Université de Ain Temouchent –Belhadj Bouchaib
Faculté des Sciences et de la Technologie

Département des mathématiques et informatique

Polycopié pédagogique

Belgrana Fatima Zohra

Titre

Algorithmique, Structures de Données et Programmation en c

Cours et exercices (semestre 1)

Destiné aux étudiants de

1ère année, socle commun mathématiques, mathématiques appliquées et informatique

Année : 2022-2023

Avant propos

L’objectif de ce cours est de permettre aux étudiants d'acquérir des notions fondamentales de l’algorithmique et des structures de données ainsi que de la programmation en langage C.

Il est question dans un premier temps d'apprendre à analyser un problème sur le plan algorithmique, de détecter les différentes données d’entrée et de sortie, pour enfin choisir les structures de données adéquates. L’algorithme est une étape intermédiaire, l’étudiant apprendra par la suite comment le traduire dans un langage de programmation, le choix de ce dernier dépend des objectifs du programmeur.

Le langage choisi dans ce cours est le C, un langage facile, rapide et précis, il permet également de bien gérer la mémoire, les composants de l'ordinateur et les unités d'entrée et de sortie. C’est un langage parent de beaucoup d’autres, facilitant ainsi l’apprentissage de nouveaux langages de programmation.

Table des matières

Introduction générale

Aujourd'hui, l'informatique a pris beaucoup de place dans nos vies professionnelles et personnelles, elle est devenue omniprésente. Depuis l'invention de la machine de Turing (1936), en passant par l'architecture de John Von Neumann(1945), jusqu'aux modèles des micro-ordinateurs évolués d'aujourd'hui, l'objectif principal reste le même, il s'agit d'automatiser le traitement des données et de l'information. La programmation est à la base de tout développement informatique, que ce soit un site internet, un logiciel ou une application mobile. La réalisation de tout programme passe par plusieurs étapes importantes, dont l'algorithme est le maillon fort. Il faudrait par conséquent connaitre les différentes structures de données existantes et les briques de base de l'algorithmique. Le langage utilisé dans ce cours est le C, pour sa simplicité et sa structure mais aussi par ce qu'il est considéré comme un langage parent de plusieurs autres langages, ce qui facilitera leur apprentissage par la suite.

Ce polycopié est dédié aux étudiants du tronc commun en mathématiques et informatique, il pourra également être utilisé par les étudiants du tronc commun ST et SM. Il comporte des cours avec plein d'exemples et d'exercices corrigés. Il est présenté de façon graduelle où il est réparti en six chapitres, constituant ainsi le premier semestre de la matière Algorithmique et Structure de Données (ASD).

Le premier chapitre est une introduction au domaine de l'algorithmique et de la programmation. Il permet d'expliquer le lien entre les deux, et donne un aperçu du principe de fonctionnement d'un programme dans un ordinateur. Le second chapitre présente la structure linéaire et les éléments qui permettront au lecteur d'écrire son premier programme. Quant au troisième chapitre, il est consacré aux structures alternatives, il sera question ici d'invoquer la notion de condition ou de structures conditionnelles. Le quatrième chapitre est dédié aux structures répétitives, appelées aussi boucles. Dans le cinquième chapitre nous verrons les tableaux, vu que de simples variables ne peuvent suffirent à la résolution de problèmes avec de multiples données du même type, et quand ces derniers sont de types différents, il serait judicieux de faire appel aux structures personnalisées qui font l'objet du sixième et dernier chapitre.

CHAPITRE I

Introduction à l'algorithmique et la programmation

Plan du chapitre :

I.1 Introduction

Ce premier chapitre introduit quelques définitions clés pour la bonne compréhension des prochains cours, il présente ainsi quelques concepts de base, notamment la chaine de réalisation et d'exécution d'un programme dans un ordinateur, où nous verrons aussi le principe de fonctionnement de ce dernier.

I.2 Définitions

I.2.1 Informatique

L'informatique est l'ensemble des méthodes, d'outils et de techniques permettant le traitement et la communication de l'information. Ce terme à connu le jour en (1962) [1], il est créé à partir de deux mots :

- *Information*
- *Automatique*

L'informatique alors est une science du traitement de l'information de façon automatique par des machines (ordinateur). Deux axes fondamentaux qui sont en relation directe avec cette discipline :

✓ Le matériel (hardware): il s'agit d'un ensemble de dispositifs physiques utilisés pour ce traitement dit automatiquement des informations.

✓ Le logiciel (software) : c'est un ensemble structuré d'instructions décrivant un traitement d'informations à faire réaliser par un matériel informatique.

Les termes "**hardware" et "software**" s'utilisent essentiellement dans un contexte **informatique.**

I.2.2 Ordinateur

Selon LAROUSSE [2], un ordinateur (nom masculin : latin ordinator, celui qui met en ordre) est une machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.

Un ordinateur est une machine capable d'effectuer automatiquement des opérations arithmétiques et logiques (à des fins scientifiques, administratives, comptables, . . .) à partir de programmes définissant la séquence de ces opérations [3].

a. Principe de fonctionnement

L'ordinateur traite l'information en convertissant tous types de données en nombres binaires (uns et zéros), ensuite elle prend des décisions en utilisant de simples mathématiques.

Physiquement, un interrupteur allumé représente un 1, et un interrupteur éteint représente un 0. On appelle cela un *bit* d'information (voir figure I.1).

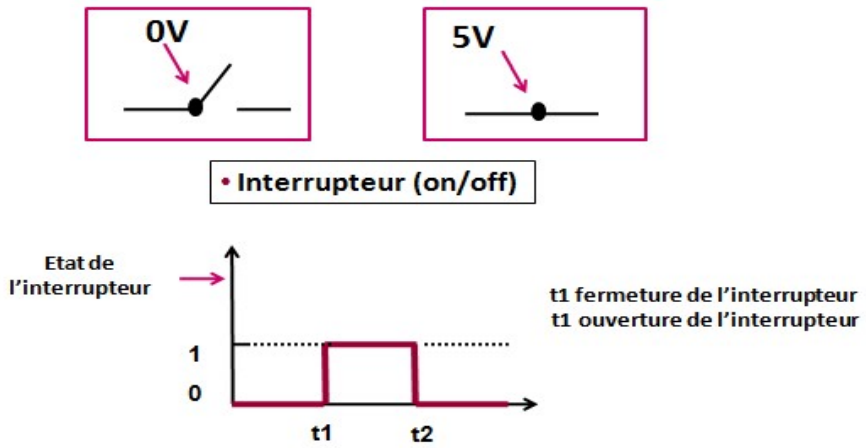


Figure I.1 Principe de fonctionnement des ordinateurs

I.3 Algorithme

Le mot algorithme vient du mathématicien arabe du 9ème siècle Abu Jaafar Muhammad bin Musa Al- **Khwarizmi,** le terme à cette époque était associé à l'algèbre. Aujourd'hui, elle représente la base de la programmation, c'est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre un problème donné. Nous pouvons prendre à titre d'exemple le mode d'emploi d'un télécopieur, où il est question d'un ensemble d'étapes à suivre (voir l'exemple ci-dessous) [4].

- **Exemple :** « Extrait du mode d'emploi d'un télécopieur concernant l'envoi d'un document ».
  - ✓ **Étape 1**. Insérez le document dans le chargeur automatique;
  - ✓ **Étape 2.** Composez le numéro de fax du destinataire à l'aide du pavé numérique ;
  - ✓ **Étape 3.** Enfoncez la touche d'envoi pour lancer l'émission.

Ce mode d'emploi précise comment envoyer un fax. Il est composé d'une suite ordonnée d'instructions (insérez..., composez..., enfoncez...) qui manipulent des données (document, chargeur automatique, numéro de fax, pavé numérique, touche d'envoi) pour réaliser la

tâche désirée (l'envoi d'un document). **Cette suite d'instructions peut être traduite en un langage informatique.**

Un algorithme prend en entrée des données et fournit un résultat permettant de donner la réponse à un problème, il est caractérisé par un langage de description constitué d'un ensemble de mots clés et de structures permettant de décrire de manière complète l'ensemble des opérations à exécuter sur des données pour obtenir des résultats .

Un algorithme est écrit en un langage compréhensible par les humains. Une phase d'analyse du problème est indispensable pour l'établissement d'un algorithme, elle permet de déterminer dans un premier temps, l'ensemble des données d'entrée et les résultats attendus [5].

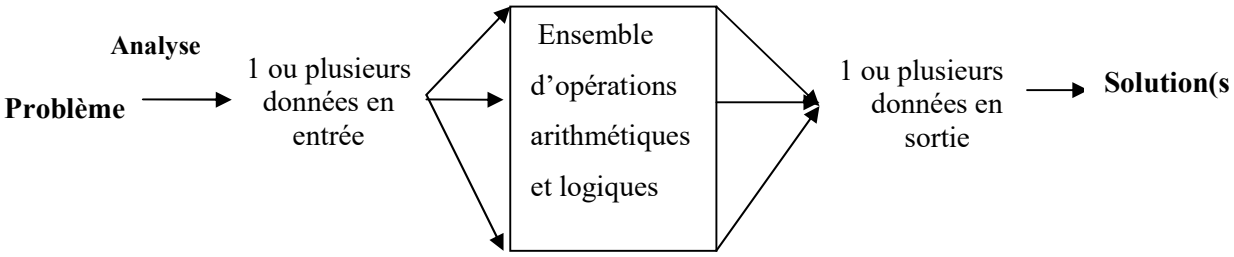


Figure I.2 Processus de réalisation d'un algorithme

I.4 Programme

Comme je expliqué dans la section précédente, l'algorithme est écrit en un langage compréhensible par les humains, mais malheureusement ce n'est pas le cas de la machine. Par conséquent, un algorithme doit être traduit dans un langage de programmation donnant ainsi naissance à un programme, appelé aussi code Source. Ce code source n'est à son tour compréhensible que par l'humain, car l'ordinateur est une machine à la quelle on ne peut s'adresser qu'en lui envoyant des 0 et des 1(voir section 1.2.2.a) appelés langage binaire (00101101100100110100111110….) ou langage machine [6], c'est alors ici qu'intervient le Compilateur (ou l'Interpréteur), sont rôle est de convertir ce programme vers le langage machine.

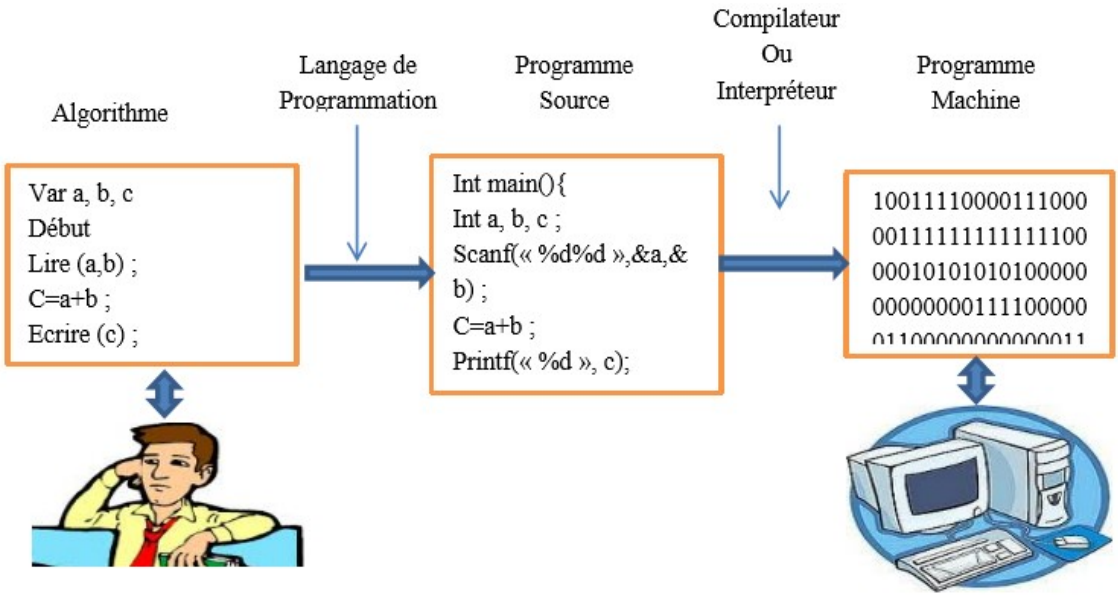


Figure I.3 Programme: de l'algorithme à la machine

I.4.1 Langage de programmation

Un langage de programmation est un langage informatique, permettant à un humain d'écrire un code source qui sera analysé par un ordinateur. Il s'agit d'un ensemble de mot clés, constituant la syntaxe de langage et ayant une sémantique bien définie, décrivant le comportement de la machine, on parle de langage évolué.

Il existe plusieurs langages, certains sont complémentaires, mais ils répondent à des objectifs différents. Y en a ceux qui sont destinés aux sites web statiques tels que le HTML et le CSS, ceux dédiés aux sites web dynamiques tels que le Php et le Java script, ceux qui sont adaptés au développement d'applications mobiles sur Androïdes tels que Kotlin, java et le c++, ces trois derniers langages ainsi que Python sont notamment utilisés pour développer des programmes de l'intelligence artificielle (IA), en plus d'autres langages tels Scala et Julia.

Parmi tous ces langages, nous allons voir ensemble, le langage c, existant depuis des années, ayant une communauté importante et une bibliothèque assez riches, il est utilisé dans la réalisation de nombreux logiciels, mais surtout des plus grands systèmes d'exploitation. Il a l'avantage d'être un langage minimaliste, c'est un langage parent de beaucoup d'autres langages évolués, ce qui facilitera l'apprentissage leur par la suite.

I.5 Compilateur et interpréteur

Un compilateur est un programme informatique qui transforme un code source écrit dans un langage de programmation (le langage source) en un autre langage compréhensible

par la machine (langage machine). C'est alors directement le système d'exploitation qui va utiliser le code machine et les données d'entrée pour calculer les données de sortie (solution).

