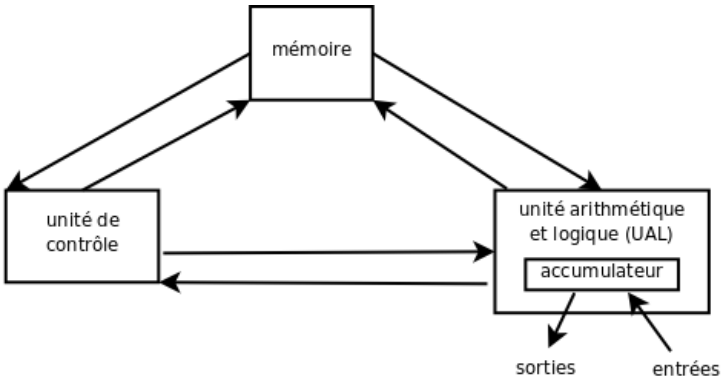


Figure 1. Architecture de Von Neumann

2. Introduction à l'algorithmique

Le mot « **algorithme** » vient de la transcription latinisée d'Al-Kwharizmi, nom d'un célèbre mathématicien arabe, et du mot grec *arithmos* qui signifie « nombre ».

Un algorithme est l'ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations. Il peut être traduit grâce à un langage de programmation, en un programme exécutable par un ordinateur ou une calculatrice.

L'algorithmique : désigne la discipline qui étudie les algorithmes et leurs applications en informatique.

Propriétés requises pour un algorithme selon Donald Knuth, professeur à l'université de Stanford:

- La **finitude** : « Un algorithme doit toujours se terminer après un nombre fini d'étapes. » définition précise : « Chaque étape d'un algorithme doit être définie

précisément, les actions à transposer doivent être spécifiées rigoureusement et sans ambiguïté pour chaque cas. »
- **entrées** : « ... des quantités qui lui sont données avant qu'un algorithme ne commence. Ces entrées sont prises dans un ensemble d'objets spécifié. »
- **sorties** : « ... des quantités ayant une relation spécifiées avec les entrées. »
- **rendement** : « … toutes les opérations que l'algorithme doit accomplir doivent être suffisamment basiques pour pouvoir être en principe réalisées dans une durée finie par un homme utilisant un papier et un crayon. »

Le **processus général de résolution algorithmique** (ou informatique) d'un problème peut grossièrement se décomposer en trois phases :

LE PROBLÈME → L'ALGORITHME → LE PROGRAMME

Le **problème** :

- Préciser les spécifications : données ? résultats ? environnement ?
- Modélisation : structures mathématiques ? propriétés ? problème déjà connu, ... ?
- Modularisation : décomposition en sous problèmes, sous problèmes plus simples, indépendants, ... ?

L'algorithme :

- Conception : méthodes de conception d'algorithmes ?
- Écriture dans un pseudo langage ?
- Structures de données ?
- Analyse : preuve (fin, validité), efficacité (complexités) ?

Le **programme** :

- Traduction de l'algorithme dans un langage de programmation précis dans un environnement précis ?
- Mise au point : tests ? documentation ? ...

Informatiser une application, c'est faire réaliser par un ordinateur, une tâche qui était réalisée par l'Homme. Pour cela il faut tout d'abord, détailler suffisamment les étapes de résolution du problème, pour qu'elles soient exécutables par l'homme Ensuite, transférer la résolution en une suite d'étapes élémentaires et simples à exécuter.

Chapitre 2 : Algorithme séquentiel simple

A. Rappel du cours

1. Notion de langage et langage algorithmique

Un **algorithme** est la description précise de la méthode de résolution d'un problème sous forme d'**instructions** simples. L'algorithme est caractérisé par :

- Identification des informations intervenant dans un programme
- Abstraction : modélisation, formalisation et représentation de ces informations
- Énumération des opérations à appliquer sur ces informations.
- Définition de l'ordre des opérations à respecter.

Langage Algorithmique : Tout algorithme est exprimé dans un langage naturel appelé Langage Algorithmique (LA) ou **pseudo-langage**, il décrit de manière structurée, claire et complète, les objets manipulés par l'algorithme ainsi que l'ensemble des instructions à exécuter sur ces objets pour obtenir des résultats.

Un **programme** est la traduction de l'algorithme dans un langage formel que la machine peut comprendre et exécuter. Ce langage est dit **langage de programmation** (Pascal, C, Java,...) et il est régi par une syntaxe tout autant que l'est le langage naturel de l'homme.

2. Parties d'un algorithme/programme C

Structure d'un algorithme :

Un algorithme est composé de **3 parties** :

- **Entête** : est la partie où est défini le nom de l'algorithme.
- **Déclaration**: est la partie où sont citées les déclarations des objets (lister les entités variables, constantes et type). Le type est un attribut qui indique la nature d'un objet et surtout sur le genre d'actions qu'on peut lui appliquer.
- **Corps** : est la partie où sont citées les opérations et les instructions.

Entête	Algorithme nom_algorithme ;
Déclaration	**const** nom_constante1=valeur1 ; nom_constante2=valeur2 ; ..... **var** nom_var1 : type ; nom_var2 : type ; .....
Corps	**début** instruction1 ; instruction2; **fin**.

Structure d'un programme :

Le **langage C** est un langage de programmation de bas niveau, ce qui signifie que les instructions figurant dans les programmes restent proches de celles que le processeur est directement capable d'exécuter. Apprendre à programmer en commençant par le langage C permet d'acquérir une bonne compréhension des mécanismes qui régissent l'exécution des programmes, notamment la façon dont le processeur accède à la mémoire de l'ordinateur. Un autre avantage de ce langage est qu'il est simple. Cette simplicité ne fait cependant pas obstacle à son utilisation pour des projets complexes : de très nombreux logiciels de grande taille sont programmés en C. Par exemple, le noyau du système d'exploitation Linux, qui équipe notamment une majorité des smartphones modernes, est principalement rédigé en C, et comprend plusieurs dizaines de millions de lignes de code.

Un programme simple se compose de plusieurs parties :

- des directives de pré-compilation
- une ou plusieurs fonctions dont l'une s'appelle obligatoirement main().

La fonction **main()** commence par une accolade ouvrante { et se termine par une accolade fermante }. Entre les accolades il se trouve des déclarations et instructions, chaque instruction se termine par un point-virgule. Toute variable doit être déclarée.

Syntaxe d'un algorithme / un programme C :

Syntaxe d'un algorithme	**Syntaxe d'un programme C**
Algorithme nom de l'algorithme ; **Const** déclarations des constantes **Var** déclarations des variables **Début** Instructions ; **Fin**.	**#include<stdio.h>** **#define** NomConst ValConst **main()** **{** Déclarations des variables ; Instructions ; **}**

3. Les données : variables et constantes et types de données

Un programme en langage C manipule des données. Une donnée possède un type et une valeur, elle peut avoir un nom et peut être modifiable (variable) ou constante.

Une **variable** est un espace mémoire nommé qui peut prendre au cours de l'exécution de l'algorithme (ou d'un programme) un nombre indéfini de valeurs. La partie déclaration de variable permet de spécifier toutes les variables qui seront utilisées dans la partie action ainsi que le type de valeurs qu'elles doivent respectivement prendre.

Le langage C fait la différence entre les MAJUSCULES et les minuscules. Les noms des variables doivent commencer par une lettre et ne contenir aucun blanc. Le seul caractère spécial admis est le soulignement (_). Il existe un certain nombre de noms réservés (while, if, case, ...), dont on ne doit pas se servir pour nommer les variables. De plus, on ne doit pas utiliser les noms des fonctions pour des variables.

Dans un programme, il arrive que certaines données ne changent pas. Ce sont des **constantes** dont la déclaration et l'initialisation est un peu différente de celles des variables. Par exemple, le calcul du périmètre d'un cercle fait intervenir son diamètre (qui dépend du cercle) et le nombre Pi (qui ne change pas).

Le **Type** permet au programmeur de définir l'ensemble de valeurs que peut prendre une variable. Le type est indispensable au compilateur pour la représentation mémoire des données (caractère : 8bits, entier : 16bits), et pour l'ensemble des opérations applicables sur ces données.

Types **prédéfinis** on dit aussi **types de base** = {Entier, Réel, Caractère, Booléen}

Les valeurs de chaque type sont par exemple :

- **Entier** : 0,-3,+754 ...
- **Réel** : 14.5, -0.78, 6.00...
- **Booléen** : {vrai, faux}.
- **Caractère** : `A` , `B`, `C`, `0`, `9`, ` ? `, `,` , ...

Il est impératif de respecter le type des objets quand il s'agit de les instancier par des valeurs.

Pour déclarer une variable dans le langage C, on fait précéder son nom par son type. Il existe 6 types de variables :

- **char** : caractère codé sur 1 octet (8 bits)
- **short** : entier codé sur 1 octet
- **int** : entier codé sur 4 octets
- **long** : entier codé sur 8 octets
- **float** : réel codé sur 4 octets
- **double** : réel codé sur 8 octets

On peut faire précéder chaque type par le préfixe **unsigned**, ce qui force les variables à prendre des valeurs uniquement positives.

Exemple de déclarations des variables :

Algorithme	Programme C
N : entier; M : réel;	int N; float M;

Exemple de déclarations des constantes :

Algorithme	Programme C

Constante PI=3.14, MAXI=32;	#define MAXI 32 ;
	main( )
	{
	const float pi=3.14;
	….

Notion de contenu :

La déclaration d'une variable sert à réserver un **espace mémoire** identifié par le nom de la variable dans un but de contenir une valeur. Un moyen d'attribuer une valeur à un espace mémoire est l'instruction d'affectation ou l'instruction lire. Le contenu d'une case mémoire c'est la valeur d'une variable.

Exemple :

Une succession d'instructions d'affectations avec une représentation de l'évolution des contenus des espaces mémoires de variables A et B.

- A <- 3;
- B <- A+2;
- A <- A+4;

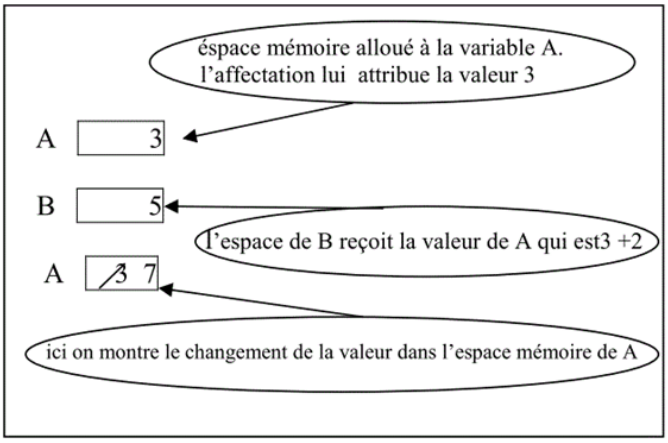


Figure 2. Notion de contenu

De l'exemple (Figure 2) on note :

- Ce qui est représenté par des rectangles ce sont les cases mémoires attribuées pour chaque variable.
- On dit qu'avec l'instruction c) la valeur de A qui était 3 est écrasée par une autre valeur 7.
- B ne change pas de valeur ce qui explique que chaque case mémoire ne dépend que de l'instruction qui la modifie.

4. Les opérations et les instructions de base

Les opérateurs permettent de **fabriquer des expressions** et d'agir sur le contenu de variables. Ils effectuent un calcul numérique (opérateurs mathématiques), fournissent une condition logique pour un test (opérateurs logiques et relationnels), agissent sur certains bits d'une variable (opérateurs de manipulation de bits) ou réalisent des opérations spécifiques (affectation, conversions, obtention d'une adresse, etc).

A un type donné, correspond un ensemble d'opérations définies pour ce type :

Type	**Opération possibles**	**Symbole ou mot correspondant**
Entier / Réel	Addition Soustraction Multiplication Division Division entière Modulo (le reste de la division entière) x exposant y Comparaisons	+ - * / / (entre deux entiers) (% en C) pow(x,y) en C <, =, >, <=, >=, <>, ==, !=
Caractère / Chaîne	Comparaisons Concaténation (sur les chaines)	<, =, >, <=, >=, <>, ==, != &
Booléen	Logiques Comparaisons	ON(!), OU(||), ET (&&) <, =, >, <=, >=, <>, ==, !=

Il existe plusieurs **types d'instructions**.

1. Expression arithmétique :

Pour résoudre le problème d'ambiguïté, des règles de priorité sont introduites sur les opérateurs :

- Priorité de {* , / } est plus grande que celle de {+, -} : les opérateurs (*, /) sont prioritaires sur (+, -).
- Les opérateurs (*, /) ont la même priorité.

- De même en ce qui concerne les opérateurs (+, -).

Remarque : les évaluations se font de gauche à droite.

En C, il existe un certain nombre d'opérateurs spécifiques, qu'il faut utiliser prudemment sous peine d'erreurs.

- **++** incrémente la variable d'une unité.
- **- -** décrémente la variable d'une unité.

Ces 2 opérateurs ne s'utilisent pas avec des réels. Quand l'opérateur ++ est placé avant une variable, l'incrémentation est effectuée en premier. L'incrémentation est faite en dernier quand ++ est placé après la variable. Le comportement est similaire pour - -.

2. L'Affectation :

L'affectation permet d'attribuer :

- une valeur à une variable.
- un contenu à une zone mémoire.

Syntaxe : nom_variable <- <expression>;

Exemple : Surf <- ray*ray*Pi ;

L'affectation sert à mettre dans la variable de gauche (Surf) la valeur de ce qui est à droite. Le membre de droite est d'abord évalué, et ensuite, cette valeur est attribuée à la variable de gauche.

Remarque :

- La valeur de l'expression est de type compatible avec nom _variable.
- L'instruction d'affectation consiste à évaluer la valeur de l'expression et puis l'affecter (ou l'attribuer) ou (la mettre) comme contenu de la zone mémoire identifiée par nom_variable.

Dans le **langage C** l'instruction d'affectation est incluse dans la liste des opérateurs au même titre que les opérateurs naturels +,-. L'**affectation** est représenté par **"="**.

Exemples Instruction d'affectation :

Algorithme	**Programme C**
N <- 10; M <- 1,5; Nom <- "mohamed";	N = 10; M = 1.5; Nom = "mohamed";

3. Les instructions d'entrées/sorties :

Les techniques d'échange entre l'ordinateur (mémoire centrale) et son extérieur (l'environnement, l'utilisateur) sont appelés techniques d'Entrée/Sortie, (E/S : FR, I/O : ANG). L'ordinateur échange des informations non seulement avec les opérateurs humains (utilisant clavier ou souris...) mais également avec des dispositifs variés (capteurs, appareil de mesure...).

a. L'ordre de sortie : (Instruction Écrire)

L'objectif d'un ordre de sortie est de visualiser une ou plusieurs valeurs (résultats) qui existe dans la mémoire centrale et éventuellement accompagnées de messages (une chaîne de caractères).

Syntaxe de l'instruction écrire : Écrire (expression/message, …) ;

- **Message** : chaîne de caractères entre guillemets ″.
- **Expression** : soit une valeur d'une case mémoire : donc expression est le nom de la variable qui contient cette valeur. Ou bien la valeur d'une constante dont expression est soit le nom de la constante ou la valeur de la constante.

En langage C, la **fonction printf()** sert à afficher à l'écran la chaîne de caractère donnée en argument, c'est-à-dire entre parenthèses.

Exemple :

Algorithme	Programme C
ecrire("message") ; ecrire(N,M) ; ecrire("message",N) ;	printf("message") ; printf("%d%f",N,M) ; printf("message%d",N) ;

En langage C, Le **caractère %** indique le format d'écriture à l'écran. Dès qu'un format est rencontré dans la chaîne de caractère entre " ", le programme affiche à sa place la valeur de l'argument correspondant.

Les formats utilisables sont :

- **%d** : **integer** entier (décimal)
- **%u** : **unsigned** entier non signé (positif)
- **%hd** : **short** entier court
- **%d** : **long** entier long
- **%f** : **float** réel, notation avec le point décimal (ex. 123.15)
- **%e** : **float** réel, notation exponentielle (ex. 0.12315E+03)
- **%lf** : **double** réel en double précision, notation avec le point décimal
- **%le** : **double** réel en double précision, notation exponentielle
- **%c** : **char** caractère
- **%s** : **char** chaîne de caractères

b. L'ordre d'entrée : (Instruction Lecture)

Cette instruction bloque le processeur (le met en attente) jusqu'à ce que l'utilisateur tape une valeur au clavier de même type que nom_variable, cette valeur sera mise (comme contenu) dans la zone mémoire de nom_variable.

Syntaxe : lire (nom_variable) ;

Pour le langage C, la fonction **scanf()** permet de lire la valeur que l'utilisateur entre (tape) au clavier et de la stocker dans la variable donnée en argument.

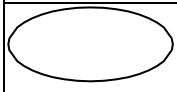
On retrouve les formats de lecture précisés entre " " utilisés pour printf(). Pour éviter tout risque d'erreur, on lit et on écrit une même variable avec le même format. Le **&** est indispensable pour le bon fonctionnement de la fonction. Il indique l'adresse de la variable.

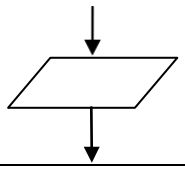
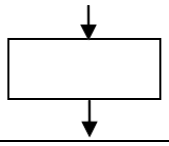
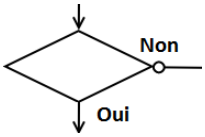
Algorithme	Programme C
lire(N,M) ;	scanf("%d%f",&N,&M) ;

5. Construction d'un algorithme simple et représentation par un organigramme

Pour représenter un algorithme, deux méthodes : une représentation graphique par un organigramme et une représentation textuelle par un pseudocode.

Un **organigramme** présente l'avantage d'être visuel. Il utilise un ensemble de symboles normalisés (ISO 7805). Les étapes de l'algorithme sont représentées sous forme de boites reliées par des flèches. Pour cela, on utilise les symboles géométriques suivants :

Symbole	Signification
	Ce symbole est utilisé pour représenter le début ou la fin de l'algorithme.

	Ce symbole est utilisé pour représenter la mise à la disposition du programme d'une donnée à traiter ou une opération d'enregistrement d'une donnée à effectuer.
	Le Procédé (symbole général de traitement) est utilisé pour représenter une opération (un groupe d'opérations) sur des données ou pour les instructions pour lesquelles il n y a pas de symbole normalisé.
	Ce symbole est utilisé pour représenter une portion de programme considérée comme une simple opération (sous-programme, procédure, fonction).
	Branchement (décision) : Ce symbole est utilisé pour représenter le test et l'exploitation d'une condition pour faire la sélection entre deux choix.

La représentation textuelle ou langage structuré (pseudocode) utilise un ensemble de mots clés (langage) et de structures permettant de décrire de manière complète et claire l'ensemble des opérations à exécuter sur des données pour obtenir des résultats. C'est une composition séquentielle finie d'opérations comme celle de l'organigramme. Donc, tout algorithme représenté par un organigramme peut être représenté à l'aide du pseudocode. L'avantage d'un tel langage est de pouvoir être facilement transcrit dans un langage de programmation procédurale (C, ...).

Un algorithme se présente en général sous la forme suivante :

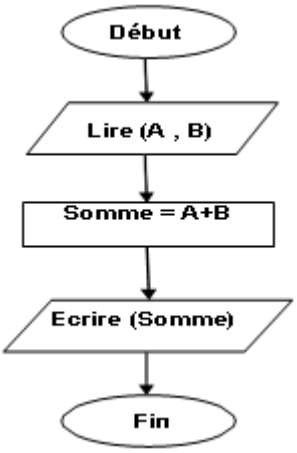
- **Déclaration des variables** : On décrit dans le détail les éléments que l'on va utiliser dans l'algorithme (variables, constantes et structures),
- **Initialisation ou Entrée des données** : On récupère les données et/ou on les initialise (par lecture et par affectation),
- **Traitement des données** : On effectue les opérations nécessaires pour répondre au problème posé,
- **Sortie** : On affiche les résultats (par l'écriture)

L'exemple suivant représente la structure algorithmique de base sous forme de pseudocode (et son équivalent en programmation C) qui permet de faire l'addition de deux entiers : A et B.

Algorithme	**Programme C**

Algorithme addition ;
Var A, B, Somme : entier ;
Début
Lire (A) ;
Lire (B) ;
Somme ← A+B ;
Ecrire ('La somme=',Somme) ;
Fin.
	#include <stdio.h>
	main()
	{
	Int A, B, Somme ;
	scanf ("%d",&A);
	scanf ("%d",&B);
	Somme = A + B ;
	printf ("La somme= %d\n", Somme);
	}

Cet algorithme (ou ce programme C) peut être représenté sous forme d'organigramme suivant :



B. Travaux Pratiques N° 1

Cette rubrique présente une série d'exercices avec leurs corrigé sur les **expressions arithmétiques et logiques, les affectations et les entrées/sorties**.

1. Exercices

a) Exercice 1.1

Ecrire un programme C qui affiche la phrase suivante « Hello, world ».

b) Exercice 1.2

Ecrire un programme C permettant la saisie d'une note et son affichage.

a) Exercice 1.3

Ecrire un programme C qui saisit deux entiers et calcul et affiche leur somme. Ajouter au niveau du même programme les opérations suivantes : produit, différence et moyenne.

b) Exercice 1.4

Ecrire un programme C qui réalise l'échange de deux valeurs entières saisis (la permutation). Afficher les entiers avant et après l'échange.

c) Exercice 1.5

Ecrire un programme C qui calcule et affiche la surface et le périmètre d'un rectangle dont on lui donne la longueur et la largeur.

d) Exercice 1.6

Écrire un programme C qui permet de calculer et afficher la moyenne générale d'un étudiant en introduisant ses notes et le coefficient de chaque module par le clavier, sachant que l'étudiant suit 03 modules.

e) Exercice 1.7

Ecrire un programme C qui permet de calculer la longueur L d'un câble entre deux pylônes, en utilisant à la formule suivante, où a est la distance entre les pylônes et f la flèche mesuré perpendiculairement au milieu du câble.

$$L = a\left(1 + \frac{2}{3}\left(\frac{2f}{a}\right)^2\right)$$

f) Exercice 1.8

Écrire un programme qui, à partir d'un entier R fixé dans le programme, affiche le diamètre, la circonférence et l'aire du cercle de rayon R.

2. Corrigés d'exercices

a) Corrigé d'exercice 1.1

```
/*Inclusion de la bibliothèque standard*/
#include <stdio.h>
main ()
{
      printf("Hello, world\n");
}
```

b) Corrigé d'exercice 1.2

```
#include <stdio.h>
void main()
{
  /*Déclaration de la note*/
  int iNote;
  clrscr();
  /*Saisis au clavier la note*/
  printf("SVP Taper la note:");
  scanf("%d",&iNote);

  /* Affichage de la note*/
  printf("la note est %d",iNote);
  getch();
}
```

c) Corrigé d'exercice 1.3

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

int main()
{
  /*Déclaration de deux entiers*/
  int Facteur1, Facteur2;

  /*Saisis au clavier du premier entier*/
  printf("valeur du premier facteur: ");
  scanf("%d",&Facteur1);

  /*Saisis au clavier du deuxième entier*/
  printf("valeur du deuxième facteur: ");
  scanf("%d",&Facteur2);

  /* Affichage du somme*/
  printf("Somme de %d et %d = %d \n",Facteur1, Facteur2, Facteur1+Facteur2);

  /* Affichage du produit*/
  printf("Produit de %d et %d = %d \n",Facteur1, Facteur2, Facteur1*Facteur2);

    /* Affichage du Difference*/
  printf("Difference entre %d et %d = %d \n",Facteur1, Facteur2, Facteur1-
Facteur2);
```

```
  /* Affichage du Moyenne*/
  printf("Moyenne de %d et %d = %f \n",Facteur1, Facteur2,
(float)(Facteur1+Facteur2)/2);
  getch();
}
```

d) Corrigé d'exercice 1.4

```
#include <stdio.h>
#include <conio.h>
void main()
{
  /*Déclaration de deux entiers*/
  int Entier1, Entier2;
  /*Déclaration d'entier pour l'échange*/
  int Echange;

  /*Saisis au clavier du premier entier*/
  printf("Entrer la valeur du premier entier: ");
  scanf("%d",&Entier1);

  /*Saisis au clavier du deuxième entier*/
  printf("Entrer la valeur du deuxième entier: ");
  scanf("%d",&Entier2);

  /*Echange des entiers*/
  Echange=Entier1;
  Entier1=Entier2;
  Entier2=Echange;

  /* Affichage des entiers echanges*/
  printf("Le 1er entier = %d, et le 2ieme = %d", Entier1,Entier2);
  getch();
}
```

e) Corrigé d'exercice 1.5

```
#include<stdio.h>
main(){
      float longueur, largeur, surface, perimetre;
      printf("Donnez la longueur: "); scanf("%f",&longueur);
      printf("Donnez la largeur: "); scanf("%f",&largeur);
      surface = longueur * largeur;
      perimetre = (longueur+largeur)*2;
      printf("La surface = %f\n",surface);
      printf("Le périmetre = %f\n",perimetre);
}
```

f) Corrigé d'exercice 1.6

```
#include<stdio.h>
main(){
      const int coef1=2, coef2=3, coef3=4;
      float note1, note2, note3, moyenne;
      printf("Note 1: "); scanf("%f",&note1);
      printf("Note 2: "); scanf("%f",&note2);
      printf("Note 3: "); scanf("%f",&note3);
```

```
       moyenne = (note1*coef1+note2*coef2+note3*coef3)/(coef1+coef2+coef3);
       printf("La moyenne = %f\n",moyenne);
}
```

g) Corrigé d'exercice 1.7

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int main()
{
       float L, f, a;
       printf("Distance entre les pylones : ");
       scanf("%f",&a);

       printf("Fleche (mesuree au milieu du cable) : ");
       scanf("%f",&f);

       L = a*(1.0+2.0/3.0*pow(2.0*f/a,2));

       printf("Longueur de cable = %f\n", L);
       getch();
}
```

h) Corrigé d'exercice 1.8

```
#include <stdio.h>
    #include <conio.h>
    #define PI 22.0/7 //Declaration de la constante PI
    int main(){
        float r, d, c, s;

      printf("Entrer le rayon R : ");
      scanf("%f",&r);

      d = 2 * r;
      c= d * PI;
      s = PI * r*r ; //ou s = 3.14 * r *r;

       printf("\n\nLe diamètre du cercle de rayon %.2f est  : %.2f\n",r , d);
       printf("\n\nLa circonférence du cercle  est  : %.2f\n", c);
       printf("\n\nLa Surface est  : %.2f\n",s);
       getch();
    }
```

Chapitre 3 : Les structures conditionnelles

A. Rappel du cours

1. Introduction

Parfois les instructions ne sont réalisées (exécutées par le processeur) que si certaines conditions sont vérifiées. C'est ce qu'on appelle instruction conditionnée.