Par contre un interpréteur, utilise directement le code source ainsi que les données en entrée, pour générer les données en sortie, en l'interprétant directement.

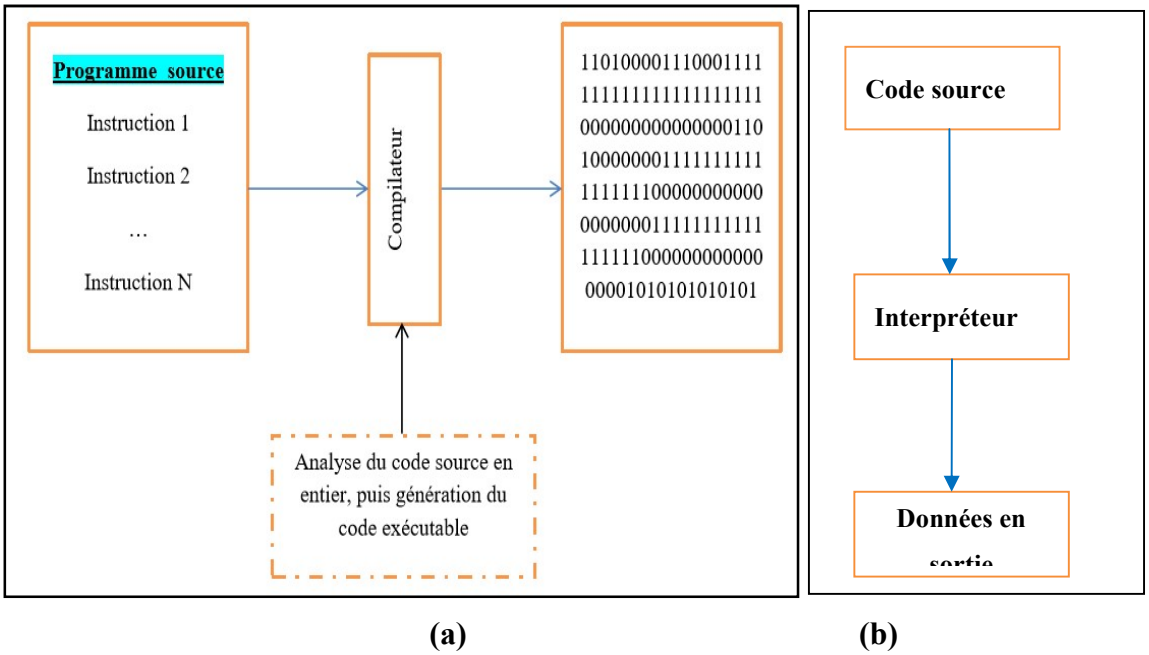


Figure I.4 (a) Processus de compilation, (b)Processus d'interprétation

I.6 Conclusion

Nous avons vu durant ce premier chapitre introductif, le voyage que font quelques idées permettant de résoudre un problème donné vers une solution établie automatiquement, grâce à un ordinateur. Ces idées résultantes d'une phase d'analyse, sont formulées en un algorithme qui sera traduit par la suite en un langage de programmation dont le choix dépend de l'objectif du programmeur. Il s'agit d'un pseudo-code qui va être ensuite compilé ou interprété pour être exécuté par la machine. L'algorithme ainsi que le programme ont une certaine syntaxe à respecter, cette syntaxe en langage de programmation tel que le C, est primordiale pour le processus d'exécution par la machine. Ils ont notamment un ensemble de structures de données permettant à aboutir à la solution optimale, sur ce prochain chapitre sera consacré à une de ces structures, il s'agit de la structure linéaire ou encore la structure séquentielle simple.

CHAPITRE II

Structure linéaire

Plan du chapitre :

II.1 Introduction

Les différents langages évolués reposent sur des concepts communs, tels que les variables qui représentent les différentes données manipulées, les instructions de base, les structures de contrôle ainsi que les structures de données. L'utilisation du compilateur impose encore une fois la distinction entre les instructions de déclaration et l'instruction exécutable, sur ce l'objectif de ce chapitre est de faire le point sur cette distinction, tout en donnant la structure générale et la syntaxe des instructions de base en algorithmique comme en langage C.

II.2. Partie 1: algorithme

II.2.1 Structure générale

Un algorithme est constitué de trois parties (voir figure II.1) :

- **L'entête**: permet d'identifier l'algorithme, caractérisé par le mot-clé **algorithme** suivi du nom de l'algorithme qui doit être significatif par rapport au traitement effectué par ce dernier;
- **Partie déclarative:** c'est une liste exhaustive des objets, grandeurs utilisées et manipulées dans le corps de l'algorithme; cette liste est placée en début d'algorithme. plusieurs types de déclaration sont possibles :
  - déclaration des données : il est question de déclarer toutes les données manipulées par l'algorithme, les constantes d'abord puis les variables;
  - déclaration des fonctions et procédures.
- **le corps de l'algorithme :** il contient les instructions de ce dernier, il est délimité par deux mots clés **début** et **fin**.

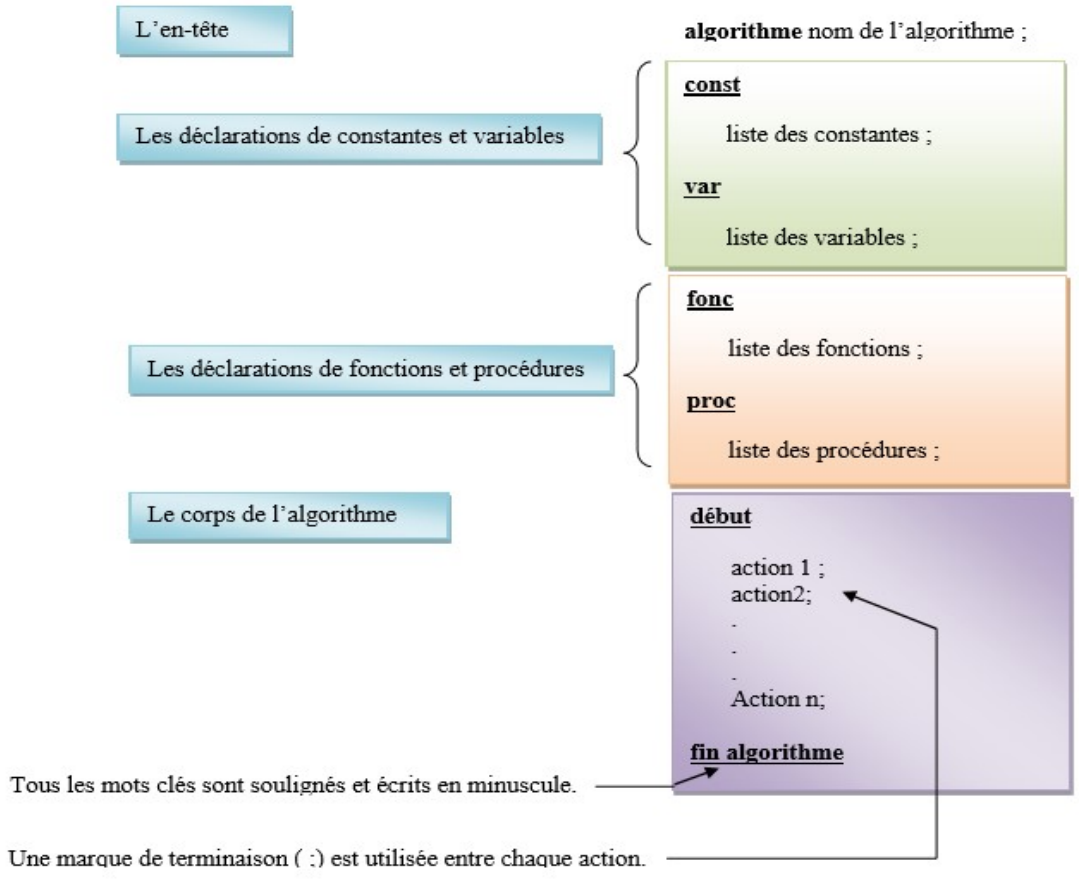


Figure II.1 Structure générale d'un algorithme

II.2.2 Les données de l'algorithme: variables et constantes

Afin d'aborder ce point, il est nécessaire d'expliquer la notion d'adresse mémoire, effectivement puisque les données manipulées sont stockées dans cette dernière. Ce qu'il faut savoir c'est que suite à la déclaration d'une donnée, une zone dans la mémoire va lui être attribuée, cette zone est caractérisée par une adresse, le nom de la donnée est utilisé comme référence à cette adresse mémoire (voir figure II.2). La taille de la zone dépend du type de la donnés, raison justement de la déclaration.

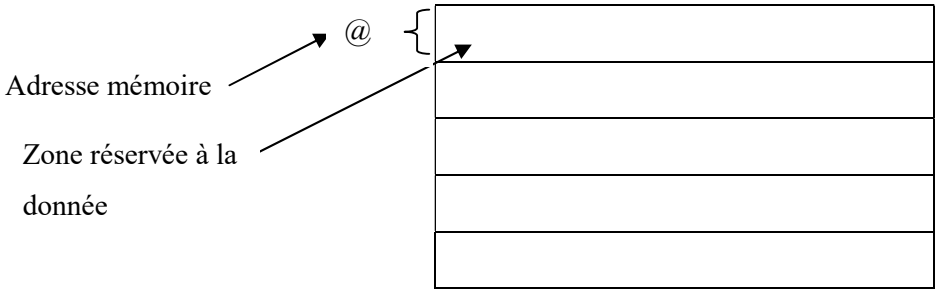


Figure II.2 Adresse mémoire

a. Les constantes

La valeur d'une constante ne peut pas être modifiée au cours de l'exécution de l'algorithme, elle peut représenter:

- des chiffres : 1, 8,…
- des nombres: 12, 28, 1856,5, …
- des caractères : a, x,?, espace, ...
- des chaînes de caractères, abc, bonjour, amine, rue Emir abedelkader 101 B9, …

La déclaration d'une constante se fait via le mot-clé : **const**

b. Les variables

Les variables peuvent stocker des chiffres, des nombres, des caractères, des chaînes de caractères, dont la valeur peut être modifiée au cours de l'exécution de l'algorithme.
La déclaration d'une variable se fait via le mot clé mot-clé : **var**

II.2.3 Type de données

Il existe plusieurs types de donnés, nous considérerons cinq types de base :

a. Le type entier

ce type permet de stocker les nombres entiers, on utilise par exemple ce type de données pour stocker l'âge d'une personne, le nombre d'étudiant dans une section, ou le code postal d'une ville.

- **Exemple:** 1, 5, 85,-5.
- **Mot clé : entier**

b. Le type réel

Le nombre flottant, permet de représenter les nombres avec virgule, sachant que l'ensemble des entiers est inclus dans les réels, ce type peut être utilisé par exemple pour représenter les dimensions des formes géométriques, le poids, etc.

- **Exemple:** 7.50, 19.569, –1564.01, 18.36 10 e-6
- **Mot clé : réel**

c. Le booléen

Il ne peut prendre que deux états: VRAI ou FAUX, il représente une réponse logique associée à une condition.

- **Mot clé : booléen**

d. Le caractère

Ce type ne peut contenir qu'un seul caractère, qui peut être une lettre alphabétique, et tous autres symboles que vous pouvez trouver sur votre clavier.

- **Exemple:** ' a ', 'A', '+', '7', 'z', ' =',….
- **Mot clé : car**

e. La chaîne de caractères

Ce type permet de contenir plusieurs caractères, il peut être utilisé pour représenter des noms, adresses ou même des phrases.

- **Exemple:** "informatique", "science"…
- **Mot clé : chaîne**

Puisque nous avons vu les différents types de variable, voici quelques exemples de déclaration :

- **Exemples :**
  - ✓ Déclaration d'une variable du type entier :

    Var x : **entier** ;
  - ✓ Déclaration d'une variable du type réel :

    Var y : **réel** ;
  - ✓ Déclaration d'une constante de valeur 10 (pesanteur)

    Const pes=10 ;

II.2.4 Opérateurs de base

Une instruction peut contenir une expression, cette dernière est constituée principalement de variable et d'opérateurs, nous allons voir dans ce qui suit les opérateurs de base.

a. Les opérateurs arithmétiques

Une opération arithmétique est constituée d'une ou plusieurs variables reliées par un ou plusieurs opérateurs, parmi ceux qui sont présentés dans la table II.1.

Table II.1 Les opérateurs arithmétiques en algorithme

Opérateur	Symbole
Addition	+
Soustraction	-
Multiplication	*

Division	/
Division entière	**Div**
Puissance	↑
Reste de la division	**Mod**

b. Les opérateurs de comparaison

L'utilisation des opérateurs de comparaison ou relationnels induit à une expression dite logique, dont le résultat et soit vrai ou faut. Ces opérateurs sont présentés dans la table II.2.

Table II.2 Les opérateurs de comparaison en algorithme

Opérateur	**Symbole**
Supérieur	>
Inférieur	<
Supérieur ou égal	>=
Inférieur ou égal	<=
Égal	=
Différent	≠

c. Les opérateurs logiques

Ce type d'opérateur sert à relier (associer) deux ou plusieurs conditions (expression logique). Il existe trois types d'opérateurs présentés dans la table II.3.

Table II.3 Les opérateurs logiques

Opérateur	**Symbole**
Et logique	Et
Ou logique	Ou
Non logique	Non

II.2.5 Les instructions de base d'un algorithme simple

Une instruction est un ordre qui permet de spécifier à la machine l'action à effectuer. Il existe trois types d'instructions primitives dites de base à savoir:

- L'instruction d'affectation;
- L'instruction de lecture;
- L'instruction d'écriture.

a. Instruction d'affectation

L'instruction d'affectation consiste à attribuer une valeur à une variable, la **syntaxe générale** est la suivante :

Nom-variable ← Expression

Après l'affectation, le contenu de la variable est modifié, il contient la valeur de l'expression de droite, l'expression peut être composée de variables, de constantes, et d'opérateur de base (voir section II.2.4).