Les **instructions conditionnelles** (ou structures de contrôle) permettent de réaliser des tests et de sélectionner la séquence d'instructions à exécuter suivant le résultat de la condition ou expression. Il existe trois types de tests différents : simple, avec alternative, et multiple.

2. Structure conditionnelle simple

C'est suivant le résultat d'une condition qu'à lieu la sélection du bloc d'instructions à exécuter. Une valeur logique vraie est représentée par un entier positif non nul et par l'entier 0 si elle est fausse.

Cette structure existe sous trois formes :

Syntaxe de la première forme :

Algorithmique	**Langage C**
Si (condition) **alors** bloc ;	**if** (condition) bloc ;

Bloc :

Début
Instruction 1:
.....
Instruction n ;
Fin

Remarque : Si bloc contient une seule instruction, les délimiteurs ‘Début...Fin ‘sont omis et en C, les accolades ‘{} ‘sont omises. Si condition est vraie alors ce sont les instructions de bloc qui seront exécutées sinon, le programme continue en séquence (à la suite, après l'instruction Si (if)).

Une **condition** est une expression composée d'opérateurs relationnels (parfois aussi d'opérateurs arithmétiques et logiques) dont la valeur est vraie ou fausse. Cela peut donc être :

- une condition, ou un test de type.

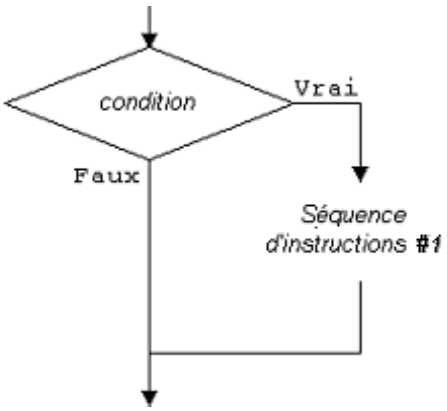


Figure 3. Structure « SI »

Syntaxe de la deuxième forme :

Algorithmique	Langage C
Si (condition) **alors** bloc1 **sinon** bloc2	**if** (condition) bloc1 **else** bloc2

Si la condition est différente de 0 (vraie), alors ce sont les instructions du bloc1 qui sont exécutées. Si par contre la condition est égale à 0, ce seront les instructions de bloc2 qui seront exécutées.

Remarque : La condition peut être composée de plusieurs conditions liées avec les opérateurs && (ET) et || (OU), comme suit :

- **if**(Condition1 && Condition2)
- **if**(Condition1 || Condition2)

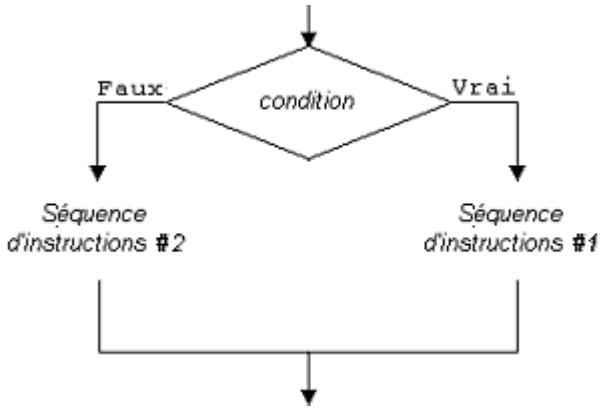


Figure 4. Structures « SI et SI-SINON »

Syntaxe de la troisième forme :

```
expression ? expression1 : expression2 ;
```

Dans le cas où expression est différente de 0 (vraie), alors expression1 est considérée, dans le cas contraire se sera expression2 qui prend effet.

max = (a>b) ? a : b ; // veut dire que max vaut a si a>b ou b sinon

3. Structure conditionnelle composée

La structure conditionnelle composée est formée par une suite d'instruction if imbriquées.

Algorithmique	Langage C
Si (Condition1) **alors** bloc1; **sinon si** (Condition2) bloc2; **sinon si** (Condition3) bloc3;	**If** (Condition1) bloc1; **else if** (Condition2) bloc2; **else if** (Condition3) bloc3; .....

Les blocs d'instructions bloc1, bloc2, peuvent à leur tour contenir des instructions **if**. Un exemple typique est celui de la résolution de l'équation du second degré.

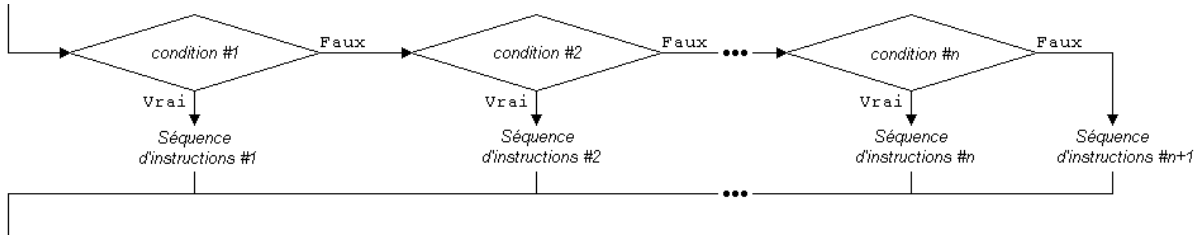


Figure 5. Structure conditionnelle « SI-SINON-SI »

Cette structure conditionnelle est employée lorsqu'on doit exécuter une et une seule séquence d'instructions en fonction d'une condition associée. Cette structure peut être interprétée ainsi :

- exécuter Séquence d'instructions #1 si et seulement si condition #1 est vraie;
- exécuter Séquence d'instructions #2 si et seulement si condition #1 est fausse et condition #2 est vraie;
- exécuter Séquence d'instructions #3 si et seulement si condition #1 et condition #2 sont fausses, mais condition #3 est vraie;
- ...
- exécuter Séquence d'instructions #i si et seulement si condition #1 à condition #i-1 sont fausses, mais condition #i est vraie;
- finalement, si aucune des conditions de la structure n'est vraie et la structure dispose d'une section **SINON**, la Séquence d'instructions #n+1 est exécutée.

4. Structure conditionnelle de choix multiple ou aiguillage

La structure de **choix multiple** consiste à exécuter une suite d'instruction selon la valeur d'une expression discrète comme par exemple, la gestion de menus. Bien qu'il soit possible d'utiliser plusieurs alternatives **if** pour réaliser un choix multiple, le langage C offre une instruction spécialisée pour réaliser l'aiguillage vers différentes instructions selon un choix qui est le **switch**, sa syntaxe est la suivante :

```
switch (expression)
{
        case valeur 1 : instructions ;
                        break ;
```

case valeur 2 : instructions ;
break ;
.....
case valeur n : instructions ;
break ;
default : instructions ;

Le fonctionnement est le suivant :

- expression : a une valeur de type entier ou caractère (type scalaire discret et non réel).
- valeur1, valeur2,... : valeurs de type entier ou caractère (type scalaire discret et non réel)
- L'exécution de l'instruction switch dépend de la valeur d'expression qui est évaluée comme une valeur entière.
- Les valeurs du mot-clé **case** sont des constantes entières. Le branchement vers l'instruction ou les instructions spécifique(s) se fait selon que la valeur d'expression soit égale à la valeur de la case.
- L'exécution des instructions correspondantes continue en séquence jusqu'à la rencontre de l'instruction **break**.
- Si la valeur d'expression n'est égale à aucune des valeurs de case, c'est les instructions suivant le mot-clé **default** qui seront exécutées.

Remarque :

- Les **conditions** sont à la base de tous les programmes. C'est un moyen pour l'ordinateur de prendre une décision en fonction de la valeur d'une variable.
- Les mots-clés **if**, **else if**, **else** signifient respectivement « **si** », « **sinon si** », « **sinon** ». On peut écrire autant de else if que l'on veut.
- Un **booléen** est une variable qui peut avoir deux états : vrai (1) ou faux (0) (toute valeur différente de 0 est en fait considérée comme « vraie »). On utilise des int pour stocker des booléens, car ce ne sont en fait rien d'autre que des nombres.
- Le **switch** est une alternative au **if** quand il s'agit d'analyser la valeur d'une variable. Il permet de rendre un code source plus clair si vous vous apprêtez à tester de nombreux cas. Si vous utilisez de nombreux else if, c'est en général le signe qu'un switch serait plus adapté pour rendre le code source plus lisible.
- Notez que c'est l'instruction **break** qui empêche la poursuite du programme en séquence.

5. Le branchement

L'instruction **goto** permet de brancher (inconditionnellement) à une ligne du programme. Celle-ci doit avoir été étiquetée (précédée d'une étiquette constituée d'un identificateur suivi de :).

Syntaxe :

Etiquette : instruction

goto Etiquette ;

Fonctionnement : Le système interrompt l'exécution séquentielle du programme, remonte ou déscend à la ligne appelée étiquette et poursuit l'exécution à partir de celle-ci.

Exemple :

```
#include <stdio.h>
main( )
{
int i=0 ;                       /*i entier initialisé à 0*/
printf("%d",i);                 /*affiche 0*/
goto message;                   /*saute à l'étiquette message*/
```

```
i++;                              /*ne sera alors pas exécuté*/
printf("%d",i);                  /*ne sera alors pas exécuté*/
message : printf("OK\n");         /*affiche OK*/
printf("FIN\n");                 /*affiche FIN*/
}                                 /*le programme affichera donc 0, OK et FIN*/
```

Remarque :

- **goto** a la réputation de rendre les programmes moins lisibles. Néanmoins, son utilisation est importante dans des cas qui l'impose.

B. Travaux Pratiques N° 2

Cette rubrique présente une série d'exercices avec leurs corrigés sur les **structures conditionnelle et alternatives**.

1. Exercices

a) Exercice 2.1

Écrire un programme C qui accepte deux nombres entiers et vérifie s'ils sont égaux ou non.

b) Exercice 2.2

Écrire un programme C pour vérifier si un nombre donné est pair ou impair.

c) Exercice 2.3

Écrire un programme C pour vérifier si un nombre donné est positif ou négatif.

d) Exercice 2.4

Écrire un programme en C pour accepter la taille d'une personne en centimètres et classer la personne en fonction de sa taille (Hauteur < 150 : Nain ; 150 <= Hauteur < 165 : Taille moyenne ; Hauteur >= 165 : Grand).

e) Exercice 2.5

Écrire un programme en C qui permet de trouver le plus grand de trois nombres.

f) Exercice 2.6

Écrire un programme C qui permet de résoudre une équation quadratique (équation du second degré ax2 + bx + c = 0).

g) Exercice 2.7

Écrire un programme en C pour lire la température en centigrade et afficher un message approprié selon l'état de la température ci-dessous :

- Temp < 0 : Temps glacial
- Temp 0-10 : Temps très froid
- Temp 10-20 : Temps froid
- Temp 20-30 : Température normale
- Temp 30-40 : Il fait Chaud
- Temp >=40 : Il fait très chaud

h) Exercice 2.8

Écrire un programme C qui permet de vérifier si un triangle est équilatéral, isocèle ou scalène. Un triangle scalène est un triangle qui a 3 côtés de longueurs différentes. Un triangle isocèle est un triangle qui a 2 côtés de même longueur. Un triangle équilatéral est un triangle qui a 3 côtés de même longueur.

i) Exercice 2.9

Écrire un programme en C qui permet de lire un nombre entier allant de 1 à 7 entré au clavier par l'utilisateur et afficher le jour correspondant (dimanche, lundi, …, vendredi, samedi). Utiliser l'instruction switch ... case.

j) Exercice 2.10

Écrire un programme en C qui permet de lire un numéro de mois en nombre entier et afficher le nombre de jours pour ce mois (utiliser switch ... case).

k) Exercice 2.11

Écrire un programme en C qui simule le fonctionnement d'une calculatrice. Le programme permettant la saisie de deux entiers et une opération (+, –, /, *) et affichant le résultat. Attention à la division par zéro.

l) Exercice 2.12

Écrire un programme C qui permet de vérifier si un caractère entré au clavier par l'utilisateur est un alphabet, un chiffre ou un caractère spécial.

2. Corrigés d'exercices

a) Corrigé d'exercice 2.1

```
#include <stdio.h>
void main()
{
    int int1, int2;

    printf("Entrez les valeurs de Nombre1 et Nombre2 : ");
    scanf("%d %d", &int1, &int2);
    if (int1 == int2)
        printf("Les nombres 1 et 2 sont égaux \n");
    else
        printf("Les nombres 1 et 2 ne sont pas égaux \n");
}
```

b) Corrigé d'exercice 2.2

```
#include <stdio.h>
void main()
{
    int num1, r;

    printf("Entrez un nombre entier : ");
    scanf("%d", &num1);
    r = num1 % 2;
    if (r == 0)
        printf("%d est un entier pair \n", num1);
    else
        printf("%d est un entier impair \n", num1);
}
```

c) Corrigé d'exercice 2.3

```
#include <stdio.h>
void main()
{
    int nb;

    printf("Entrez un nombre :");
    scanf("%d", & nb);
    if (nb >= 0)
        printf("%d est un nombre positif \n", nb);
    else
        printf("%d est un nombre négatif \n", nb);
}
```

d) Corrigé d'exercice 2.4

```
#include <stdio.h>
void main()
{
    float Taille;

    printf("Saisir la taille de la personne (en centimètres):");
    scanf("%f", &Taille);
    if (Taille < 150.0)
       printf("La personne est Nain. \n");
       else if (Taille < 165.0)
             printf("La personne est de taille moyenne. \n");
            else if (Taille <= 195.0)
                  printf("La personne est plus grande. \n");
                 else
                  printf("Taille anormale.\n");
}
```

e) Corrigé d'exercice 2.5

```
#include <stdio.h>
void main()
{
    int num1, num2, num3;

    printf("Entrez les valeurs de trois nombres : ");
    scanf("%d %d %d", &num1, &num2, &num3);
    printf("1er Nombre = %d,\t2e Nombre = %d,\t3e Nombre = %d\n", num1, num2,
num3);
    if (num1 > num2)
    {
        if (num1 > num3)
        {
            printf("Le 1er nombre est le plus grand parmi trois. \n");
        }
        else
        {
            printf("Le 3ème nombre est le plus grand parmi les trois. \n");
        }
    }
    else if (num2 > num3)
```

```
        printf("Le 2ème nombre est le plus grand parmi trois \n");
    else
        printf("Le 3ème nombre est le plus grand parmi trois \n");
}
```

f) Corrigé d'exercice 2.6

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

void main()
{
   int a,b,c,d; /* a est différent de 0*/
   float x1,x2;

   printf("Entrez les valeurs de a,b et c : ");
   scanf("%d%d%d",&a,&b,&c);
   d=b*b-4*a*c;
   if(d==0)
   {
     printf("Les deux racines sont égales.\n");
     x1=-b/(2.0*a);
     x2=x1;
     printf("Première racine Racine1 = %f\n",x1);
     printf("Deuxième racine Racine2 = %f\n",x2);
   }
   else if(d>0)
      {
         printf("Les deux racines sont réelles et différentes. -2\n");
         x1=(-b+sqrt(d))/(2*a);
         x2=(-b-sqrt(d))/(2*a);
         printf("Première racine Racine1 = %f\n",x1);
         printf("Deuxième racine Racine2 = %f\n",x2);
      }
      else
          printf("Les racines sont imaginaires;\n Pas de solution. \n");
}
```

g) Corrigé d'exercice 2.7

```
#include <stdio.h>
void main()
{
     int tmp;

    printf("Entrez la température : ");
    scanf("%d",&tmp);
   if(tmp<0)
             printf("Temps glacial.\n");
   else if(tmp<10)
            printf("Temps très froid.\n");
            else if(tmp<20)
                        printf("Temps froid.\n");
                   else if(tmp<30)
                              printf("Température normale.\n");
                          else if(tmp<40)
                                      printf("C'est chaud.\n");
                                  else
```

```
                                        printf("Il fait très chaud.\n");

}
```

h) Corrigé d'exercice 2.8

```
#include <stdio.h>
int main()
{
    int a, b, c; // sont trois côtés d'un triangle

    printf("Entrer les trois côtés du triangle : ");
    scanf("%d %d %d", &a, &b, &c);  // Lire tous les côtés d'un triangle

    if(a==b && b==c) // vérifier si tous les côtés sont égaux
    {
        printf("C'est un triangle équilatéral.\n");
    }
    else if(a==b || a==c || b==c) // vérifier si deux côtés sont égaux
    {
        printf("C'est un triangle isocèle.\n");
    }
    else //vérifier si les côtés sont différents
    {
        printf("C'est un triangle scalène.\n");
    }

    return 0;
}
```

i) Corrigé d'exercice 2.9

```
#include <stdio.h>
void main()
{
   int noj;
   printf("Entrez le numéro de jour : ");
   scanf("%d",& noj);
   switch(noj)
   {
       case 1:
              printf("Dimanche \n");
              break;
       case 2:
              printf("Lundi \n");
              break;
       case 3:
              printf("Mardi \n");
              break;
       case 4:
              printf("Mercredi \n");
              break;
       case 5:
              printf("Jeudi \n");
              break;
       case 6:
              printf("Vendredi \n");
              break;
       case 7:
```

```
            printf("Samedi  \n");
            break;
      default:
            printf("Numéro de jour invalide. \n Veuillez essayer à
nouveau....\n");

      }
}
```

j) Corrigé d'exercice 2.10

```
#include <stdio.h>
void main()
{
   int nomois;
   printf("Entrez le numéro de mois : ");
   scanf("%d",& nomois);
   switch(nomois)
   {
         case 1:
         case 3:
         case 5:
         case 7:
         case 8:
         case 10:
         case 12:
               printf("Le mois a 31 jours. \n");
               break;
         case 2:
               printf("Le deuxième mois est un février et compte 28 ou 29 jours.
\n");
               break;
         case 4:
         case 6:
         case 9:
         case 11:
               printf("Le mois a 30 jours. \n");
               break;
default:
               printf("Numéro de mois invalide.\nVeuillez essayer à
nouveau....\n");

      }
}
```

k) Corrigé d'exercice 2.11

```
#include<stdio.h>
int main(){
   int a,b;
   char operation;
   printf("Donner l'operande 1 : ");
   scanf("%d", &a);

   printf("Donner l'operation : ");
   scanf(" %c",&operation);

   printf("Donner l'operande 2 : ");
```

```
   scanf("%d",&b);

   if(operation=='+') printf("%d+%d=%d",a,b,a+b);
   else if (operation=='-') printf("%d-%d=%d",a,b,a-b);
   else if (operation=='*') printf("%d*%d=%d",a,b,a*b);
   if(operation=='/') {
         if(b==0) printf("Division par zero");
         else printf("%d/%d=%f",a,b,(float)a/b);
   }
}
```

l) Corrigé d'exercice 2.12

```
#include <stdio.h>
int main()
{
    char ch;

    printf("Veuillez entrer n'importe quel caractère :  ");
    scanf("%c", &ch);

    if( (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') )
    {
        printf("\n %c est un Alphabet ", ch);
    }
    else if (ch >= '0' && ch <= '9')
    {
        printf("\n %c est un chiffre", ch);
    }
    else
        printf("\n %c est un caractère spécial ", ch);

    return 0;
}
```

Chapitre 4 : Les boucles

A. Rappel du cours

1. Introduction

La notion d'itération (ou de boucles) est utilisée souvent pour faire plusieurs fois le même traitement sur un même objet ou plusieurs objets de même nature. L'exécution du traitement répété est contrôlée à l'aide d'une **variable ou expression** de contrôle d'où le nom de structures de contrôles.

Il existe trois façons pour exprimer l'itération (3 types de boucles) :

- la boucle **while (Tant que ... faire)**
- la boucle **do...while (Répéter...jusqu'à...)**
- la boucle **for (Pour)**

La section suivante explique plus en détail les différents notions et types de boucles en algorithmique et en langage C :

2. La boucle Tant que (while)

La boucle Tant que permet de répéter des opérations tant qu'une condition est vérifiée. Dès que la condition devient fausse, la boucle est rompue (on sort de la boucle).

Syntaxe :

Dans sa forme la plus simple (structure TANTQUE) telle que présentée dans le tableau ci-dessous, une structure répétitive sous forme pseudo-code est composée des mots réservés **TANTQUE** et **FAIRE**, d'une condition et d'une séquence d'instructions à exécuter tant que la condition est vraie.

Algorithme	**Langage C**
Tant que (condition) **faire** bloc ;	**while** (condition) bloc ;

Si condition est vraie alors ceux sont les instructions de bloc qui seront exécutées sinon, le programme continue en séquence c.à.d. à la suite, après l'instruction Tant que (while). Quand Condition devient fausse, on sort de la boucle.

Remarque :

- Si bloc contient une seule instruction, les délimiteurs 'Début...Fin sont omis et en C, les accolades '{} 'sont omises.
- L'utilisation d'une variable compteur sur laquelle est basée la condition d'arrêt avec l'instruction Tant que (while), nécessite l'initialisation et l'incrémentation/décrémentation de celle-ci.
- La condition de la boucle peut être composée de plusieurs conditions liées avec les opérateurs && (ET) et || (OU), comme suit :
  - if(Condition1 && Condition2)
  - if(Condition1 || Condition2)

Une instruction Tant que (while) peut contenir à son tour une ou plusieurs instructions Tant que (while)

mais attention la boucle interne doit toujours se terminer en premier.

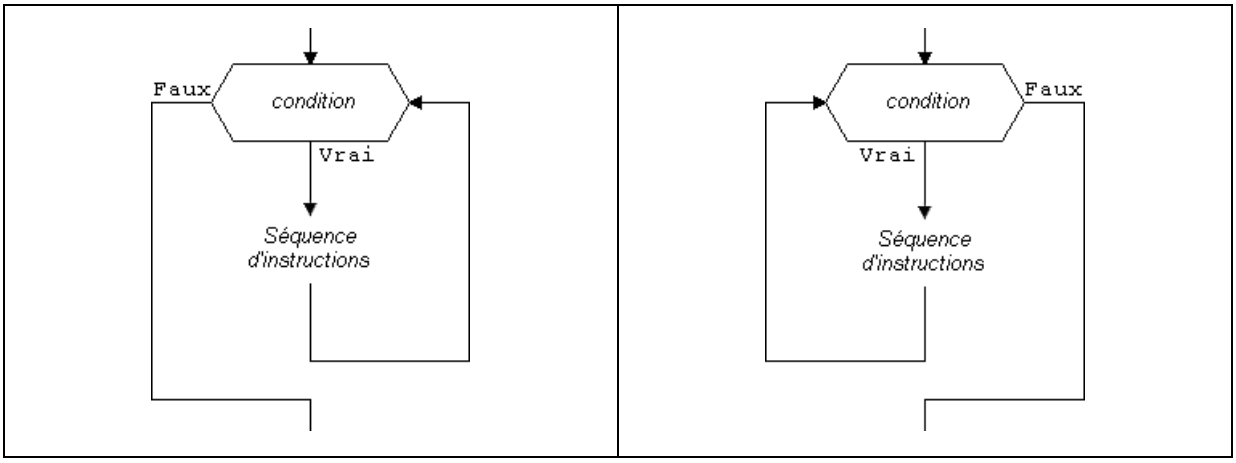


Figure 6. La boucle « Tant que »

Dans l'organigramme ci-dessus, la structure TANTQUE comprend une condition dans un hexagone suivie de la séquence d'instructions sur la branche étiquetée Vrai. L'orientation de la branche Faux peut être à gauche ou à droite de la condition. Notez que la branche Vrai retourne à la condition, indiquant ainsi que le flux d'exécution retourne évaluer la condition une fois la séquence d'instructions exécutée.

Exemple :

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    while (i < 5)
    {
        printf("La variable i vaut %d\n", i);
        i++;
    }
    return 0;
}
```

Le fonctionnement est simple à comprendre : Au départ, la variable i vaut zéro. Étant donné que zéro est bien inférieur à cinq, la condition est vraie, le corps de la boucle est donc exécuté. La valeur de i est affichée. i est augmentée d'une unité et vaut désormais un. La condition de la boucle est de nouveau vérifiée. Ces étapes vont ainsi se répéter pour les valeurs un, deux, trois et quatre. Quand la variable i vaudra cinq, la condition sera fausse, et l'instruction while sera alors passée.