On peut avoir une affection dans:

- ✓ Une **constante (dans la déclaration seulement)**.
  - **Exemple** : **const** surface ← 40;
- ✓ Une autre **variable**
  - **Exemple** : x ← y //sachant que y possède déjà une valeur
- ✓ Le résultat d'une **fonction**.
  - **Exemple** : résultat ← **racine** (nombre) ;
- ✓ Un calcul portant sur ces différents éléments.
  - **Exemple** : surface ← (PI * Carre (Diamètre)) / 4 ;

b. Instruction de lecture

Cette instruction permet de récupérer (entrer) une donnée à partir du périphérique d'entrée, généralement le clavier. La machine attend que l'utilisateur saisisse une valeur pour la stocker dans une variable (en mémoire), le mot-clé utilisé est **lire**.

- **Syntaxe**: **Lire (**nom-variable**)**

  Sachant que la variable est déjà déclarée.
- **Exemple: Lire**(x)
- **Remarque:** la valeur de x correspond à la valeur saisie au clavier par l'utilisateur, pour cela il faudrait justement lui demander de faire cette saisie.

c. Instruction d'écriture (d'affichage)

Cette instruction permet d'effectuer l'affichage à l'écran, généralement d'un résultat important issu de l'algorithme qui est stocké dans une variable, ou d'un message afin d'interagir avec l'utilisateur, le mot-clé utilisé est **Écrire.**

- ✓ **Syntaxe 1**: **Écrire (‘**expression**’)**;

  Permet d'afficher l'expression telle quelle à l'écran,
  - **Exemple**: **Écrire (‘entrer votre nom’)** ;

- ✓ **Syntaxe 2 : Écrire (**nom-variable**) ;**

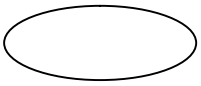
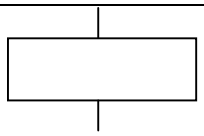
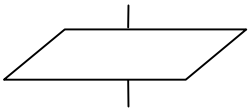
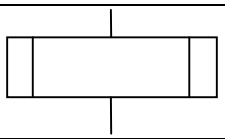
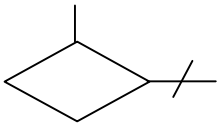
  Permet d'afficher le contenu de la variable,

  - **Exemple : Écrire** ('x=', x) ;

II.2.6 Représentation d'un algorithme par un algorigramme

Algorigramme ou appelé aussi organigramme, est une représentation graphique de l'algorithme (ou du programme). Il permet de visualiser facilement les différents blocs du code. Pour le construire, on utilise des symboles normalisés, la table II.4 présentée ci-dessous comporte quelques symboles utilisés dans sa construction définis par la norme la norme *NF-Z-67.*

Table II.4 Signification des symboles utilisés en algorigramme selon la norme NF-Z-67

Symbole	Désignation
	Début et fin
	Commentaire (indications sur les opérations effectuées)
	Une ou plusieurs instructions groupées pour lesquelles n'existe aucun autre symbole
	Les instructions d'entrée et sortie (lecture et écriture) ou enregistrement d'une information
	Appel de sous-programmes
	Condition
	Liaison entre les différents symboles

II. 2.7 Exercices

1). Exercice 01 :

Q1. Écrire un algorithme permettant d'afficher à l'écran le message "My first algorithm"

2). Exercice 02 :

Q1. Parmi ces affectations (considérées indépendamment les unes des autres), lesquelles provoqueront des erreurs, et pourquoi ?

```
Algorithme principal ;
Const E=5 ;
Var A, B, C : entier ;
    D : car ;
Début
    A ← B+C ;
    A ← A + B * C ;
    B ← A – D ;
    C ← E / B ;
    E ← E+1 ;
Fin algorithme
```

3). Exercice 03:

Q1. Quelles seront les valeurs de A et B après la suite d'instructions suivantes:

```
Algorithme calcul ;
Var A, B : entier ;
Début
    A←3 ;
    B←A*2 ;
    A←5 ;
Finalgorithme
```

4) Exercice 04 :

Q1. Écrire un algorithme permettant de faire l'addition de deux nombres **a** et **b** sachant que a est un entier **a**=11 et **b** est une constante b=24 et mettre le résultat dans la variable **c** puis l'afficher.

5). Exercice 05 :

Q1. Écrire un algorithme permettant de faire la division de deux nombres réels **a** et **b** sachant que les valeurs de **a** et **b** sont saisies par l'utilisateur. Le résultat ensuite doit être affiché

6). Exercice 06 :

Q1. Écrire un algorithme permettant de demander l'année de naissance de l'utilisateur puis calculer son âge et l'afficher

Q2. Donner l'algorigramme correspondant

II.2.8 Corrigé type des exercices

1) Exercice 01:

Q1. **Algorithme** affichage ;

Début

Écrire ('My first algorithm');

Fin algorithme

2) Exercice 02:

Q1. Algorithme principal ;

Const E=5 ;

Var A, B, C : **entier** ;

D : **car** ;

Début

A ← B+C ; // B et C n'ont pas encore de valeurs

A ← A + B * C ;

B ← A – D ; // erreur, les données ne sont pas du même type

C ← E / B ;

E ← E+1 ; //erreur, la valeur d'une constante ne peut pas être modifiée

Fin algorithme

- **Remarque :** les réponses sont données en commentaires. Le symbole « // » permet d'introduire des commentaires dans la ligne d'instruction

3) Exercice 03:

Q1. Algorithme calcul ;

Var A, B : **entier** ;

Début

A←3 ; //A=3

B←A*2; //B=6

A←5 ; // A=5

Fin Algorithme

Réponses : A=5 et B=6

4) Exercice 04:

Q1. Algorithme somme ;

Const b=24 ;

Var a, c: **entier ;**

Début

a←11 ;

c←a+b ;

Écrire (c);

Fin algorithme

5) Exercice 05:

Q1. Algorithme division ;

Var a,b,c :**réel ;**

Début

Écrire (“Donner deux nombres réels“) ;

Lire (a,b) ;

c←a/b ;

Écrire (b);

Fin algorithme

6) Exercice 06:

Q1.

Algorithme affichage ;

Var age, AN, AC : entier ;

Début

Écrire (‘Donner vote année de naissance’);

Lire(AN);

Écrire (‘Donner l'année en cours’);

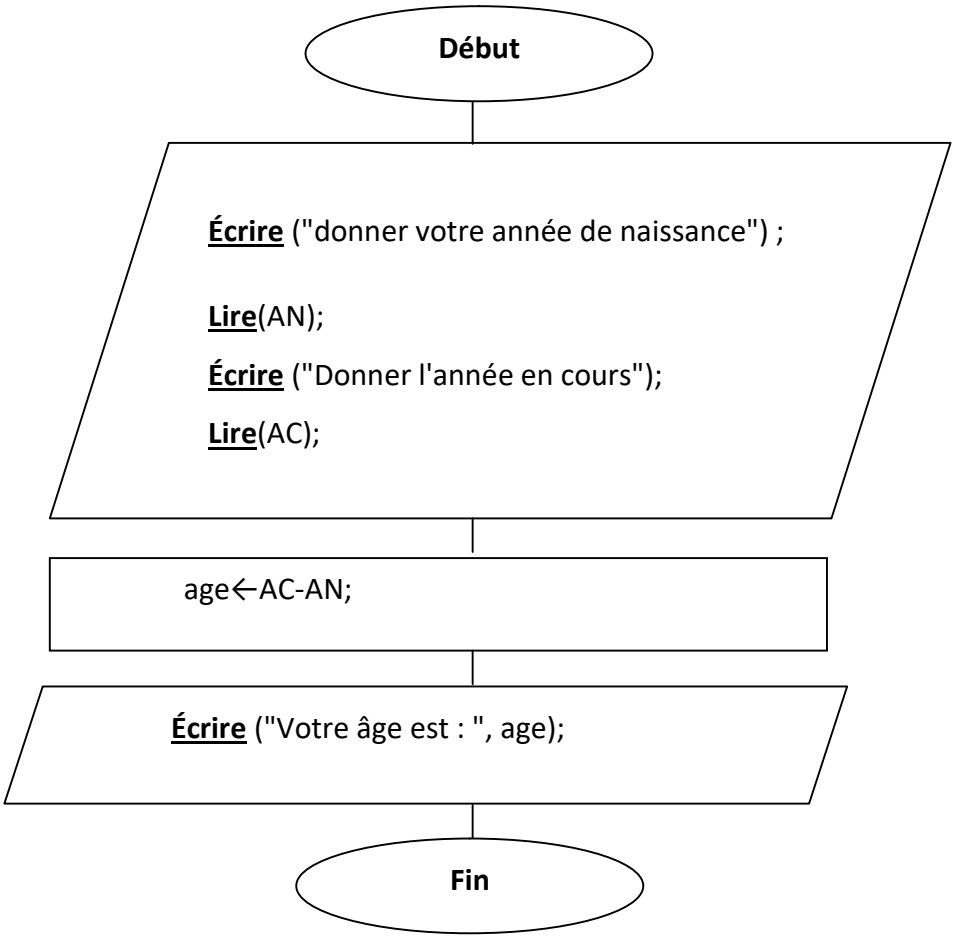
Lire(AC);

age←AC-AN;

Écrire (‘Votre âge est : ‘,age);

Finalgorithme

Q2.



II. 3 Partie 2: Programme en C

II.3.1 Rappels

✓ **Traduction en langage**

Nous avons déjà abordé ce point dans la section I.4, en donnant une brève définition, mais nous rappelons toutefois qu'un programme est une séquence d'**instructions**, d'ordres, donnés à l'ordinateur afin qu'il exécute des actions. L'élément responsable de l'exécution de ces ordres dans l'ordinateur est un composant électronique particulier nommé **processeur.**

Le programme doit être écrit en un langage de programmation évolué qui est ensuite traduit en langage machine grâce au compilateur, Il existe des milliers de langages de programmation de haut niveau tels que le pascal, JavaScript, Java, Python, PHP, C/C++, etc. Tout au long de nos cours nous étudierons le langage de programmation C.

✓ **Le langage de programmation C**

Le langage C [5] a été développé en 1972 par D. RITCHIE pour la récriture du système d'exploitation UNIX. Conçu à l'origine comme langage d'écriture du système, ce langage est utilisé pour la programmation de toutes sortes d'applications. il est encore très utilisé aujourd'hui, pour des raisons d'efficacité du code produit. Ce langage a été normalisé (c'est-à-dire d'établir des règles internationales et officielles pour ce langage) en 1989 par l'ANSI 7, puis en 1999 et 2011 par l'ISO 8 (normes C99 et C11) [6].

Le **C** est un langage de programmation évolué caractérisé par :

- Sa popularité : le c possède une communauté importante ;
- Son minimalisme : il possède un nombre réduit de concepts et/ou d'abstractions ce qui facilite son apprentissage ainsi que le développement.
- Ses performances : suite à son minimalisme, le C permet aux compilateurs de produire des exécutables performants sans trop occupé de l'espace mémoire
- Sa portabilité: un programme développé en C peut fonctionner dans différents environnements d'exécution (différentes machines et/ou systèmes d'exploitation) sans apporter de modification le code ne doive être modifié.

II.3.2 Structure d'un programme en C

Un programme c contient quatre parties importantes à savoir :

1. les directives destinées au processeur;
2. les déclarations des variables externes;

3. les fonctions secondaires: elles peuvent être placées indifféremment avant ou après la fonction principale;
4. la fonction principale (corps du programme): elle est caractérisée par le mot-clé **main()**, son corps est délimité par deux accolades, la syntaxe générale est comme suit :

La fonction principale

```
main()
{
    Déclarations de variables internes ;
    Bloc d'instructions ;
}
```

II.3.3 Les types prédéfinis

Le C est un langage *typé*, cela signifie en particulier que toute variable, constante ou fonction est d'un type précis. Suite à la déclaration, un espace mémoire est réservé pour chaque donnée, cet espace dépend principalement du type (voir la notion d'adresse, section II.2.2).

Les types de base en C, concernent les caractères, les entiers et les flottants (nombres réels), nous avons déjà abordé ce point dans la première partie de l'algorithmique, mais au niveau du langage c on peut également avoir d'autres sous types pour plus de précisions en matière d'espace en mémoire, ceci dépend aussi du type de processeur.

Les mots-clés utilisés, l'espace mémoire réservé ainsi que les intervalles des nombres possibles sont illustrés dans la table II.5 illustrée ci-dessous:

Table II.5 Syntaxe, borne et espace mémoire des principaux types de variables [7]

Syntaxe en c	Type	PC Intel (Linux)	Borne :
Shar	Caractère	8 bits	de $-(2^7)$ à 2^7-1 ([128,127])
Short	entier court	16 bits	de $-(2^{15})$ à $2^{15}-1$
Int	Entier	32 bits	de$-(2^{31})$ à $2^{31}-1$
Long	entier long	32 bits	de $-(2^{31})$ à $2^{31}-1$
long long	entier long (non ANS	64 bits	de $-(2^{63})$ à $2^{63}-1$

Float	Réel ou Flottant	32 bits	de $3.4*10^{-38}$ à $3.4*10^{38}$
Double	Flottant avec double précision	64 bits	De $1.7*10^{-308}$ à $1.7*10^{308}$
long double	Flottant avec quadruple précision	128 bits	de $3.4*10^{-4932}$ à $3.4*10^{4932}$

Voici quelques exemples de déclaration de variables:

- Les caractères : **char** c = 'A';
- Les entiers courts : **short** x **;**
- Les entiers longs : **long y ;**
- Les entier standard: **int** n **;**
- Les réels standard: **float z ;**
- Les réels avec double précisions : **double** d ;
- Les réels **avec quadruple précisions** : **long double** b **;**

Quant à la syntaxe générale de la déclaration des constantes, elle est comme suit :

- ✓ **Syntaxe** **const** type variable=valeur ;
  - **Exemple :** **const float** pi=3.14 ;

II.3.4 Les operateurs

Les opérateurs sont utilisés pour créer des expressions, qui font à leur tour partie des instructions du programme. Les différents symboles utilisés en langage c pour faire référence à ces opérateurs sont illustrés dans la table II.6.