3. La boucle Répéter...jusqu'à... ou Faire... Tant que (do... while)

Le bloc d'instructions s'exécute, puis la condition est vérifiée. Si elle est vraie, on effectue de nouveau le bloc d'instructions, et ainsi de suite. La boucle est rompue (on sort de la boucle) quand la condition devient fausse.

Syntaxe :

Algorithme	**Langage C**
Répéter bloc	**do** bloc

Jusqu'à (condition) ; *ou* **Faire** bloc **Tant que** (condition) ;	**while** (condition) ;

Remarque :

- Avec l'instruction **do ..while**, la condition de la boucle est vérifiée à la fin. La différence avec l'instruction **while** est que celle-ci s'exécute au moins une fois avant de vérifier la condition. Ce type de boucle est moins utilisé.
- La boucle **while** peut très bien ne jamais être exécutée si la condition est fausse dès le départ. Pour la boucle do... while, c'est différent : cette boucle s'exécutera toujours au moins une fois.

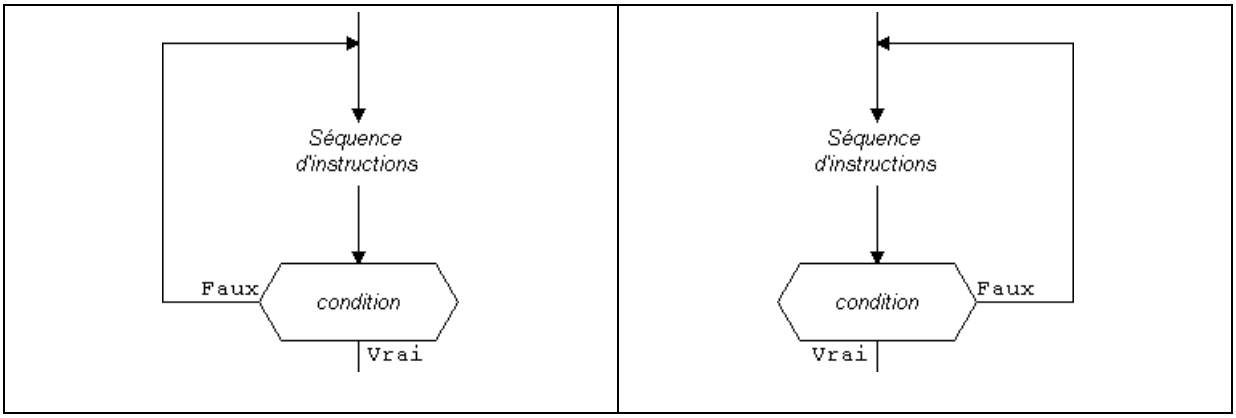


Figure 7. La boucles «Répéter...jusqu'à»

Exemple :

Voici le même code que celui présenté avec l'instruction while :

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    do
    {
        printf("La variable i vaut %d\n", i);
        ++i;
    } while (i < 5);
    return 0;
}
```

4. La boucle Pour (for)

La boucle **pour**, en algorithmique et **for** en programme C, est une structure de contrôle qui permet de **répéter l'exécution d'une séquence d'instructions**. Dans cette forme de boucle, une variable prend des valeurs successives sur un intervalle. Cette forme est souvent utilisée pour exploiter les données d'une collection indexée.

La boucle **pour** ou for a **deux parties** :

- Un **entête** qui spécifie la manière de faire l'itération.

- Un **corps** qui est exécuté à chaque itération.

Syntaxe :

Algorithme	**Langage C**
Pour (Initialisation; Expression ou Condition ; Pas) **Début** bloc ; **Fin**;	**For** (Initialisation; Expression ou Condition ; Pas) **{** bloc ; **}**

L'entête de cette boucle est devisé en trois parties :

- **Initialisation** : contient une ou plusieurs instructions d'initialisations à exécuter une seule fois avant d'accéder à la boucle.
- **Expression** : c'est une expression logique (booléenne) à évaluer, on quitte la boucle si cette expression n'est pas vérifiée (prend la valeur faux).
- **Pas** : est exécuté après chaque itération (exécution du bloc 3).

Instructions : séries d'instructions à exécuter itérativement.

Exemple :

Le fonctionnement de cette boucle est plus simple à appréhender à l'aide d'un exemple.

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    do
    {
        printf("La variable i vaut %d\n", i);
        ++i;
    } while (i < 5);
    return 0;
}
```

Remarque :

- Les **boucles** sont des structures qui nous permettent de répéter une série d'instructions plusieurs fois.
- Il existe plusieurs types de boucles : **while**, **do... while** et **for**. Certaines sont plus adaptées que d'autres selon les cas.
- **for** est probablement celle qu'on utilise le plus. On y fait très souvent des incrémentations ou des décrémentations de variables.

5. Les boucles imbriquées

Les boucles peuvent être **imbriquées** les unes dans les autres. Une boucle Tant que peut contenir une autre boucle Tant que, une autre boucle Répéter, ou une autre boucle Pour, et vice versa. De plus, une boucle peut contenir une autre boucle, qui elle-même peut contenir une autre boucle, et ainsi de suite.

L'exemple suivant illustre le concept :

```
for(compteur = 2;compteur <= 9;compteur++) {
   // Ici, le code présent dans cette boucle sera exécuté 10 fois.
        for(i = 0; i < 100; i++) {
                // Ici, le code présent dans cette boucle sera exécuté 100 fois.
Comme sa boucle maître sera répétée 10 fois.
```

```
                // On peut calculer que ce code sera exécutée 10*100=1000 fois
sur toute la durée de la boucle maître.
        }
}
```

B. Travaux Pratiques N° 3

Cette rubrique présente une série d'exercices avec leurs corrigés sur **les structures itératives, les boucles et notion de compteur, les boucles avec instructions conditionnelles, et les boucles imbriquées**.

1. Exercices

a) Exercice 3.1

Ecrire un programme C qui calcul la somme de tous les nombres naturels entre 1 et n en utilisant la boucle for. Réécrire le programme en utilisant la boucle do…while et la boucle while.

b) Exercice 3.2

Ecrire un programme C qui calcul la puissance en utilisant uniquement la multiplication (utiliser la boucle while et la boucle for).

c) Exercice 3.3

Ecrire un programme C qui calcul la factorielle d'un nombre entier en utilisant la boucle while et la boucle for. Par exemple, la factorielle de 7 : 7!=7*6*5*4*3*2*1=5040.

d) Exercice 3.4

Ecrire un programme C pour afficher la table de multiplication d'un entier donné.

e) Exercice 3.5

Ecrire un programme C qui permet de vérifier si un nombre est un nombre premier ou non.

f) Exercice 3.6

Ecrire un programme C qui permet de trouver le PGCD (Plus Grand Commun Diviseur) de deux nombres entiers. Le PGCD est le plus grand nombre qui divise exactement les deux nombres entiers (sans reste). Par exemple, le PGCD de 54 et 24 est de 6. ***Exercice 3.7***

Ecrire un programme C qui permet d'afficher la suite de Fibonacci jusqu'à n termes, sachant que :

$$u(n) = \begin{cases} 0 & si\ n = 0 \\ 1 & si\ n = 1 \\ u(n-2) + u(n-1), si\ n > 1 \end{cases}$$

Par exemple : le nombre de termes : 10. Suite de Fibonacci : 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.

h) Exercice 3.8

Pour chaque suite, écrire un programme C qui permet d'afficher la somme :

- 1 + 1/2 + 1/3 + 1/4 + 1/5 ... + 1/n
- 1/1! + 1/2! + 1/3! + … + 1/n!
- 1/1! + 2/2! + 3/3! + …. + n/n!
- 1 + x + x^2/2! + x^3/3! +.... x^n/n!

2. Corrigés d'exercices

a) Corrigés d'exercice 3.1

```
#include <stdio.h>
int main()
{
    int i, n, somme=0;
    /* Entrée de la limite supérieure de l'utilisateur */
    printf("Entrez un nombre entier positif : ");
    scanf("%d", &n);
    /* Trouver la somme de tous les nombres */
    for(i=1; i<=n; i++)
    {
        somme = somme + i;
    }
    printf("Somme des %d premiers nombres naturels = %d", n, somme);
    return 0;
}
```

```
#include <stdio.h>
int main() {
    int n, i, somme = 0;
    printf("Entrez un nombre entier positif : ");
    scanf("%d", &n);
    i = 1;
    while (i <= n) {
        somme += i;
        ++i;
    }
    printf("Somme des %d premiers nombres naturels = %d", n, somme);
    return 0;
}
```

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int n, i, somme = 0;
    printf("Entrez un nombre entier positif : ");
    scanf("%d", &n);
    i = 0;
    do
    {
        somme = somme + i; // stocker la somme des nombres naturels
        i++;               // incrémenter de 1
    } while (i <= n);
    printf("Somme des %d premiers nombres naturels = %d", n, somme);
}
```

b) Corrigés d'exercice 3.2

```
#include<stdio.h>
int main()
{
    int base, exposant, puissance, i;
    //Lecture de la base et de l'exposant
    printf("Entrez la base :  ");
    scanf("%d",&base);
    printf("Entrez l'exposant :  ");
    scanf("%d",&exposant);

    puissance = 1;
    i = 1;
    //caculatif de la puissance d'un nombre donné
    while(i <= exposant)
    {
        puissance = puissance * base;
        i++;
    }
    printf("La puissance : %d", puissance);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    int base, exposant, i, puissance;

    printf("Entrez la base : ");
    scanf("%d",&base);
    printf("Entrez l'exposant : ");
    scanf("%d", &exposant);

    puissance = 1;
    //caculatif de la puissance d'un nombre donné en utilisant la boucle for
    for(i=1; i<=exposant; i++)
        puissance = puissance * base;
    printf("La puissance : %d", puissance);
    return 0;
}
```

c) Corrigés d'exercice 3.3

```
#include <stdio.h>
int main()
{
    int n,i;
```

```
    long long fact=1;
    printf("Entrez un nombre quelconque pour calculer la factorielle : ");
    scanf("%d",&n);
    i = 1;
    //Faire tourner la boucle de 1 à un nombre entré par l'utilisateur
    while(i <= n)
    {
        fact = fact * i;
        i++;
    }
     printf("Factorielle de %d = %lld", n, fact);
 return 0;
}
```

```
#include <stdio.h>
int main()
{
    int i, n;
    long long fact=1;
    printf("Entrez un nombre quelconque pour calculer la factorielle : ");
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    {
        fact = fact * i;
    }
    printf("Factorielle de %d = %lld", n, fact);
    return 0;
}
```

d) Corrigés d'exercice 3.4