Table II.6 Les symboles des différents opérateurs utilisés en langage C

Typé de l'opérateur	L'operateur	Le mot clef
Arithmétiques	Addition	+
	Soustraction	-
	Multiplication	*
	Division	/

	%reste de la division (modulo)	%
De comparaison	strictement supérieur	>
	supérieur ou égal	>=
	strictement inférieur	<
	inférieur ou égal	<=
	Egal	==
	différent	!=
Logique	et logique	&&
	ou logique	||
	négation logique	!

II.3.5 Les instructions de base

Trois instructions de base déjà vue en algorithmique, la différence réside dans la syntaxe, une description de chaque instruction accompagnée du mot-clé désigné en c sont illustrés dans la table II.7.

Table II.7 Instructions de base essentielle dans un programme linéaire (séquentiel simple)

Instruction	Description	Mot clef
Affectation	• Permet d'attribuer une valeur à une variable	=
Lecture (formatée)	• Fonction de scanne formatée • Elle permet de récupérer une valeur saisie au clavier et de la stocker dans une variable déclarée (adresse mémoire) • Fonction prédéfinit dans le fichier en-tête stdio.h	**scanf**
Ecriture (formatée)	• Fonction d'impression formatée • Elle permet d'afficher la valeur d'une variable ou une expression • Fonction prédéfinit dans le fichier en-tête stdio.h	**printf**

a. La syntaxe de la fonction printf

- ✓ **Syntaxe 1** : pour afficher un simple message: printf("message") ;
  - **Exemple:** printf("My first program") ;
- ✓ **Syntaxe 2 :** pour afficher la valeur d'une variable: printf ("%format ", nom-variable) ;

Le format de conversion dépend du type de la variable, la table II.8 illustre les formats associés à quelques types de variables.

Table II.8 Liste des formats en c associés à quelques types de variable

Format	Type de variable
%hd	Short
%d	Int
%ld	long int
%f	Float
%lf	Double
%lf	long double
%c	Char

- **Exemple:** afficher le contenu de la variable x de type entier

  printf ("%d ", x) ;
- **Exemple:** afficher le contenu de la variable x du type entier et y du type réel

  printf ("x=%d et y=%f", x) ;

b. La syntaxe de la fonction scanf

Une seule syntaxe possible : scanf("%format",&nom-variable) ;

Sachant que le symbole **&** est un opérateur d'adresse, il désigne l'adresse de la variable **nom-variable**, puisque effectivement les donnés sont stockés dans des adresses en mémoires (voir section II.2.2).

- **Exemple:** récupérer le contenu de la variable x, de type entier, saisi au clavier :

  scanf("%d",&x) ;

✓ **Interaction avec l'utilisateur**

Pour pouvoir récupérer le contenu (la valeur) d'une variable, il faudrait que l'utilisateur saisisse au clavier cette valeur-là, pour cela un message lui demandant de faire cette saisie doit être affiché, par conséquent une instruction d'écriture permettant cet affichage doit être rajoutée.

- **Exemple:** récupérer le contenu de la variable x du type entier qui doit être saisie au clavier par l'utilisateur

```
printf ("Veuillez saisir un nombre entier") ; / /interaction avec l'utilisateur
scanf("%d", & x) ;
```

✓ **Il est notamment possible de faire une lecture de plusieurs variables en une seule instruction**

- **Exemple** : récupérer le contenu de la variable **x** de type entier et **y** de type réel, saisies au clavier

```
printf("saisir un nombre entier et un nombre réel") ;
scanf("%d %f", &x, &y) ;
```

c. Inclusion du fichier entête

scanf et printf sont deux fonctions (parmi d'autres) prédéfinit dans un fichier source (bibliothèque), celui qui concerne ces deux fonctions est stdio.h (**S**tandard **I**nput/**O**utput **H**eader). L'inclusion de fichier source se fait via la directive:

```
#include <nom-fichier.h>
```

- **Exemple :** dans le cas des fonctions printf et scanf

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

II. 3.6 Exercices

1) Exercice 01 :

Q1. Créez un répertoire de travail dans la racine C : et nommez le TP

Q2. Lancez un envinement de d le DEV-C++, créez un nouveau fichier source et écrivez le code source suivant :

```
#include < stdio.h>          } Partie directive
Fonction principale → main ()
{
```

```
        printf("Bonjour\ n ") ;
        printf("mon premier programme C") ;
}
```

Corps de la fonction

Q3. Vérifiez votre répertoire de travail et lancez le fichier exécutable généré

Q4. Que fait votre programme, que permet \n (enlever et tester pour voir).

- **Remarque :** Dev-C++ est un environnement de développement intégré (IDE pour Integrated Development Environment en anglais) gratuit et en open source. Ce programme est destiné à programmer très facilement en langage C et C++ pour Windows.

2). Exercices 02 :

Q1. Écrire ces instructions en C d'un programme dans un fichier source :

```
int i=2;
float x;
x = 5/i;
```

Q2. Afficher à l'écran la valeur de x. Voyez-vous quelque chose qui vous semble étrange ? Si vous changez « x=5/i » dans votre programme par : x = 5/(float)i; que se passe-t-il ?

3). Exercices 03 :

Q1. Écrire le programme en C qui demande l'année de naissance de l'utilisateur et qui le lui affiche l'âge ensuite (algorithme de l'exercice 6, section II.2.7).

4) Exercice 04 :

Q1. Écrire un algorithme, qui demandent à l'utilisateur le rayon d'un cercle et lui affiche ensuite la surface et le périmètre.

Q2. Donner l'organigramme correspondant.

Q3. Écrire le programme équivalent

5) Exercice 05:

Q1. Écrire un programme en C qui permet d'échanger le contenu de deux variables x et y du type réel sachant que ces variables sont saisies au clavier.

Q2. Modifier le programme de telle sorte qu'il permet d'échanger le contenu de trois variables maintenant x, y et z du type entier de manière circulaire.

- **Exemple :**

  Avant permutation: x=5, y=2 et z=9

  Après permutation: x=2, y=9 et z=5

II. 3.7 Corrigé type des exercices

1). Exercice 01:

Q1. Aller au répertoire de la racine C, cliquer sur le bouton droit de la souris, choisir l'item « nouveau dossier » du menu contextuel (voir figure II.3).

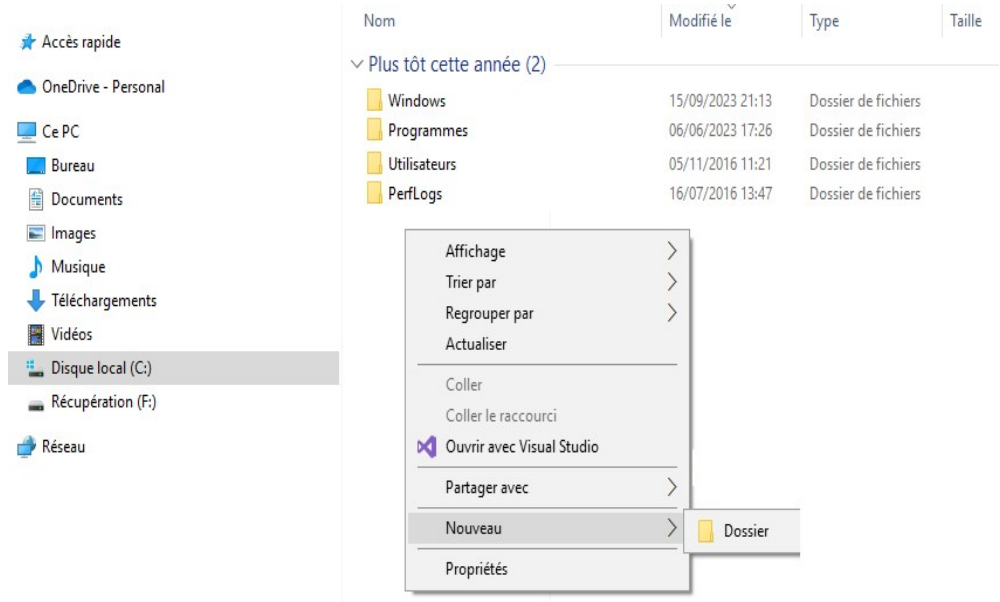


Figure II. 3 Création d'un nouveau répertoire de travail dans la racine

Q2. Choisir l'item « nouveau » puis « fichier source» (new, source file) du menu « fichier » (File). Finalement, taper le programme dans le ficher, puis enregistrer (Figure II.4).

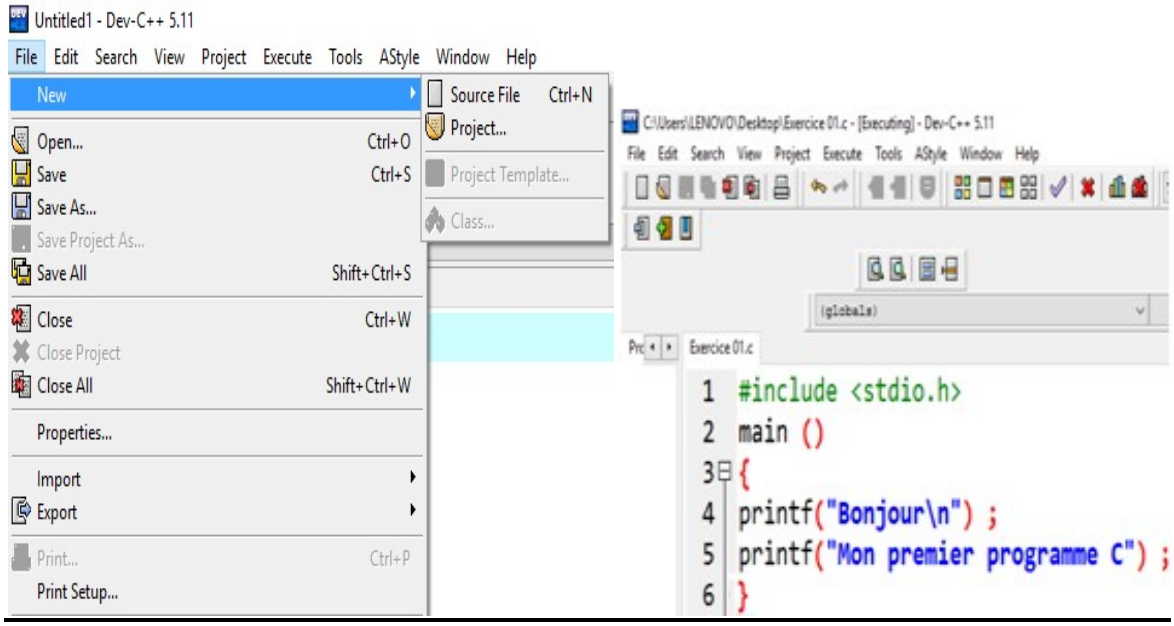


Figure II .4 Création d'un nouveau fichier source et écriture du code source

Q3. Apres exécution, un fichier. exe est généré, il s'agit du fichier exécutable correspondant au programme écrit (voir Figure II.5).

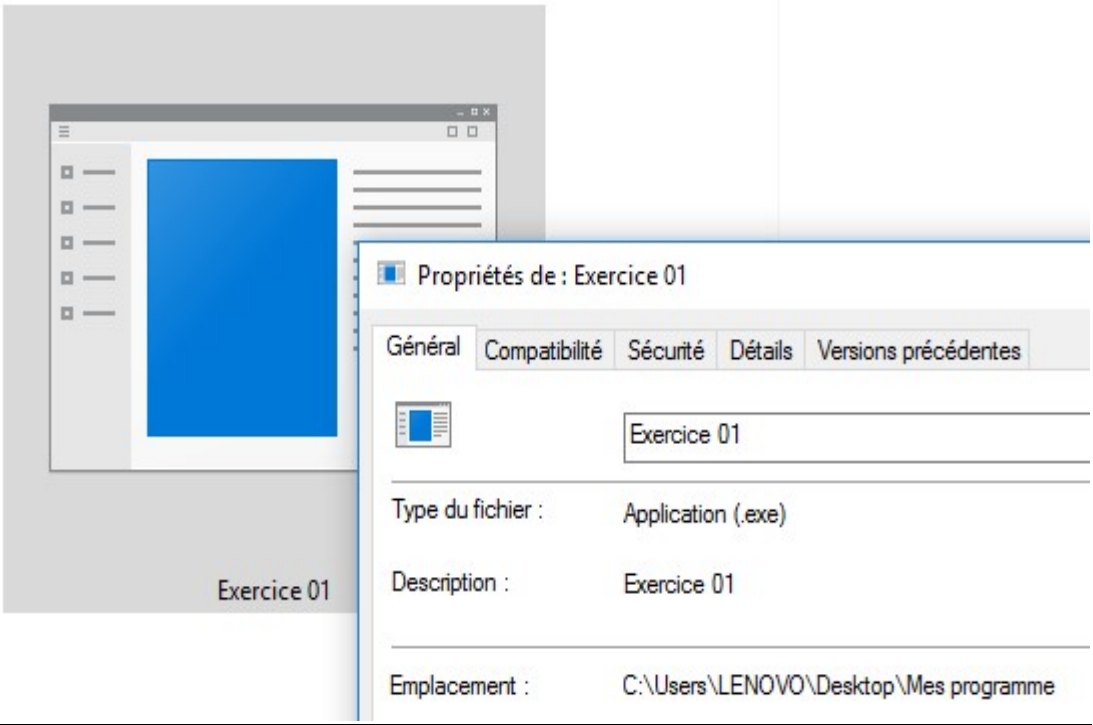


Figure II. 5 L'exécutable du fichier source généré après compilation et exécution

Q4. Le programme réalise des affichages de messages, \n permet de faire un saut de ligne (voir Figure II.6)

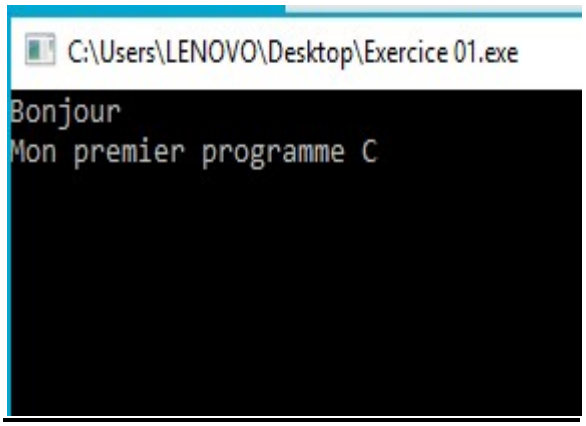


Figure II.6 Exécution du programme, résultat affiché sur la console.