```
#include <stdio.h>
int main()
{
    int i, n;
    printf("Entrez le nombre pour afficher la table de multiplication : ");
    scanf("%d", &n);
    for(i=1; i<=10; i++)
    {
        //Afficher la table de multiplication du nombre entré par l'utilisateur
        printf("%d x %d = %d\n", n, i, (n*i));
    }
    return 0;
}
```

e) Corrigés d'exercice 3.5

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

int main()
{
    int n, i, f;

    printf("Entrez un nombre quelconque : ");
    scanf("%d",&n);
    f=0;
    for(i=2; i <= sqrt(n); i++)
    {
        if(n%i == 0)
        {
            f=1;
            break;
        }
    }
    if(f==0)
        printf("%d est un nombre premier", n);
    else
        printf("%d n'est pas un nombre premier", n);
    return 0;
}
```

f) Corrigés d'exercice 3.6

```
#include <stdio.h>
int main()
{
    int a, b, pgcd, i;
    printf("Entrez deux nombres entiers : ");
    scanf("%d %d", &a, &b);
    for(i=1; i <= a && i <= b; i++)
    {
        if(a%i==0 && b%i==0)
            pgcd = i;
    }
    printf("PGCD de %d et %d = %d", a, b, pgcd);
    return 0;
}
```

g) Corrigés d'exercice 3.7

```
#include <stdio.h>
int main() {
  int i, n;
  // initialiser le premier et le second terme
  int t1 = 0, t2 = 1;
  // initialiser le terme suivant (3éme terme)
  int termeSuivant = t1 + t2;
```

```
  // obtenir le nombre de termes de l'utilisateur
  printf("Entrez le nombre de termes : ");
  scanf("%d", &n);
  // afficher les deux premiers termes t1 et t2
  printf("Suite de Fibonacci : %d, %d, ", t1, t2);
  // Afficher du 3éme au n-éme terme
  for (i = 3; i <= n; i++) {
    printf("%d, ", termeSuivant);
    t1 = t2;
    t2 = termeSuivant;
    termeSuivant = t1 + t2;
  }
  return 0;
}
```

h) Corrigés d'exercice 3.8

```
// 1+1/2+1/3+1/4+...+1/n
#include<stdio.h>
int main ()
{
    int n,i;
    float s;
    printf("Entrez le nombre de termes : ");
    scanf("%d", &n);
    s=0;

    for(i=1;i<=n;i++)
       {
         s = s + (float)1/i;
        }
    printf("La somme de %d termes de la suite est %f\n", n, s);
}
```

```
// 1+1/2!+1/3!+1/4!+…..+1/n!
#include<stdio.h>
int main()
{
int n,i,j,fact=1;
float s=0;
printf("Entrez le nombre de termes : ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
    fact=1;
    for(j=i;j>0;j--)
    {
        fact=fact * j;
    }
```

```
    s=s+(1.0/fact);
}
printf("La somme de %d termes de la suite est %f\n", n, s);
return 0;
}
```

```
// 1/1! + 2/2! + 3/3! + …. + n/n!
#include <stdio.h>
int main()
{
    int i = 1, j, n;
    float s = 0.0, fact;
    printf("Entrez le nombre de termes : ");
    scanf("%d", &n);
    while(i <= n)
    {
        fact = 1;
        for(j = 1; j <= i; j++)
        {
            fact = fact * j;
        }
        s = s + (i / fact);
        i++;
    }
    printf("La somme de %d termes de la suite est %f\n", n, s);
    return 0;
}
```

```
// 1 + x + x^2/2! + x^3/3! +.... x^n/n!
#include <stdio.h>
void main()
{
      float x,s,p;
      int i,n;
      printf("Entrez la valeur de x : ");
      scanf("%f",&x);
      printf("Entrez le nombre de termes : ");
      scanf("%d",&n);
      s = 1; p = 1;
      for (i=1;i<n;i++)
      {
        p = p*x/(float)i;
        s =s+ p;
      }
      printf("La somme de %d termes de la suite est %f\n", n, s);
}
```

Chapitre 5 : Les tableaux et les chaînes de caractères

A. Rappel du cours

1. Introduction

Les variables utilisées, jusqu'à présent, étaient toutes de type simple prédéfini (entier, réel, caractère ou logique). Dans cette partie, nous allons voir une autre manière de représenter les données, c'est le type structuré Tableau. Une variable de type tableau n'est plus une variable simple, mais une structure de données, regroupant des informations de même type. Nous distinguons deux types de tableaux: les **tableaux à une dimension** (Vecteurs) et les **tableaux à plusieurs dimensions** (Matrices). En C, une **chaîne de caractères** est équivalente à un tableau de caractères. Ce chapitre introduit ces deux notions (chaînes et tableaux).

2. Le type tableau

Les **tableaux** sont une sorte de structure de données qui permet de stocker une collection séquentielle de taille fixe d'éléments du même type. Au lieu de déclarer des variables individuelles, telles que nombre0, nombre1, ..., et nombre99, on peut déclarer une variable de tableau telle que nombres et utilisez nombres[0], nombres[1], et ..., nombres[99] pour représenter les variables individuelles. On accède à un élément spécifique d'un tableau par un index.

Déclaration des tableaux

Pour déclarer un tableau en C, il faut spécifie le type des éléments et le nombre d'éléments requis par un tableau comme suit :

type nomTableau **[**tailleTableau**]**;

Exemple :

double balance[10];

Ici, balance est un tableau variable suffisant pour contenir jusqu'à 10 nombres doubles.

Initialisation des tableaux

On peut initialiser un tableau en C soit un par un, soit en utilisant une seule instruction comme suit :

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

Le nombre de valeurs entre accolades { } ne peut être supérieur au nombre d'éléments que nous déclarons pour le tableau entre crochets [ ]. Si la taille du tableau n'est pas précisée, un tableau juste assez grand pour contenir l'initialisation est créé. Par conséquent, si on écrit :

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

Le même tableau sera créé que dans l'exemple précédent. Voici un exemple pour assigner un seul élément du tableau :

balance[4] = 50.0;

L'instruction ci-dessus attribue au 5ème élément du tableau une valeur de 50.0. Tous les tableaux ont 0 comme indice de leur premier élément, également appelé indice de base, et le dernier indice d'un

tableau sera la taille totale du tableau moins 1. Le tableau ci-dessous est la représentation graphique du tableau dont nous avons parlé plus haut.

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

Accès aux éléments d'un tableau

On accède à un élément en indexant le nom du tableau. Pour ce faire, on place l'indice de l'élément entre crochets après le nom du tableau. Par exemple :

double salaire = balance[9];

L'instruction ci-dessus prendra le 10ème élément du tableau et assignera la valeur à la variable salaire.

L'exemple suivant montre comment utiliser les trois concepts mentionnés ci-dessus, à savoir la déclaration, l'affectation et l'accès aux tableaux.

```
#include <stdio.h>
int main () {
   int n[10] ; /* n est un tableau de 10 entiers */
   int i,j ;
    /* initialiser les éléments du tableau n à 0 */
   for (i = 0 ; i < 10 ; i++) {
      n[i] = i + 100; /* fixe l'élément à l'emplacement i à i + 100 */
   }
    /* affiche la valeur de chaque élément du tableau */
   for (j = 0 ; j < 10 ; j++) {
      printf("Element[%d] = %d\n", j, n[j]);
   }
   return 0;
}
```

3. Les tableaux multidimensionnels

La section précédente traite les tableaux, également connus sous le nom de tableaux à une dimension. Ils sont excellents et très utilisés dans la programmation en C. Cependant, si on souhaite stocker des données sous forme de tableau, comme un tableau avec des lignes et des colonnes, il faut se familiariser avec les **tableaux multidimensionnels**. Un tableau multidimensionnel est en fait un tableau de tableaux. Voici la forme générale d'une déclaration de tableau multidimensionnel :

type nomTableau [taille1] [taille2]... [tailleN] ;

Par exemple, la déclaration suivante crée un tableau d'entiers à trois dimensions :

int troisdim[5][10][4];

Les tableaux peuvent avoir un nombre quelconque de dimensions. Dans cette section, les tableaux les plus courants sont présentés, à savoir les tableaux à deux dimensions (2D).

Tableaux à deux dimensions

Un tableau bidimensionnel est également connu sous le nom de **matrice** (un tableau de lignes et de colonnes). Pour créer un tableau d'entiers en 2D, consultez l'exemple suivant :

int matrice[2][3] = { {1, 4, 2}, {3, 6, 8} };

La première dimension représente le nombre de lignes, tandis que la deuxième dimension représente le nombre de colonnes. Les valeurs sont placées dans l'ordre des lignes, et peuvent être visualisées comme suit :

	Colonne 0	Colonne 1	Colonne 2
Ligne 0	1	4	2
Ligne 1	3	6	8

Accès aux éléments d'un tableau en 2D

Pour accéder à un élément d'un tableau bidimensionnel, il faut spécifier le numéro d'index de la ligne et de la colonne. L'instruction suivante accède à la valeur de l'élément de la première ligne (0) et de la troisième colonne (2) du tableau matriciel.

```
int matrice[2][3] = { {1, 4, 2}, {3, 6, 8} };
printf("%d", matrice[0][2]);  // Sortie : 2
```

Changer les éléments d'un tableau 2D

Pour modifier la valeur d'un élément, il faut se référer au numéro d'index de l'élément dans chacune des dimensions. L'exemple suivant modifiera la valeur de l'élément de la première ligne (0) et de la première colonne (0) :

```
int matrice[2][3] = { {1, 4, 2}, {3, 6, 8} };
matrice[0][0] = 9;
printf("%d", matrice[0][0]);  // Il produit maintenant 9 au lieu de 1.
```

Boucle dans un tableau 2D

Pour effectuer une boucle dans un tableau multidimensionnel, il faut une boucle pour chacune des dimensions du tableau. L'exemple suivant affiche tous les éléments du tableau de la matrice :

```
int matrice[2][3] = { {1, 4, 2}, {3, 6, 8} };
int i, j;
for (i = 0; i < 2; i++) {
  for (j = 0; j < 3; j++) {
    printf("%d\n", matrce[i][j]);
  }
}
```

4. Les chaînes de caractères

Les **chaînes de caractères** sont en fait des tableaux unidimensionnels de caractères terminés par un caractère **nul '\0'**. Ainsi, une chaîne terminée par un caractère nul contient les caractères qui composent la chaîne, suivis d'un caractère nul.

Déclaration et Initialisation des chaînes de caractères

La déclaration et l'initialisation suivantes créent une chaîne composée du mot "Hello". Pour maintenir le caractère nul à la fin du tableau, la taille du tableau de caractères contenant la chaîne est supérieure d'une unité au nombre de caractères du mot "Hello".

char salutation[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

Si nous suivons la règle de l'initialisation des tableaux, nous pouvons écrire la déclaration ci-dessus comme suit :

char salutation[] = "Hello" ;

Voici la présentation en mémoire de la chaîne définie ci-dessus en C :

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Adresse	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

En fait, on ne place pas le caractère nul à la fin d'une constante de chaîne. Le compilateur C place automatiquement le caractère '\0' à la fin de la chaîne lorsqu'il initialise le tableau.

Essayons d'afficher la chaîne de caractères mentionnée ci-dessus :

```
#include <stdio.h>
int main () {
   char salutation[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Message de salutation : %s\n", salutation);
   return 0;
}
```

Lorsque le code ci-dessus est compilé et exécuté, il produit le résultat suivant :

Message de salutation : Hello

Le C supporte un large éventail de fonctions qui manipulent les chaînes de caractères à terminaison nulle :

- strcpy(s1, s2) : Copie la chaîne s2 dans la chaîne s1.
- strcat(s1, s2) : Concatène la chaîne s2 à la fin de la chaîne s1.
- strlen(s1) : Renvoie la longueur de la chaîne s1.
- strcmp(s1, s2) : Renvoie 0 si s1 et s2 sont identiques ; inférieur à 0 si s1<s2 ; supérieur à 0 si s1>s2.
- strchr(s1, ch) : Renvoie un pointeur sur la première occurrence du caractère ch dans la chaîne s1.
- strstr(s1, s2) : Renvoie un pointeur sur la première occurrence du caractère s2 dans la chaîne s1.

B. Travaux Pratiques N° 4

Cette rubrique présente une série d'exercices avec leurs corrigés sur la **manipulation des tableaux à une et à deux dimensions (matrices) et les chaînes de caractères.**

1. Exercices

a) Exercice 4.1

Ecrire un programme C qui permet de chercher un caractère dans une chaîne et compter toutes ces occurrences.

b) Exercice 4.2

Ecrire un programme C qui permet de compter le nombre total de voyelles ou de consonnes dans une chaîne.

c) Exercice 4.3

Ecrire un programme C pour vérifier si une chaîne est palindrome ou non. Par Exemple: coloc, été, gag, radar, pop, elle ou kayak sont toutes des chaînes de caractères palindromes.

d) Exercice 4.4

Ecrire un programme C pour lire les éléments d'un tableau et calculer la somme et la moyenne des éléments du tableau.

e) Exercice 4.5

Ecrire un programme C pour trouver le plus petit et le plus grand nombre avec leurs positions dans un tableau.

f) Exercice 4.6

Ecrire un programme C pour entrer des éléments dans un tableau et trier les éléments du tableau dans l'ordre croissant (ou décroissant).

g) Exercice 4.7

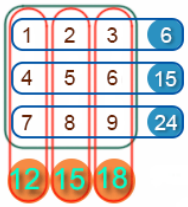
Ecrire un programme C pour entrer des éléments dans un tableau et placer les éléments pairs et impairs dans des tableaux séparés.

h) Exercice 4.8

Ecrire un programme C qui permet de faire l'addition de deux matrices carrées de même taille.

i) Exercice 4.9

Ecrire un programme C qui permet de lire les éléments d'une matrice et calculer la somme des éléments de chaque ligne et de chaque colonne de la matrice. Par exemple :



j) Exercice 4.10

Ecrire un programme C pour la multiplication de deux matrices. La multiplication de deux matrices est définie comme suite :

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

k) Exercice 4.11

Ecrire un programme en C pour trouver la transposée d'une matrice donnée. La transposition d'une matrice A est définie comme la conversion de toutes les lignes en colonnes et toutes les colonnes en lignes.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^{T} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

2. Corrigés d'exercices

a) Corrigés d'exercice 4.1

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char chaine[100], c;
    int i=0, nbocc=0;
    printf("Entrer une chaîne de caractères : \n");
    gets(chaine);
    printf("Entrez le caractère à rechercher : ");
    c=getchar();

    while(chaine[i] != '\0') // '\0' est le dernier élément (caractère nul)
    {
        if(chaine[i]==c) {
             nbocc++;
             }
             i++;
    }
    printf("Le caractère '%c' apparaît %d fois \n", c, nbocc);
}
```

b) Corrigés d'exercice 4.2

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main ()
{
    char chaine[100];
    int i, l, v, c;
    printf("Entrer une chaîne de caractères : \n");
    gets(chaine);

    v = 0;
    c = 0;
    l = strlen(chaine);

    for(i=0; i<l; i++)
    {
        if(chaine[i] =='a' || chaine[i]=='e' || chaine[i]=='i' || chaine[i]=='o'
|| chaine[i]=='u'
             ||   chaine[i]=='A'   ||   chaine[i]=='E'   ||   chaine[i]=='I'   ||
chaine[i]=='O' || chaine[i]=='U')
        {
            v++;
```

```
        }
        else   if((chaine[i]>='a'   &&   chaine[i]<='z')   ||   (chaine[i]>='A'   &&
chaine[i]<='Z'))
        {
            c++;
        }
    }
    printf("\nLe nombre total de voyelles dans la chaine est : %d\n", v);
    printf("Le nombre total de consonnes dans la chaine est : %d\n\n", c);
}
```

c) Corrigés d'exercice 4.3

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

int main(){
    char chaine[100];
    int i, n, palindrome = 0;
    printf("Entrez une chaine de caractère: ");
    gets(chaine);
    n = strlen(chaine);
    for(i=0;i < n ;i++){
        if(chaine[i] != chaine[n-i-1]){
            palindrome = 1;
            break;
             }
   }
    if (palindrome) {
        printf("%s n'est pas un palindrome", chaine);
    }
    else {
        printf("%s est un palindrome", chaine);
    }
    return 0;
}
```

d) Corrigés d'exercice 4.4

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i, n, t[100], s=0;
    float m;

    /* Taille d'entrée du tableau */
    printf("Donnez la taille du tableau:\n");
```

```
    scanf("%d",&n);

      /* Éléments d'entrée dans le tableau */
    printf("Entrez %d element(s):\n",n);

    for(i=0;i<n;i++){
      scanf("%d",&t[i]);
            s += t[i];
      }
    m = (float)s/n;
      printf("La somme  = %d\n", s);
      printf("La moyenne  = %f", m);
}
```

e) Corrigés d'exercice 4.5

```
#include<stdio.h>
int main()
{
      int t[100], n, i, Min, Min_Position, Max, Max_Position;

      printf("\nVeuillez entrer la taille du tableau  :  ");
      scanf("%d",&n);

      printf("\nVeuillez entrer %d éléments d'un tableau : \n", n);
      for(i=0; i<n; i++)
      {
            scanf("%d",&t[i]);
      }

      Min = t[0];
      Max = t[0];

      for(i=1; i<n; i++)
      {
      if(Min > t[i])
      {
                  Min = t[i];
                  Min_Position = i;
            }
            if(Max < t[i])
      {
                  Max=t[i];
                  Max_Position = i;
      }
      }

      printf("\n Le plus petit élément du tableau = %d", Min);
      printf("\n Position d'index du plus petit élément = %d", Min_Position);
```

```
      printf("\n Le plus grand élément du tableau = %d", Max);
      printf("\n   Position   d'index   de   l'élément   le   plus   grand   =   %d",
Max_Position);
      return 0;
}
```

f) Corrigés d'exercice 4.6

```
#include <stdio.h>
#define taille_max 100    // Taille maximale du tableau
int main()
{
    int t[taille_max];
    int n;
    int i, j, temp;

    /* Taille d'entrée du tableau */
    printf("Entrez la taille du tableau : ");
    scanf("%d", &n);

    /* Éléments d'entrée dans le tableau */
    printf("Entrer des éléments dans le tableau : \n");
    for(i=0; i<n; i++)
    {
        scanf("%d", &t[i]);
    }

    for(i=0; i<n-1; i++)
    {
        /*
         * Place l'élément actuellement sélectionné t[i]
         * à sa place correcte.
         */
        for(j=i+1; j<n; j++)
        {
            /*
             * Swap (échanger) si l'élément de tableau actuellement sélectionné
             * n'est pas à sa position correcte.
             */
            if(t[i] > t[j])
            {
                temp     = t[i];
                t[i] = t[j];
                t[j] = temp;
            }
        }
    }
```

```
    /* Afficher le tableau trié */
    printf("\nÉléments du tableau dans l'ordre croissant : \n");
    for(i=0; i<n; i++)
    {
        printf("%d\t", t[i]);
    }
    return 0;
}
```

g) Corrigés d'exercice 4.7

```
#include <stdio.h>

void main()
 {
    int t1[100], t2[100], t3[100];
    int i,j=0,k=0,n;
       printf("\n\nSeparez les nombres entiers pairs et impairs dans des tableaux
separes :\n");
       printf("-----------------------------------------------------\n");

       printf("Entrez le nombre d'elements à stocker dans le tableau :");
       scanf("%d",&n);
       printf("Entrez %d elements dans le tableau :\n",n);
       for(i=0;i<n;i++)
            {
            printf("element - %d : ",i);
            scanf("%d",&t1[i]);
          }
    for(i=0;i<n;i++)
    {
      if (t1[i]%2 == 0)
      {
         t2[j] = t1[i];
         j++;
      }
      else
      {
         t3[k] = t1[i];
         k++;
      }
    }

    printf("\nLes elements pairs sont : \n");
    for(i=0;i<j;i++)
    {
      printf("%d ",t2[i]);
    }
```

```
    printf("\nLes elements impairs sont :\n");
    for(i=0;i<k;i++)
    {
       printf("%d ", t3[i]);
    }
    printf("\n\n");
 }
```

h) Corrigés d'exercice 4.8

```
#include <stdio.h>
void main()
{
  int a[50][50],b[50][50],c[50][50],i,j,n;
       printf("\n\nAddition de deux Matrices :\n");
       printf("------------------------------\n");
        printf("Entrez la taille de la matrice carree : ");
       scanf("%d", &n);

    /* Valeurs stockées dans la matrice*/
       printf("Éléments d'entrée dans la première matrice :\n");
       for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                  printf("element - [%d],[%d] : ",i,j);
                  scanf("%d",&a[i][j]);
            }
        }

       printf("Éléments d'entrée dans la deuxième matrice :\n");
       for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                  printf("element - [%d],[%d] : ",i,j);
                  scanf("%d",&b[i][j]);
            }
        }
   printf("\nLa première matrice est :\n");
  for(i=0;i<n;i++)
    {
      printf("\n");
      for(j=0;j<n;j++)
           printf("%d\t",a[i][j]);
    }
```

```
  printf("\nLa deuxième matrice est :\n");
  for(i=0;i<n;i++)
    {
      printf("\n");
      for(j=0;j<n;j++)
      printf("%d\t",b[i][j]);
    }
/* calculer la somme de la matrice */
   for(i=0;i<n;i++)
       for(j=0;j<n;j++)
            c[i][j]=a[i][j]+b[i][j];
   printf("\nL'addition de deux matrices est : \n");
   for(i=0;i<n;i++){
       printf("\n");
       for(j=0;j<n;j++)
            printf("%d\t",c[i][j]);
   }
   printf("\n\n");
}
```

i) Corrigés d'exercice 4.9

```
#include <stdio.h>
void main ()
{
    int a[100][100];
    int i, j, m, n, somme = 0;
    printf("Entrez la taille de la matrice  : \n");
    scanf("%d %d", &m, &n);
    printf("Entrez des elements dans la matrice\n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            scanf("%d", &a[i][j]);
        }
    }
      /* Calculer la somme des éléments de chaque ligne de la matrice */
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            somme = somme + a[i][j] ;
        }
        printf("La somme de la ligne %d est = %d\n", i, somme);
        somme = 0;
    }
    /* Calculer la somme des éléments de chaque colonne de la matrice */
```

```
       somme = 0;
    for (j = 0; j < n; ++j)
    {
        for (i = 0; i < m; ++i)
        {
            somme = somme + a[i][j];
        }
        printf("La somme de la colonne %d est = %d\n", j, somme);
        somme = 0;
    }
}
```

j) Corrigés d'exercice 4.10

```
#include <stdio.h>
int main() {
  int i,j,k,al,ac,bl,bc;
  printf("Donner la taille (AL * AC) de la matrice A : ");
  scanf("%d%d",&al,&ac);
  int A[al][ac];
  printf("Entrer les éléments de la matrice A : ");
  for(i=0;i<al;i++){
      for(j=0;j<ac;j++){
            printf("A[%d][%d] = ", i, j);
            scanf("%d", &A[i][j]);
       }
  }
  printf("Donner la taille (BL * BC) de la matrice B : ");
  scanf("%d%d",&bl,&bc);
  int B[bl][bc];
  printf("Entrer les éléments de la matrice B : ");
  for(i=0;i<bl;i++){
      for(j=0;j<bc;j++){
            printf("B[%d][%d] = ", i, j);
            scanf("%d", &B[i][j]);
       }
  }
  if(ac==bl){
      int C[al][bc];
      printf("La matrice C : \n");
      for(i=0;i<al;i++){
            for(j=0;j<bc;j++){
                  C[i][j]=0;
                  for(k=0;k<ac;k++){
                        C[i][j] = C[i][j] + A[i][k]*B[k][j];
                    }
                    printf("%d\t",C[i][j]);
              }
```

```
                printf("\n");
            }
    }
    else
    printf("Impossible de faire la multiplication entre les deux matrices. ");
}
```

k) Corrigés d'exercice 4.11

```
#include <stdio.h>
void main()
{
    static int a[100][100];
    int i, j, m, n;
     printf("Entrez la taille de la matrice  : \n");
    // Saisie d'éléments dans la matrice par l'utilisateur
    scanf("%d %d", &m, &n);
    printf("Entrez des elements dans la matrice : \n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            scanf("%d", &a[i][j]);
        }
    }
    //Affichage de la matrice originale
    printf("La matrice donnée est : \n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            printf(" %d", a[i][j]);
        }
        printf("\n");
    }
    //Affichage de la transposition de matrice
    printf("La transposée de la matrice est : \n");
    for (j = 0; j < n; ++j)
    {
        for (i = 0; i < m; ++i)
        {
            printf(" %d", a[i][j]);
        }
        printf("\n");
    }
}
```

Chapitre 6 : Les types personnalisés

A. Rappel du cours

1. Introduction

Un type simple est un domaine de valeurs, qui peuvent être des booléens, des caractères, des nombres entiers ou des nombres réels. Nous ne sommes pas limités au compilateur fourni par ces types de données et leurs dérivés comme les tableaux. Nous pouvons définir nos propres types de données, dérivés des types de base. Tels que les **énumérations**, les **enregistrements** et **d'autres types**.

2. Enumérations

L'énumération en C est également connue sous le nom de type énuméré. Il s'agit d'un type de données défini par l'utilisateur qui se compose de valeurs entières et qui fournit des noms significatifs à ces valeurs. L'utilisation de l'énumération en C rend le programme facile à comprendre et à maintenir. L'énumération est définie à l'aide du mot-clé **enum**.

Syntaxe :

```
enum nom_de_énumération {
     enumerateur1 ,
     enumerateur2 ,
     ….
     enumerateurN
} ;
```

Les valeurs associées aux différentes constantes symboliques sont, par défaut, définies de la manière suivante : la première constante est associée à la valeur 0 ; les constantes suivantes suivent une progression de 1. On peut aussi changer les valeurs par défaut des constantes entières au moment de la déclaration.

Exemple :

```
enum fruits{mangue, pomme, fraise, papaye} ;
```

La valeur par défaut de la mangue est 0, la pomme est 1, la fraise est 2 et la papaye est 3. Si nous voulons modifier ces valeurs par défaut, nous pouvons procéder comme indiqué ci-dessous :

```
enum fruits {
     mangue=2,
     pomme=1,
     fraise=5,
     papaye=7,
     } ;
```

Nous pouvons déclarer la variable d'un type de données défini par l'utilisateur, tel que enum. Voyons par exemple comment déclarer la variable d'un type enum.

```
#include <stdio.h>
enum joursSemaine {dimanche=1, lundi, mardi, mercredi, jeudi, vendredi, samedi} ;
int main()
{
enum joursSemaine j ; // déclaration de variable de type joursSemaine.
j=lundi ; // attribution de la valeur de lundi à j.
printf("La valeur de j est %d", j);
return 0 ;
}
```

Dans le code ci-dessus, nous créons un type enum nommé jours de la semaine, et il contient le nom de tous les sept jours. Nous avons attribué 1 valeur au dimanche, et tous les autres noms recevront une valeur égale à la valeur précédente plus un.