2). Exercices 02 :

Q1. Voir la Figure II.7

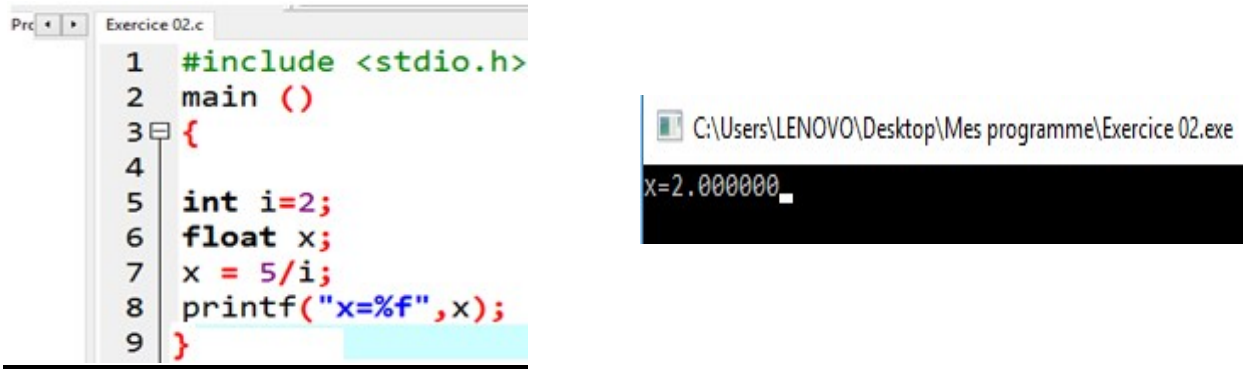


Figure II.7 Programme en C et son exécution

Q2. On remarque l'affichage de la valeur « 2 » de type entier, alors que le résultat devrait être égal à « 2.5 » de type réel. Si on substitue « x=5/i » dans le programme par : « x = 5/(float)i»; on obtiendra l'affichage du résultat réel « 2,5 », il est aussi possible d'arranger cela, on modifiant le type de la variable i à réel (voir la Figure II.8).

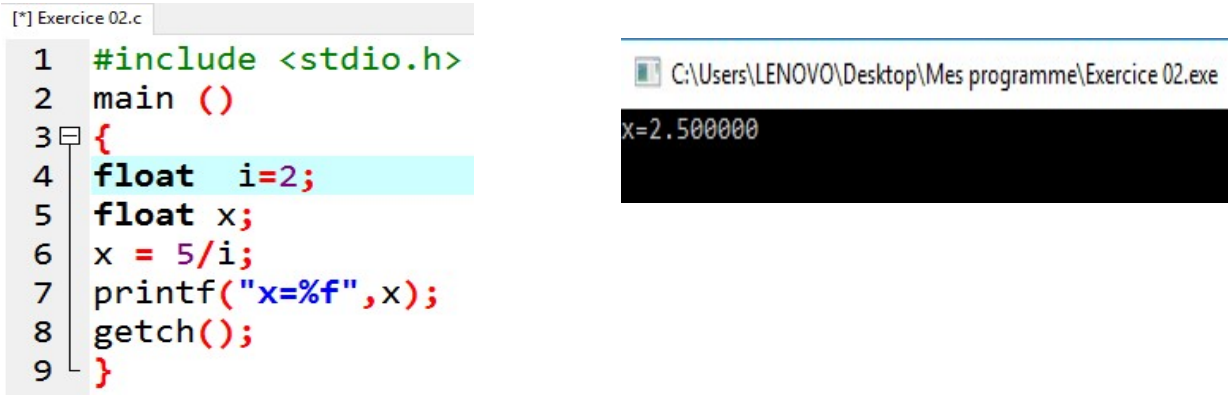


Figure II.8 Correction du programme en C précédant et son exécution

3). Exercices 03 :

```
# include<stdio.h>
main()
   {
        int age,AN,AC;
        printf("Donner vote année de naissance");
        scanf("%d",&AN);
        printf("Donner l'année en cours");
        scanf("%d",&AC);
```

```
    age=AC-AN;
    printf("Votre age est :%d",age);
}
```

4) Exercice 04 :

Q1.

Algorithme cercle ;

Const pi←3.14 ;

Var per, sur ,R: réel ;

Début

Écrire ("donner le rayon du cercle ") ;

Lire (R) ;

per←2*pi*R ;

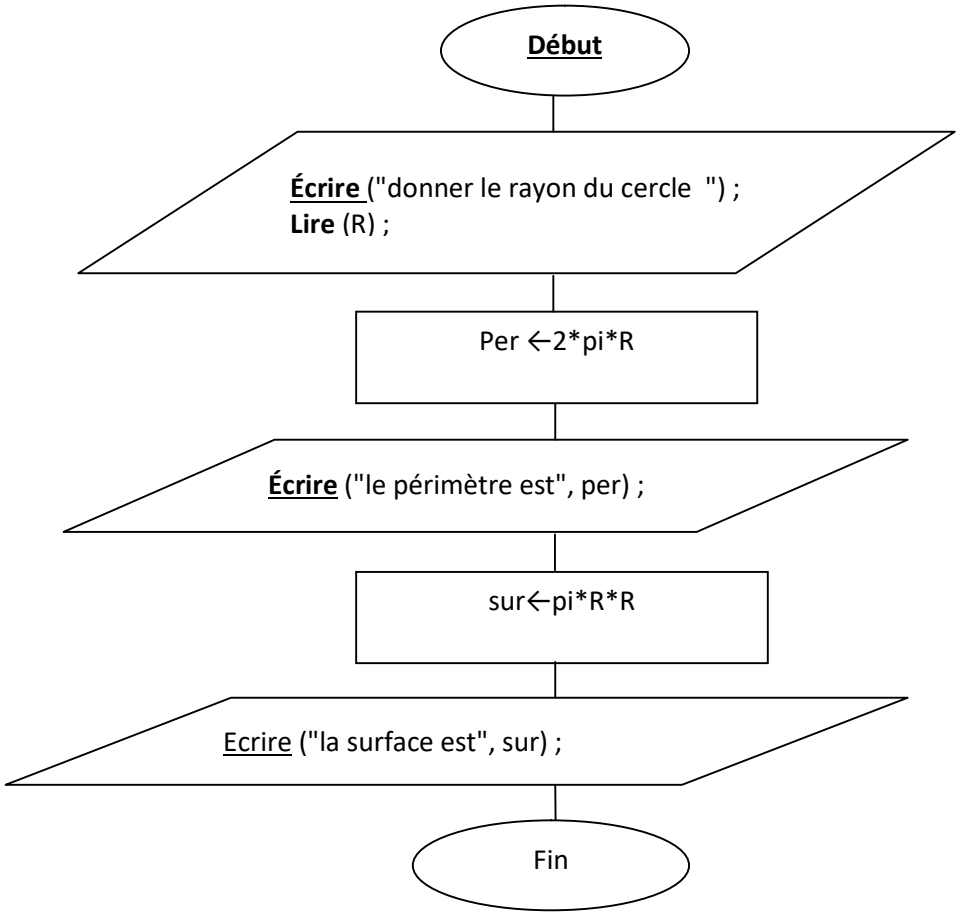
Écrire ("le périmètre est", per) ;

sur←pi*R*R ; // sur←pi* puissance(R,2) ;

Écrire ("la surface est", sur) ;

Finalgorithme.

Q2.



Q3.

```
#include <stdio.h>
main()
{
    const float pi=3.14 ;
    float per, sur ,R ;
    printf ("donner le rayon du cercle  ") ;
    scanf ("%f",&R) ;
    per=2*pi*R ;
    printf("le périmètre est %f", per) ;
    sur=pi*R*R ;//sur=pi*pow(R,2);
    printf("la surface est %f", sur) ;
}
```

5) Exercice 05:

Q1.	Q2.

main()
{
float x,y ,temp;
printf("Donner deux nombres entiers");
scanf("%f%f ",&x,&y);
temp=x ;
x=y ;
y=temp ;
}
	main()
	{
	float x,y,z ,temp;
	printf("Donner trois nombres entiers");
	scanf("%f%f %f",&x,&y,&z);
	temp=x ;
	x=y ;
	y=z ;
	z=temp ;
	}

II.7 Conclusion

Nous avons pu durant ce chapitre voir la structure générale d'un algorithme et celle d'un programme en c, nous avons également survolé les différents types existants de données, où nous avons illustré comment et pourquoi on procède à leur déclaration, tout en mettant l'accent sur la notion d'adresse mémoire. Nous avons présenté les principales instructions de base d'une structure séquentielle simple telles que les affectations, les fonctions de lecture et d'écriture. Malheureusement la structure séquentielle simple seule ne suffit pas, effectivement, il existe quelques types de problème où il serait question de plusieurs possibilités d'instruction reliées à des conditions, il s'agit ici de faire appel aux structures conditionnelles qui feront l'objet du chapitre suivant.

Chapitre III

Chapitre III: Les structures conditionnelles

Plan du chapitre :

III.1 Introduction

Il existe un ensemble de structures fondamentales (élémentaires) en algorithmique comme en programmation C (seule la syntaxe différencie les deux), relatives à la résolution d'un problème et qui peuvent, en fonction de leurs enchaînements être organisées suivant quatre familles de structures, à savoir les structures linéaires, les structures conditionnelles, les structures de choix multiples, et finalement les structures itératives (ou répétitives).

La structure séquentielle simple vue dans le chapitre précédent, dite aussi linéaire, se caractérise par une suite d'actions à exécuter successivement dans l'ordre énoncé (séquentiellement). Hélas cette structure seule s'est avérée insuffisante pour la résolution de quelques problèmes, où il est question de plusieurs alternatives de solutions possibles, qui dépendent principalement des données et qui varient d'une exécution à une autre. Pour palier à ce problème, il faudrait faire appel aux structures conditionnelles, principal objet de ce chapitre.

III.2 Définition

Cette structure n'offre que deux issues possibles à la poursuite de l'algorithme qui s'exclut mutuellement. Elle est caractérisée principalement par une condition.

On peut rencontrer deux types de structures conditionnelles :

- Structure conditionnelle complète
- Structure conditionnelle réduite

III.3 Type de structure conditionnelle

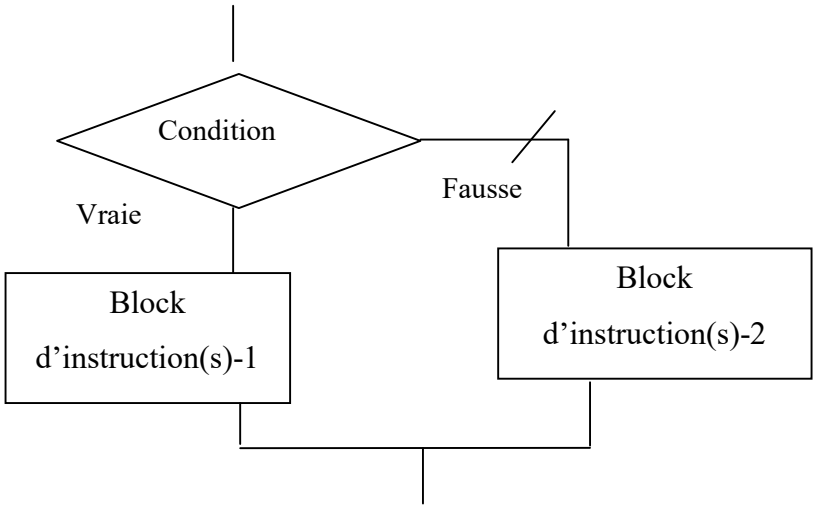
III.3.1 Structure conditionnelle complète

Dans ce type de structure, si la condition est vérifiée, seul le premier bloc d'instruction(s) est exécuté et si la condition n'est pas vérifiée, c'est le deuxième bloc d'instruction(s) qui est seulement exécuté.

Syntaxe algorithmique :	**Syntaxe en C :**

Si(Condition) **alors**
<block d'instruction(s)-1>
Sinon
<block d'instruction(s)-2>]
Finsi
	if (Condition)
	{<block d'instruction(s)-1>}
	else
	{<block d'instruction (s)-2>] }

Le format de l'algorigramme est comme suit :



- **Exemple :** parité d'un nombre x

En algorithme :	**En langage C :**
Si (x mod 2=0)**alors** **Écrire** ("x est pair'') **Sinon** **Écrire** ("x est impair'') **Finsi**	**if** (x%2==0) { **printf**("x est pair") ;} **else** {**printf**("x est impair") ;}

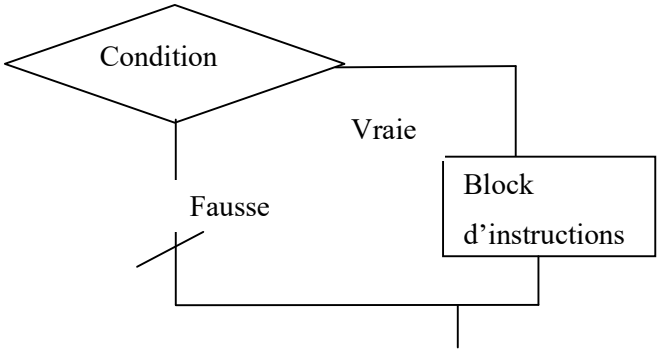
III.3.2 Structure conditionnelle réduite

La structure conditionnelle réduite se distingue de la précédente par le fait que seule la situation correspondante à la validation de la condition entraîne l'exécution du bloc d'instruction, l'autre situation conduis systématiquement à la sortie de la structure. Les syntaxes en algorithmique et en langage C sont présentées ci-dessous.

Syntaxe algorithmique :	**Syntaxe en C :**

Si (Condition) **alors**
< block d'instruction(s)>
Finsi
	if (Condition)
	{
	<s block d'instruction(s)>
	}

Le format de l'algorigramme est le suivant:



- **Remarque :** il est possible d'utiliser plusieurs fois la forme réduite pour exprimer plusieurs alternatives.
- **Exemple :** reprendre l'exemple précédent (parité d'un nombre) via la forme réduite

Syntaxe algorithmique	**Syntaxe en C :**
Si (x mod2=0) **alors** **Écrire** ('x est pair') ; **Finsi** **Si** (x mod2≠0) **alors** **Écrire** ('x est impair') ; **Finsi**	**if** (x%2==0) **printf** ("x est paire ") ; **if** (x%2 !=0) **printf** ("x est impaire ") ;

III.3.3. Structure conditionnelle imbriquée

La structure conditionnelle complète ouvre deux voies qui correspondent à deux flux d'exécution différents. Mais il y a des situations où deux voies ne suffisent pas. Alors on pourrait utiliser soit plusieurs formes conditionnelles réduites ou la forme conditionnelle imbriquée. La syntaxe est donnée comme suit:

Syntaxe en algorithmique :	**Syntaxe en C** :
Si (Condition1) **alors** <block d'instruction(s)-1> **Sinon si (**Condition 2) **alors** < block d'instruction(s)-2> **Sinon si (**Condition 3) **alors** <blockd'instruction(s)-3> **……** **Sinon** < block d'instruction(s)-n>	**if** (Condition 1) {< block d'instruction(s)-1>} **else if** (Condition 2) {< block d'instruction(s)-2>} **else if** (Condition 3) {< block d'instruction(s)-3>} **……** **else** {< block d'instruction(s)-n>}

```
        Finsi
    Finsi
  .....
Finsi
```

III.3.4 Structure de choix multiples

La structure de choix multiple permet en fonction de plusieurs conditions, d'effectuer des actions différentes, suivant les valeurs que peut prendre une même variable. Il s'agit dans ce cas d'une comparaison de type égalité seulement, il n'est alors pas possible de l'utiliser dans le cas d'inferiorité ou superiorité. Ça syntaxe algorithmique ainsi que sa syntaxe en langage C est présentée Ci-dessous :

Syntaxe algorithmique :	**Syntaxe en C** :
Suivant (nom-variable)**faire** : **cas** valeur1 : action1 ; **cas** valeur2 : action2 ; … … **cas** valeurN : actionN ; **sinon** actionN+1 ; **Finsuivant** ;	**switch (**nom-variable**)** **{** **case** valeur1 : action1 ; **break** ; **case** valeur2 : action2 ; **break,** …… **case** valeurN : actionN ; **break** ; **default** : actionN+1; **}**

Si la valeur de la variable (nom-variable) correspond à une des constantes (valeur1,…, valeurN), le traitement correspondant est exécuté. Sinon, le traitement par défaut via l'instruction ***default*** est exécuté.

III.4 Conditions composées

Une condition est une expression reliant deux données (variable ou constante) via un opérateur de comparaison, le resultat est une valeur booleéene, qui peut etre vraie ou fausse. Une condition composée est constitituée de plusieurs conditions reliées à leurs tours par les opérateurs logiques **et** et **ou**.

- **Exemple :**

Un étudiant est admis si ça moyenne est supérieur ou égale à 10 et il n'a aucune note éliminatoire des deux matières fondamentales(>5).

Syntaxe algorithmique	**En langage C :**
Si (moy>=10) et note1>5 et note2>5) **alors** **Écrire**("Admis") ; **Sinon** **Écrire** ("Ajourné") ; **Finsi ;**	**if** (moy>=10 && note1>5 && note2>5) **printf** ("Admis ") ; **else** **printf** ("Ajourné") ;

III.5 Exercices

1) Exercice 01:

Q1. Écrire un algorithme qui demande à l'utilisateur un entier et affiche sa parité.

Q2. Donner le programme C équivalent.

2) Exercice 02:

Q1. Ecrire un programme C qui permet de determiner le minimum et le maximum de deux nombres entiers saisis au clavier.

Q2. Modifier le programme afin de considerer le cas d'egalité des deux nombres saisis.

3).Exercice 03:

Q1. Ecrire un algorithme permettant de calculer le salaire d'un employé sachant que :

- L'utilisateur saisit le nombre d'heures travaillées, le salaire horaire et l'ancienneté de l'employé.
- Les retenues de sécurité sociale sont calculées à partir du salaire brut multiplié par le taux de retenue de la sécurité sociale qui est une constante valant 0.19.
- L'employé bénéficie d'une prime d'ancienneté qui équivaut à 2% du salaire brut pour une ancienneté comprise entre 10 et 20 ans et 5% du salaire brut pour ceux qui ont plus de 20 ans d'ancienneté.

4). Exercice 04:

Q1.Écrire un algorithme permettant d'afficher le mois en toutes lettres selon son numéro saisi au clavier.

5) Exercice 05 :

Q1.Écrire un algorithme permettant d'informer l'uilisateur si un caractére saisie par ce dernier est une voyelle ou une consonne.

Q2. Inclure le cas ou le caractére ne fait pas partie de l'alphabet.

Q3. Écrire le programme correspondant.

III.6 Corrigé type des exercices

1) Exercice 01 :

Q1. Algorithme	Q2. Programme

Algorithme pair-impair ;
var a: **entier** ;
Début
Écrire ("donner un nombre entier")
Lire (a) ;
Si (a mod2=0) **alors**
Écrire ("a est pair") ;
Sinon
Écrire ("a est impair") ;
Finsi
FinAlgorithme
	main()
	{
	int a ;
	printf ("donner unnombre entier") ;
	scanf("%d ", &a) ;
	if (a%2==0)
	printf("a est pair") ;
	else
	printf("a est impair") ;
	}

2) Exercice 02 :

Q1.	Q2.

main()
{ **int** x,y,max,min;
printf("donner deux nombre entiers");
scanf("%d%d",&x,&y);
if(x>y)
{
max=x;
min=y;
}
else
{
max=y;
min=x;
}
	main()
	{ **int** x,y,max,min;
	printf("donner deux nombre entiers");
	scanf("%d%d",&x,&y);
	if(x>y)
	{
	max=x;
	min=y;
	printf("max=%d,min=%d",max,min);
	}
	else if (x<y)
	{
	max=y;min=x;
	printf("max=%d,min=%d",max,min);

```
    printf("max=%d,min)%d",max,min);
}
```

```
    }
    else
    {
        max=x;
        min=x;
        printf("max=min=%d",max);
    }
}
```

3) Exercice 03 :

Algorithme salaire ;

Var nbH,salH,anc,salB,coeffPrime,salF: *réel* ;

Début

Écrire("Saisissez le nombre d'heures travaillées, le salaire horaire puis l'ancienneté de l'employé")

Lire (nbH, salH, anc) ;

salB←salH*nbH ;

Si(anc>= 20) **alors**

coeffPrime ←0.05

Sinonsi(anc>= 10)**alors**

coeffPrime ← 0.02

Sinon

coeffPrime =0 ;

Finsi

Finsi

salF←salB-(salB*0.19)+(salB* coeffPrime) ;/

Écrire ("Le salaire final",salF) ;

Finalgorithme

4) Exercice 04 :

Q1.Algorithme Mois ;

Var N : **entier** ;

Début

Écrire("Donner le numéro du mois de 1à 12:") ;

Lire(N) ;

Suivant N **faire** :

cas 1: **écrire**("Janvier");

cas 2 : **écrire** ("Février");

cas 3 : **écrire** ("Mars");

cas 4 : **écrire** ("Avril");

cas 5 : **écrire** ("Mai") ;

cas 6 : **écrire** ("Juin") ;

cas 7 : **écrire** ("Juillet");

cas 8 : **écrire** ("Août") ;

cas 9: **écrire** ("Septembre");

cas 10: **écrire** ("Octobre");

cas 11: **écrire** "Novembre");

cas 12 :**écrire** ("Décembre")

Sinon

Écrire ("Le numéro saisi est incorrecte :") ;

Fin Suivant ;

Finalgorithme ;

5) Exercice 05:

Q1	**Q2**
Algorithme consonne-voyelle ; **Var** X : **car** ; **Début** **Écrire**("Donner un caractère") ; **Lire**(X) ; **Suivant** X **faire**	**Algorithme** consonne-voyelle ; **Var** X : **car** ; **Début** **Écrire**("Donner un caractère") ; **Lire**(X) ; **Si((X>='a' et X<='z') ou(X>='A' et X<='Z')) alors**

```
        cas  'a' : écrire ("voyelle") ;
        cas  'A' : écrire ("voyelle") ;
        cas  'o' : écrire ("voyelle") ;
        cas  'O' : écrire ("voyelle") ;
        cas  'u' : écrire ("voyelle") ;
        cas  'U' : écrire ("voyelle") ;
        cas  'e' : écrire ("voyelle") ;
        cas  'E' : écrire ("voyelle") ;
        cas  'i' : écrire ("voyelle") ;
        cas  'I' : écrire ("voyelle") ;
        cas  'y' : écrire ("voyelle") ;
        cas  'Y' : écrire ("voyelle") ;
       sinon écrire ("consonne") ;
   Fin  suivant
Finalgorithme
```

```
      Suivant X faire
         cas  'a' : écrire ("voyelle") ;
         cas  'A' : écrire ("voyelle") ;
         cas  'o' : écrire ("voyelle") ;
         cas  'O' : écrire ("voyelle") ;
         cas  'u' : écrire ("voyelle") ;
         cas  'U' : écrire ("voyelle") ;
         cas  'e' : écrire ("voyelle") ;
         cas  'E' : écrire ("voyelle") ;
         cas  'i' : écrire ("voyelle") ;
         cas  'I' : écrire ("voyelle") ;
         cas  'y' : écrire ("voyelle") ;
         cas  'Y' : écrire ("voyelle") ;
         sinon écrire ("consonne") ;
      Fin suivant
   Sinon
      Écrire ("ne fait pas partie de l'alphabet) ;
   Finsi
Finalgorithme
```

Q3.

```
#include <stdio.h>
main()
{
  char x;
  printf("Donner une lettre");
  scanf("%c",&x);
  if((x>='a' &&x<='z')||(x>='A' &&x<='Z'))
  {
       switch (x)
       {
          case 'a': printf("voyelle");  break;
```

```
            case 'A': printf("voyelle"); break;
            case 'o': printf("voyelle");  break;
            case 'O':printf("voyelle"); break;
            case 'u': printf("voyelle");  break;
            case 'U ':printf("voyelle"); break;
            case 'e': printf("voyelle");    break;
            case 'E ':printf("voyelle");break;
            case 'i': printf("voyelle");break;
            case 'I': printf("voyelle");break;
            case 'y': printf("voyelle");break;
            case 'Y': printf("voyelle");break;
            case 'u': printf("voyelle");break;
            case 'U': printf("voyelle");break;
           default: printf("ce n'est pas unevoyelle"); break;
        }
      }
     else printf("ce n'est pas une lettre alphabétique");
  }
```

III.5 Conclusion

Nous avons pu voir dansce chapitre les différents types de structures conditionnelles, à savoir les formes réduites, complétes et imbriquées, ainsi que la structure de choix multiples. Il est à noter que ce type de strucutre est utilisé dans le cas d'un probléme avec une ou plusieurs alternatives de solutions, ou de situations ratachées avec des conditions.

Il faut savoir que parfois le programmeur est confronté à des cas ou il est question de répéter le meme traitement à plusieurs reprises, une structure adaptée à cela existe belle est bien, il s'agit des structures répétitives, que nous allons voir dans chapitre suivant.

CHAPITRE IV

Les structures répétitives

Plan du chapitre :

V.1 Introduction

Calculer la moyenne d'un étudiant revient à additionner les notes multipliées par le coefficient de chaque matière et ensuite diviser la valeur obtenue par la somme des coeifficients. Calculer la moyenne de **N** étudiants revient à repeter ce traitement **N** fois. Cependant, pour éviter de réécrire le code **pusieurs fois**, d**es structures répétitives** ou itératives (les boucles) ont été introduites. La structure itérative répète l'exécution d'une opération ou d'un traitement(bloc-d'instruction). On considère deux cas, dans le premier, le nombre de répétitions des instructions est connu, par contre dans le second cas, la répétition est plutôt rattachée à une condition.