L'énumération est utilisée lorsque nous voulons que notre variable n'ait qu'un ensemble de valeurs. Par exemple, nous créons une variable de direction. Elle est également utilisée dans une instruction switch case dans laquelle nous passons la variable enum entre parenthèses switch. Il garantit que la valeur du bloc case doit être définie dans une énumération.

3. Enregistrements (Structures)

Une **structure** (ou **enregistrement**) est une structure de données consistant en un nombre fixé de composants, appelés champs. A la différence du tableau, ces composants ne sont pas obligatoirement du même type, et ne sont pas indexés. La définition du type enregistrement précise pour chaque composant un identificateur de champ, dont la portée est limitée à l'enregistrement, et le type de ce champ.

Syntaxe :

```
struct
Nom_Structure {                 // Déclaration de structure
      <type1>  <id_ch1> ;       // Membres de la structure
      <type2>  <id_ch2> ;
      ...
      <typeN> <id_chN> ;
};                              // Fin de la structure par un point-virgule
```

Où <id_ch1>, <id_ch2>, ..., <id_chN> sont les identificateurs des champs et <type1>, <type2>, ..., <typeN> leurs types respectifs.

Une fois qu'on a défini un type structuré, on peut déclarer des variables enregistrements exactement de la même façon que l'on déclare des variables d'un type primitif.

Exemple :

```
struct Etudiant {
    char nom [30];
    int age ;
    float moyenne ;
};
void main () {
    struct Etudiant e1, e2 ;
}
```

Ici les deux variables e1 et e2 de type struct etudiant sont déclarées. Chaque variable contient trois champs : nom de type chaîne, age de type entier et moyenne de type nombre réel.

En C, il est préférable d'utiliser **typedef** pour déclarer un type struct, avant de déclarer des variables. typedef permet de créer un alias de structure. Ainsi l'exemple précédent devient :

```
typedef struct {
    char nom [30];
    int age ;
```

```
    float moyenne ;
} Etudiant;
void main () {
    Etudiant e1, e2 ;
}
```

La manipulation d'une structure se fait au travers de ses champs. Les champs d'une structure sont accessibles à travers leur nom, grâce à l'opérateur '.'.

Exemple :

```
void main ()
{
    Etudiant e1, e2;
    puts("Donnez votre nom : ");
    scanf("%s", &e1.nom);
    puts("Donnez votre age : ");
    scanf("%d", &e1.age);
    puts("Donnez votre moyenne : ");
    scanf("%f", &e1.moyenne);
    e2=e1;
}
```

Comme pour les tableaux, il n'est pas possible de manipuler une structure globalement, sauf pour affecter une structure à un autre de même type (Par exemple : e2=e1 ;). Par exemple, pour afficher une structure il faut afficher tous ses champs un par un.

4. Autres possibilités de définition de type

Une **union** est un type de structure qui peut être utilisé lorsque la quantité de mémoire utilisée est un facteur clé. De la même manière que la structure, l'union peut contenir différents types de types de données. Chaque fois qu'une nouvelle variable est initialisée à partir de l'union, elle écrase la précédente et utilise cet emplacement mémoire. Ceci est particulièrement utile lorsque le type de données transmis par les fonctions est inconnu, l'utilisation d'une union contenant tous les types de données possibles peut remédier à ce problème.

Définition d'une union :

```
union Personne {
char nom ;
int age ;
double taille ;
} ;
```

L'union étant toujours un type de structure, la méthode pour la déclarer est la suivante :

```
union Personne {
char nom ;
int age ;
double taille ;
}nouvellePersonne ;
ou
Personne nouvellePersonne ;
```

Le processus d'initialisation des variables dans une union est identique à celui d'une structure, mais ce sont les résultats qui rendent les unions uniques. L'opérateur point est toujours utilisé afin d'atteindre les types de données à l'intérieur de l'union.

Exemple :

```
Personne nouvellePersonne ;
nouvellePersonne.age = 20 ;
nouvellePersonne.taille = 6,2 ;
```

Une fois que la variable taille est initialisée, la variable age est écrasée et n'existe plus.

Autres types :

- Les pointeurs : Un pointeur est une variable dans un programme désigne un "tiroir" dans la mémoire de l'ordinateur, dans lequel on peut stocker une valeur d'un certain type de données. Un pointeur est aussi un tiroir, une variable, mais pour stocker une référence à un autre tiroir.
- Les listes chaînées, les piles et les files : ce sont des structures de données à une dimension permettant de stocker de manière linéaire une suite finie de données. On les dénomme parfois structures linéaires.
- Les arbres : servent à représenter un ensemble de données structurées hiérarchiquement. Elles représentent une collection finie de nœuds, de même type de données.
- Classe (C++ uniquement) : est un ensemble de données (comme une structure), et un ensemble de fonctions sur ces données.
- Référence (C++ uniquement) : les références permettent de manipuler une variable avec un nom différent de celui qui a été déclaré. type &référence = identificateur; int &ref = i; Ici, i et ref deviennent des noms pour la même variable.

B. Travaux Pratiques N° 5

Cette rubrique présente une série d'exercices avec leurs corrigés sur les **structures, les énumérations et les unions**.

1. Exercices

a) Exercice 5.1

Ecrire un programme C qui définit une structure *point* qui contiendra les deux coordonnées d'un point du plan. Puis lit deux points et affiche la distance entre ces deux derniers. La distance entre deux points (x1,y1) et (x2,y2) est :

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

b) Exercice 5.2

Ecrire un programme C qui définit une structure *etudiant* où un étudiant est représenté par son nom, son prénom et une note. Lit ensuite une liste d'étudiants entrée par l'utilisateur et affiche les noms de tous les étudiants ayant une note supérieure ou égale à *10* sur *20*.

c) Exercice 5.3

Compléter le code C ci-dessous afin de construire un jeu de 32 cartes. Le jeu est représenté par un tableau de 32 éléments distincts, chacun étant du type Carte.

```
#include<stdio.h>
#include<stdlib.h>
enum Couleur {trefle,carreau , pique, coeur};
enum Face {sept , huit , neuf , dix , valet , dame, roi , as};
struct Carte
{
```

```
Couleur couleur;
Face face ;
};
int main()
{
Carte jeu [32] ;
// Code pour construire le jeu
...
}
```

d) Exercice 5.4

Ecrire un programme C qui lit un ensemble de personnes avec leurs âges, dans un tableau de structures, et supprime ensuite toutes celles qui sont âgées de vingt ans et plus.

e) Exercice 5.5

Créer une union qui stocke un tableau de 21 caractères et 6 entiers (6 depuis 21 / 4 == 5, mais 5 * 4 == 20 donc vous avez besoin de 1 de plus pour cet exercice), définir les entiers sur 6 valeurs données, puis afficher le tableau de caractères à la fois sous forme de série de caractères et de chaîne.

2. Corrigés d'exercices

a) Corrigés d'exercice 5.1

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct point{
       float x;
       float y;
       };
int main()
{
    struct point A,B;
    float dist;
    printf("Entrez les coordonnees du point A:\n");
    scanf("%f%f",&A.x,&A.y);
    printf("Entrez les coordonnees du point B:\n");
    scanf("%f%f",&B.x,&B.y);
    dist = sqrt( (A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y) );
    printf("La distance entre les points A et B est: %.2f\n",dist);
    system("pause");
    return 0;
}
```

b) Corrigés d'exercice 5.2

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct etudiant{
       char nom[30];
       char prenom[30];
       int note;
       };
int main()
{
```

```
    struct etudiant e[100];
    int n,i;
    printf("Donner le nombre d'etudiants:\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
       printf("Donnez le nom, prenom et la note de l'etudiant %d:\n",i+1);
       scanf("%s%s%d",&e[i].nom,&e[i].prenom,&e[i].note);
    }
    printf("nom, prenom de(s) etudiant(s) ayant une note >= 10 : \n");
    for(i=0;i<n;i++)
    {
        if(e[i].note>=10)
        printf("%s %s\n",e[i].nom,e[i].prenom);
    }
}
```

c) Corrigés d'exercice 5.3

```
#include<stdio.h>
#include<stdlib.h>
enum Couleur { trefle, carreau , pique, coeur};
enum Face {sept , huit , neuf , dix , valet , dame, roi , as};
struct Carte
    {
    Couleur couleur;
    Face face ;
    };
int main()
{
Carte jeu [32] ;
int x = 0 ;
for (int i = 0 ; i < 8 ; i ++)
    for (int j = 0 ; j < 4 ; j ++)
        {
           jeu [x].face = (Face)i ;
           jeu [x].couleur = (Couleur)j;
           x++;
        }
}
```

d) Corrigés d'exercice 5.4

```
#include<stdio.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct personne{
       char nom[20];
       int age;
}Personne;
int main()
{
    Personne t[100];
    int n,i,j;
    printf("Donnez le nombre de personnes a lire:\n");
    scanf("%d",&n);
```

```
    for(i=0;i<n;i++)
    {
        getchar();
        printf("Entrez le nom complet de la personne N%d:\t",i+1);
        gets(t[i].nom);
        printf("Entrez son age:\t");
        scanf("%d",&t[i].age);
    }
    printf("\nLe tabeau avant la suppression:\n");
    for(i=0;i<n;i++)
        printf("L'age de %s est: %d ans.\n",t[i].nom,t[i].age);

    for(i=0;i<n;i++)
    {
        if(t[i].age>=20)
        {
            for(j=i+1;j<n;j++)
            {
                strcpy(t[j-1].nom, t[j].nom);
                t[j-1].age = t[j].age;
            }
            n--;
            i--;
        }
    }
    printf("\nLe tabeau apres la suppression:\n");
    for(i=0;i<n;i++)
        printf("L'age de %s est: %d ans.\n",t[i].nom,t[i].age);
    system("pause");
    return 0;
}
```

e) Corrigés d'exercice 5.5

```
    #include <stdio.h>

    union hiddenMessage {
        int  ints[6];
        char chars[21];
    };

    int main() {
        union hiddenMessage intCharacters = {{1853169737, 1936876900, 1684955508,
1768838432, 561213039, 0}};

        printf("[");
        for(int i = 0; i < 19; ++i)
            printf("%c, ", intCharacters.chars[i]);
        printf("%c]\n", intCharacters.chars[19]);
        printf("%s\n", intCharacters.chars);
    }
```

Bibliographie

[1] Barrault Frantz. (2016). L'essentiel de l'informatique en prépa - Exemples et exercices corrigés en SQL et Python, Editions Ellipses.
[2] Picard. G., Jégou, R. Paradigmes algorithmiques. Quelques méthodes de conception d'algorithmes. [PowerPoint slides]. MINES Saint-Étienne. https://www.emse.fr/~picard/cours/prpd/PRPD-slides.pdf.
[3] Raaf, H. (2022). *Algorithmique et Structures de Données 1, Université des Sciences et de la Technologie Oran Mohamed-Boudiaf* [Online course]. https://elearning.univ-usto.dz/course/view.php?id=1363
[4] Tlemsani, R., Reguig, H., Nemmich, M. A. (2022-2023). *Algorithmique et Structures de Données 1 pour Math.* https://elearning.univ-usto.dz/course/view.php?id=2844
[5] Hannane, A. M., hannaneamirmokhtar2213. (2022, décembre). ASD1 [Vidéo]. YouTube. URL : https://www.youtube.com/watch?v=pzXYvVRVsbQ&list=PLZDMv-6RTV_yoMqTDGgm9RQPDURKnwe5g
[6] Messabihi, M. (2022). *Initiation à l'algorithmique, Faculté des Sciences. Université de Tlemcen* [Online course]. https://elearn.univ-tlemcen.dz/course/view.php?id=3606
[7] Si Tayeb, M. (2023). ASD1 - Cours En Ligne, *Université des Sciences et de la Technologie Oran Mohamed-Boudiaf* [Online course]. https://elearning.univ-usto.dz/course/view.php?id=1492
[8] w3resource. (2022). *C programming*. https://www.w3resource.com/c-programming/programming-in-c.php
[9] Ritchie, D. (2023). C Programming Guidebook - by Codeforwin, https://codeforwin.org/c-programming
[10] Senthil, K. (2022). Learn To Solve It, https://www.learntosolveit.com/index.html
[11] Messabihi, M. (2022). Algorithmique et Structures de Données 1, Faculté des Sciences. Université de M'sila [Online course]. https://elearning.univ-msila.dz/moodle/course/view.php?id=10217
[12] Nemmich, M. A. (2021). ASD1 : TD/TP, Université des Sciences et de la Technologie Oran Mohamed-Boudiaf [Online course]. https://elearning.univ-usto.dz/course/view.php?id=2669
[13] Bouziane, H. (2021). ASD1, Université des Sciences et de la Technologie Oran Mohamed-Boudiaf [Online course]. https://elearning.univ-usto.dz/course/view.php?id=1463