V.2 Nombre de répétitions connu

Lorsque le nombre de répétitions est connu, il est plus judicieux d'utiliser la boucle POUR**.** Dans cette structure, la sortie de la boucle d'itération s'effectue lorsque le nombre souhaité de répétition est atteint.C'est pour cette raisonqu'on utilise une variable appelée aussi compteur (var-compteur), permettant de contrôler les itérations, cette variable est caractérisée par:

- sa valeur initiale (valeur-initiale) ;
- sa valeur finale (valeur-finale);
- son pas de variation (val-pas)

Les syntaxes en algorithmique et en langage C sont les suivantes:

✓ **Synatxe en algorithmique**

```
Pour var-compteur i de valeur-initiale à valeur-finale pas valeur-pas faire
    <Bloc-d'instruction> ;
FinPour
```

▪ **Synatxe en language C**

```
for(var-compteur= valeur-initiale ; var-compteur<=valeur-finale ; var-compteur++)
    {
        <Bloc-d'instructions> ;
    }
```

▪ **Exemple1 :** afficher 10 fois le message « ma première boucle »

En algorithme	En programme c
Pour i **de** 1 **à** 10 **pas** 1 **faire** **Écrire**("ma première boucle ") ;	**for(**i= 1; i<=10 ; i++**)** **{** **printf**("ma première boucle ") ;

FinPour	**}**

- **Exemple 2 :**demander de façon itérative à l'utilisateur un nombre entier, la demande de saisie s'arrête lorsque le nombre saisi est pair.
- **Remarque :**Le nombre de répétitions dans ce dernierexemple n'est pas connu, bien qu'il s'agit d'un problème itératif ; alors il n'est pas possible d'utiliser la boucle **pour**. Il serait plutôt question de voir le deuxième cas dans la section suivante.

V.3 Nombre de répétitions inconnu: boucle conditionelle (deuxiéme cas)

Dans ce cas, le nombre de répétion est variable, il dépend d'une condition. Il existe deux structures de base, la structure **RÉPÉTER JUSQU'À** et la structure **TANT QUE…FAIRE.**

V.3.1 La structure RÉPÉTER … JUSQU'À …

Dans cette structure, le traitement est exécuté une première fois, puis sa répétition se poursuit jusqu'à ce que la condition soit vérifiée (condition pour sortir de la boucle). L'action est toujours exécutée au moins une fois.

En langageC, la structure équivalente est **do_while**, il faudrait toute fois noter que la condition s'inverse, elle permet justement de repeter la boucle jusqu'à ce que la condition cesse d'être vérifiée.

Syntaxe en algorithmique	Syntaxe en langage C
Répéter <bloc-d'instructions> **jusqu'à**(condition)// **permet de sortir**	**Do** **{** <bloc-d'instructions> **}while (**condition**) ; //permet de boucler**

- **Exemple :** nous allons maintenant reprendre l'exemple2 de la section V.2 , pour ce faire on devrait poser la question suivante « doit-on exécuter aumoins une fois l'instruction d'écriture afin de demander de saisir un nombre ». La **Reponse est** oui, alors l'exercice peut étre résolu en utilisant la structure **répéter-jusqu'à**

Syntaxe en algorithme	Syntaxe en langage C
Algorithmedemande-repétitive	**main()**

Var n :**entier ;** **Debut** **Répéter** **Écrire**("donner une nombre entier") **;** **Lire (n);** **Jusqu'à (n mod 2 = 0)** **Fin algorithme**	**{** **int** n; **do** { **printf**("Donner un nombre entier"); **scanf**("%d",&n); }**while**(n%2!=0); **}**

- **Remarque:** la condition dans la syntaxe algorithmique est la négationde celle exprimée en langage C via la boucle do while.

V.3.2 La structure TANT QUE … FAIRE…

Dans cette structure, on commence par tester la condition; si elle est vérifiée, le traitement est exécuté. L'action peut alors ne jamais être exécutée.

Syntaxe en algorithmique	**Syntaxe en langage C**
Tant que(condition)**faire** **<bloc-d'instructions>** **Fintant que** ;	**while** (condition) **{** **<bloc-d'instructions>** **}**

- **Exemple 1:** nous allons reprendre l'exemple 2 de la section V.2 mais cette fois-ci en utilisant la sturture **TANT QUE … FAIRE…**

  Rappel de l'éxemple: demander de façon itérative à l'utilisateur un nombre entier, la demande de saisie s'arrête lorsque le nombre saisi est pair.

En algorithme	En langage c
Algorithmedemande-repétitive **Var** n :**entier ;** **Debut** **Écrire**("donner une nombre entier") **;** **Lire(**n**) ;** **Tant que(n mod 2 ≠ 0)** **Écrire**("donner une nombre entier") **;** **Lire(n) ;** **Fin tant que** **Fin algorithme**	**main()** **{** **int** n; **printf**(("donner une nombre entier") ; **scanf**("%d",&n); **while** (n%2!=0) **{** **printf**("Donner un nombre entier"); **scanf**("%d",&n); **}** **}**

V.4 Incrémentation et décrémentation

La variable compteur, utilisée par defaut dans la boucle **POUR(for)**, peut etre également utilisée par les boucles **répéter. ..jusqu'à (do.. while())** et **tant que()… faire(while()).**
Cette variable peut etre incrémentée, décrémentée, d'un pas ou plus.

- **Exemple 1 :** compte à rebour à partir de 10

Dans cette exemple il est question d'afficher les valeurs de 10 à 1 d'une façon décrémentée, le nombre de répétitions étant connu, la boucle POUR peut etre utilisée. Les boucles **tant que** et **répeter** sont généralement attachée à une condition, mais elle peuvent également êtres utilisées lorsque le nombre de répétitions est connu, il s'uffit de rajouter le compteur, ce dernier doit être initilalisé avant la boucle, puis incrémenté ou décrémenté (c'est lecas de cet exemple) à l'interieur de la boucle.

Pour i de 10 à 1 **par pas de** -1 **faire** **Écrire**(i) ; **Fin Pour**	i=10 ; **Tant que** (i>0) **Écrire**(i) ; i=i-1 ; **Fin Tant que**	i=10 ; **Repeter** **Écrire**(i); i=i-1 ; **jsuqu'a** (i==0)
for(i=10;i>0 ;i--) **printf**(" %d", i) ;	i=10 ; **while**(i>0) { **printf**(" %d", i) ;	i=10 ; **do**{ **printf**(" %d", i) ;

	i=i-1 ; }	i=i-1 ; }**while**(i>0) ;

V.5 Les boucles imbriquées

Dans certains problémes, il est nécéssaire d'utiliser les boucles imbriquées. Une boucle peut tou

t à fait contenir d'autres boucles.

- **Exemple 1 :** calculer la moyénne **de 10** etudiants ayant 4 notes d'examens (4matiéres)nécéssite de saisir les 4 notes des **10 étudiants**, calculer leurs moyennes etles afficher.
  - Il faut alors une boucle pour saisir les notes , calculer la somme puis la moyenne
  - Une deuxiéme boucle pour itérer sur tousles etudiants

  Il s'agit alors ici des boucles imbriquées.

```
Algorithme moyenne-étudiants
Var Not,j,i : entiers ;
    Som,Moy : réelle ;
Début
 Pour j de 1 à 10 pas 1 faire//ou j=0 ;tant que(j<=10) faire
    Som←0 ;
    Pour i de 1 à 4 pas 1 faire
      Écrire ("Donner la note",i) ;
      Lire(Not) ;
      Som←Som+Not ;
    FinPour
      Moy←Som/4 ;                    // j=j+1
    Écrire ("la moyenne de l'etudiant ",j, "est",Moy ) ;
  Finpour// fin tant que
Fin algorithme
```

Boucle intérieure

Boucle extérieure

- **Remarque :** il faut être particulièrement vigilant aux variables qui sont modifiées pendant les itérations d'une boucle imbriquée car celles-ci sont susceptibles d'avoir des répercussions sur l'autre boucle. En particulier, on veillera à ne pas modifier le compteur de la boucle externe dans la boucle imbriquée.

V.6 Exercices

1) Exercice 01 :

Q1.Écrire un algorithme qui permet d'afficher tous les nombres entiers naturels dont le cube est inférieur à 5000.

Q2. Donner le programme équivalant.

2) Exercice 02 :

Q1.Écrire un algorithme qui calcule la somme des N premiers entiers :

$$S= 1+2+3+4+5+\ldots+N$$

Q2. Donner le programme correspondant

3) Exercice 3:

Q1. Écrire l'algorithme et le programme permettant de calculer le PPCM(plus petit commun multiple)entre deux nombres entiers.

Q2. Donner le programme correspendant.

4) Exercice 4:

Q1. Écrire l'algorithme et le programme permettant de calculer le PGCD(plus grand commun diviseur)entre deux nombres entiers.

Q2. Donner le programme correspondant.

5) Exercice 5:

Q1.Écrire un algorithme permettant d'afficher les nombres de 5 à 25

Q2. Donner le programme en C correspondant.

6) Exercice 6:

Q1. Écrire un algorithme qui affiche la table de multiplication d'un chiffre. Ce chiffre sera saisi par l'utilisateur.

Q2. Donner le programme en C correspondant.

V.7 Corrigé type des exercices

1)Exercice 01 :

Q1. L'algorithme :	Q2. Le programme :

Algorithme cub-5000
Var n,i :**entier ;**
Debut
i←0 ;
Tant que(puissance(i,3)<5000**)**
Érire(i,",") **;**
i←i+1 ;
Fin tant que
Fin algorithme
	#include<math.h>
	main()
	{ int n,i ;
	i=0;
	while (pow(i,3)<5000)
	{
	printf("%d\n",i);
	i=i+1;
	}
	}

- **Remarque:** en langage C, la fonction **pow** est utilisée pour trouver la puissance d'un nombre, elle est déclaré dans le fichier entête **math.h.**

2)Exercice 02:

Q1. Algorithme :	Q2. Le programme en c

Algorithme somme-N-entiers
Var N ,i, S:**entiers ;**
Début
Écrire ("Donner un nombre entier") ;
Lire(N) ;
S←0 ;
Pour i **de** 1 **à** N par **pas** de 1 **faire**
S←S+i ;
Finpour
Écrire ("la somme des", N, "premiers entiers =",S) ;
Fin algorithme
	main()
	{
	int N , i, S;
	printf("Donner un nombre entier") ;
	scanf("%d",&N) ;
	S=0 ;
	for(i=1; i<=N;i++)
	S=S+i ;
	printf ("la somme des %d premiers entiers =%d", N,S) ;
	}

3)Exercice 03:

- **Etape analyse(comment calculer le ppcm) :**

Pour calculer le ppcm des deux nombres 132 et 72, nous devrions considérer leurs multiples en commençant par le plus petit nombre, comme suit :

- Les multiples de 72 sont :
  0 ; 72 ; 144 ; 216 ; 288 ; 360 ; 432 ; 504; 576 ; 648 ; 720 ; **792** ; ... (liste infinie)
  Rajouter la valeur du nombre 72, jusqua ce qu'il soit supérieur à la liste de valeurs du deuxieme nombre
- Les multiples de 132 sont :
  0 ; 132 ; 264 ; 396 ; 528 ; 660 ; **792** ; ... (liste infinie)
- Rajouter la valeur du nombre 132, jusqua ce qu'il soit supérieur à la liste de valeurs du premier nombre
- **Critére d'arrêt :** le processus est répété jusqu'à ce que les dernières valeurs sont identiques, dans l'exemple la valeur est 792.

Q1. Algorithme :	Q2. Le programme en c

Algorithme ppcm ;
Var x,y,a,b,ppcm **: entier ;**
Début
Écrire("donner deux nombre entier");
Lire(x,y);
a←x;
b←y;
Tant que (a≠b) **faire**
Si (a<b) alors a←a+x;
Sinon si (a>b) **alors** b←b+y;
Finsi
Finsi
ppcm←a;
Écrire("le ppcm",ppcm);
Fin algorithme
	main()
	{
	int x,y,a,b,ppcm;
	printf("donner deux nombre entier");
	scanf("%d %d",&x,&y);
	a=x;
	b=y;
	while(a!=b)
	{
	if (a<b) a=a+x;
	else if (a>b) b=b+y;
	}
	ppcm=a;
	printf("le ppcm=%d",ppcm);
	}

4)Exercice 04:

Etape analyse(comment calculer le PGCD):

Algorithme d'Euclide

- si (b=0) alors pgcd(a,b) =a
- sinon pgcd(a,b) = pgcd(b, a mod b)

Prenons a titre d'éxemple le pgcd de 96 et 81 :

pgcd((96, 81) = pgcd(81, 15) = pgcd(15, 6) = pgcd(6,3)=pgcd(3,0)=3

Q1. Algorithme :	Q2. Le programme en c

Algorithme PGCD;
var
a,b,r,pgcd : entier ;
Écrire("donner deux nombre entier");
Lire(a, b);
Tant que (b>0) faire
r← a mod b ;
a ←b ;
b←r ;
Fin Tant que
pgcd←a;
Écrire("le pgcd :",pgcd);
Fin algorithme
	main()
	{
	int a,b,r,pgcd;
	printf("Donner deux nombres entiers ");
	scanf("%d %d",&x,&y);
	while (b > 0) {
	r = a % b;
	a = b;
	b = r;
	}
	pgcd=a;
	printf(" pgcd= %d",pgcd);
	}

5) Exercice 05 :

Q1.Algorithme	Q2.programme C
Algorithme affichage ; **Var** i : **entier;** **Début** **Pour i de** 5 **à** 25 **pas** 1 **faire** **Écrire**(i, ",") ; **Finpour ;**	**main()** **{** **int** i **;** **for(**i= 5; i<=25; i++**)** **{** **printf**("%d,",i) ;

Finalgorithme	**}** **}**

6) Exercice 06 :

Q1. Algorithme	Q2. Programme en c
Algorithme tab-mult() ; **Var**i, N:**entier;** **Début** **Ecrire (**"Donner un nombre entier"**) ;** **Lire (N) ;** **Pour** i **de** 1 **à** 10 **pas** 1 **faire** **Ecrire(**N, "*",i, "=",N*i) ; **Finpour ;** **Finalgorithme**	**main()** **{** **int** i **,N;** **printf(**"Donner un nombre entier"**) ;** **scanf(**"%d,&N"**) ;** **for(**i= 1; i<=10 ;i++**)** **{** **printf**("%d*%d=%d,",N,i,i*N) ; ; **}** **}**

V.8 Conclusion

Nous avons pu voir durant ce chapitre, les differents types d'instructions de répétition existants en algorithme comme en programme C, oú il y a exactement trois type de boucles, à savoir la boucle **POUR(for)**, la boucle **Répéter…jusqu'à**()(**do.. while())**et **la boucle Tantque()… fair… (while())**. La boucle **POUR** ne peut etre utilisée que lorsque le nombre de répétition est connu, par contre les deux autres boucles peuvent etre utilisées et dans le premier cas, et lorsque le critere d'arret est lié à une condition, avec la différence de verifier cette derniere en amont ou en aval.Ces boucle sont hélas le seul moyen de manipuler une nouvel structure de données que nous allons voir dans le chapitre suivant, il s'agit des tableaux.

CHAPITRE V

Les tableaux et les chaines de caractères

Plan du chapitre :

V.1 Introduction

Nous avons vu durant les chapitres précédents, les instrucions de base à savoir la lecture, l'ecriture et l'affectation, nous avons notament présenté quelques structures telles que les boucles et les conditions. Il était aussi question et dans les premier temps d'exposer les differents types de données simples tels que les entiers, les réels et les constantes, malheureusement face à un certain types de problémes, l'utilsation des types simples, seul ne peut aboutir à une bonne et correcte résolution, par conséquent, nous allons voir dans ce qui suit une nouvelle structure de données, il sagit des tableaux.

V.2 Problématique

Pour comprendre mieux l'intérêt de l'utilisation des tableaux, prenant un exemple de problème, où on voudrait réaliser un programme (algorithme) permettant de saisir d'abord toutes les notes des examens de **N** matières du semestre 1 de **M** étudiants, puis par la suite calculer la moyenne de chaque étudiant, et afficher les notes et la moyenne de l'ensemble.

- Pour un étudiant il nous faut alors déclarer :
  - ✓ **N** variables pour les notes : $note_1$,$note_2$, …$note_N$;
  - ✓ Une variable pour la moyenne d'un étudiant
- Par contre, pour **M** étudiant il nous faut alors déclarer :
  - ✓ **M *N** (N variables pour les notes : $note_1$,$note_2$, …$note_N$)
  - ✓ **M** variables pour la moyenne de M étudiant

Malheureusement procéder ainsi peut causer quelques problèmes, il faudrait connaitre au départ N, le nombre de matières (pour déclarer le bon nombre de variables), en plus c'est une notation très lourde notamment si le nombre de matières **(N)** est élevé et, encore plus lourde si le nombre des étudiants **(M)** est élevé.

Pour pallier ces problèmes, il faudrait rassembler toutes ces variables dans une structure de données particulière appelée : tableau, il s'agit d'un type de variables composées et indicées.

V.3 Les tableaux à une dimension

Un tableau (ou encore une variable indicée) est une structure de données qui contient un ensemble d'éléments de **même** type (exemple tableau des entiers, tableau de caractères, …) et chaque élément à une position définie dans le tableau par un indice qui est obligatoirement du type **entier,** mais il peut être exprimé comme une variable ou une expression calculée de type entier bien évidement.

V.3.1 Caractéristiques d'un tableau

Un tableau est caractérisé par :

- Une taille fixe, définie lors de sa déclaration;
- des éléments, ou chaque élément est manipulé individuellement ;
- un type qui représente le type de ses éléments (entier, réel, caractère,…);
- chaque élément du tableau est désigné par le nom du tableau, suivi de l'indice de l'élément, entre crochets ;
- l'indice représente la position, en langage C les indices commencent par 0.

Exemple : tableau de 5 éléments du type entier

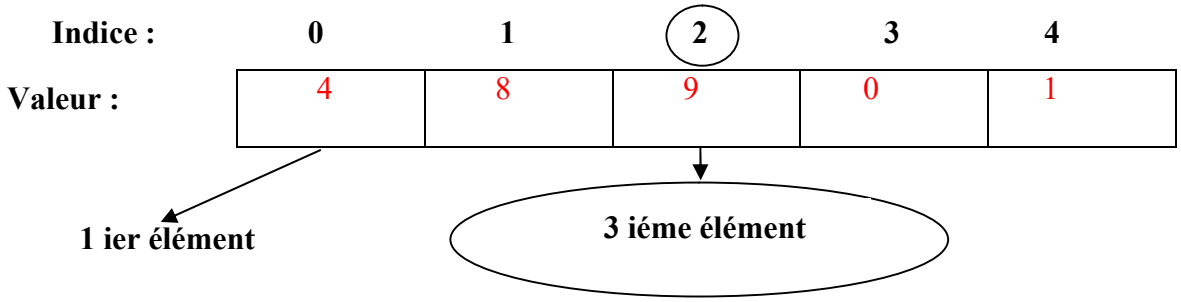


Figure V.1 Tableau à une dimension: notion d'indice

V.3.2 Déclaration

Toute déclaration est traduite par un ordre destiné à réserver de l'espace mémoire durant la session de travail. Les tableaux sont aussi des variables composées et indicées et doivent être déclarés. La syntaxe de déclaration est comme suit :

Syntaxe en algorithmique	Syntaxe en langage C
Var nom-tableau : **tableau** [dimension] **de type** ;	**Type** nom-tableau [dimension] ;

- **Exemple :** déclaration d'un tableau contenant 10 valeurs de type entier

Var T : **tableau** [10] d'**entier** ;	**int** nom-tableau [10]

V.3.3 Manipulation des éléments d'un tableau

La manipulation des éléments d'un tableau se fait via les indices, si notre tableau est de taille N alors:

- nom-tableau[i] désigne l'élément du tableau d'indice i
- les indices valides seront compris entre 0 et N -1

a. Initialisation d'un tableau

Il faudrait savoir que par défaut les tableaux sont vides, l'initialisation consiste à donner une valeur à chaque élément du tableau. Elle peut être séquentielle, sélective (1 élément bien particulier), ou via une lecture.

- **Initialisation séquentielle:** consiste à initialiser avec des valeurs prédéfinies tous les éléments du tableau lors de la déclaration
  - **Exemple :**

En algorithme	En langage C
Var tab[3]= {1,2,3};	**int** tab[3] = { 1, 2, 3 };

- **Initialisation sélective:** consiste à initialiser un, plusieurs même voir tous les éléments du tableau en spécifiant les indices et les valeurs en question.

En algorithme	En langage C
tab[0]← 1 ; tab[1]← 2 ; tab[2]← 3 ;	tab[0]=1 ; tab[1]=2 ; tab[2]=3 ;

b. Lecture des éléments d'un tableau

L'initialisation peut également être effectuée à travers une lecture des éléments saisis par l'utilisateur; il s'agit de répéter l'instruction de lecture selon le nombre des éléments dans le tableau, pour cela il faudrait utiliser les boucles, il est alors possible d'utiliser la boucle **Pour**, **Tant que** ou encore la boucle **Répéter**, le programmeur a en fait le choix. Dans la table V.1 ci-dessous une présentation des trois cas possibles.

Table V. 2 Instructions algorithmiques du remplissage d'un tableau T de N éléments entiers via les trois boucles

Boucle pour	Boucle tant que	Boucle répéter
Var T :**tableau** [N] d'**entiers** ; i : **entier** ; **Début** **Pour** i **de** 0 **à** N-1 **pas** 1 **faire** **Écrire (**"Donner un nombre entier"**) ;**	**Var** T :**tableau** [N] **d'entiers ;** i**: entier ;** **Début** i←0 ; **Tant que** (i<N) **faire** **Écrire (**"Donner un nombre entier"**) ;** **Lire**(T[i]) ;	**Var** T :**tableau**[N **d'entiers ;** i **:entier ;** **Début** i←0 ; **répéter** **Écrire (**"Donner un nombre entier"**) ;**

Lire (T[i]) ; **Finpour**	i←i+1 ; **Fin tant que**	**Lire**(T[i]) ; i←i+1 ; **jusqu'à (i=N)**

c. Affichage des éléments du tableau

On parle ici d'un affichage ou une écriture sur la console, on pourrait afficher des éléments particuliers ou bien tout le tableau

- **Exemple :** afficher le 2 ème élément d'un tableau T d'entiers

En algorithme	En langage c
Écrire (T[1]) ;	**printf(**"%d",T[1]**) ;**

Table V.2 Instructions algorithmiques d'affichage d'un tableau T de N entiers via les trois boucles

Boucle pour	Boucle tant que	Boucle répéter
Var T **: tableau** [N] **d'entiers;** i : **entier ;** **Début** **Pour** i **de** 0 **à** N-1 **pas** 1**faire** **Écrire** (T[i]) ; **FinPour**	**Var** T**: tableau** [N] **d'entiers ;** i**: entier ;** **Début** i←0 ; **Tant que** (i<N) **faire** **Écrire** (T[i]) ; i←i+1 ; **Fin tant que**	**Var** T **: tableau** [N] **d'entiers ;** i **: entier ;** **Début** i←0 ; **Répéter** **Écrire** (T[i]) ; i←i+1 ; **jusqu'à (i==N)**

- **Remarque:** avant de déclarer un tableau il faudrait d'abord avoir la taille exacte de ce tableau.

d. Recherche d'un élément dans un tableau

Le déplacement dans un tableau à une dimension se fait grâce à la notion d'indice. Pour rechercher un élément bien particulier, il sera question de parcourir tous les éléments et les comparer avec l'élément recherché.

- **Exemple :** Recherche d'un élément X dans un tableau T de taille n

Algorithme	Programme C

Pour i **de** 1 **à** n-1 **pas** 1 **faire**
Si (T[i]=X) **alors**
Écrire ("l'élément existe dans le tableau dans la position",i);
Finsi
Fin pour
	for (i=0;i<=n-1;i++)
	{
	if(T[i]==X)
	printf("l'element existe dans le tableau dan la position %d" , i) ;
	}

V.3.4 Tri d'un tableau

Trier un **tableau** consiste à ranger ses éléments en ordre croissant ou décroissant. Il existe différentes méthodes (algorithmes) permettant de réaliser le tri, telles que le tri à Bulles, tri par insertion, tri par fusion, tri par sélection, tri rapide, etc. Nous allons voir dans ensemble le une méthode tri, il s'agit du par sélection des minimums successifs (tri croissant) [8].

L'algorithme est itératif, chaque itération consiste à chercher le minimum (tris croissant) ou le maximum (tris décroissant), s'il existe une permutation est effectuée.

- **Exemple:** pour mieux comprendre voila un exemple, pour trier le tableau T via la méthode de la sélection des minimums successifs, on va avoir les boucles suivantes:

T= {12, 17, 9, 18, 7};

i=0 7 17 9 21 12 // pas de valeur plus petite que 7 alors pas d'échange

i=1 7 17 9 18 12 // permutation entre les valeurs 17 et 9

i=2 7 9 17 18 12 // permutation entre les valeurs 17 et 12

i=3 7 9 12 18 17 // permutation entre les valeurs 18 et 17

➢ **Résultat final:**

7 9 12 18 17 // pas de permutation, le dernier élément est trié automatiquement

Arrêt à l'indice i=N-2

- **Exemple :** réaliser le tri croissant des éléments d'un tableau de N valeurs du type entier par la méthode du tri par sélection.

Algorithme	Programme C
Algorithme Tri-selection **Var** n, i,j,posmin : **entier** ; T **: tableau** [N] **d'entiers ;** **Début** **Écrire** ("Donner la taille du tableau"); **Lire**(N);	main() { **int** N, i,j,min,posmin; **printf**("Donner la taille du tableau"); **scanf**("%d",&N); **int** t[N]; **for**(i=0;i<=N-1;i++)

```
  Redim (T) ;
//-----------------lecture de t----------------------
  Pour i de 1 à N-1 pas 1  faire
    Écrire ("Donner un entier");
    Lire (t[i]);
  Fin pour
 //---------------------Tri croissant de t ----------
  Pour i de 1 à N-2 pas 1  faire
    posmin←i ;
  Pour j de i+1 à N-1 pas 1  faire
    Si (t[j]< t[posmin] ) alors
        posmin←j ;
    Finsi
    Si (t[i]≠ t[posmin])// si (i ≠posmin) alors
                permut←t[i] ;
                t[i] ← t[posmin] ;
                t[posmin] ←permut ;
      Finsi
  Fin pour
  Pour i de 1 à N-1 pas 1  faire
      Écrire ("t [",i, "]=",t[i]);
   Fin pour
Fin algorithme
```

```
      {
          printf("Donner un entier");
          scanf("%d",&t[i]);
       }
   for(i=0;i<=N-2;i++)
       {
          min= t[i];
          posmin=i;// {12,17,9,18,7}
   for(j=i+1;j<=N-1;j++)
      {
          if(t[j]<min)
          {
                  min=t[j];
                  posmin=j;
           }
       }
           if (i!=posmin)
          {
                  t[posmin]=t[i];//permutation
                  t[i]=min;
           }
      }
 printf("Affichage du tableau  trie par ordre
croissant ");
 for(i=0;i<=N-1;i++)
      {
         printf("T[%d]=%d\n",i,t[i]);
      }
}
```

V.4 Les tableaux à deux dimensions

Un tableau à deux dimensions peut être assimilé aux matrices, il est constitué d'un ensemble de lignes **L** et de colonnes **C**. Comme les tableaux à une dimension, les éléments du tableau à deux dimensions doivent être du même type. Il est interprété comme un tableau de

taille L dont chaque élément est un tableau de dimension C; il **contient donc L*C éléments (**voir figure V.1**).**

L'accès à ces éléments se fait via l'indice de la colonne et l'indice de la ligne.

Mat[i][j] : désigne la ligne **d'indice i** et la colonne **d'indice j**.

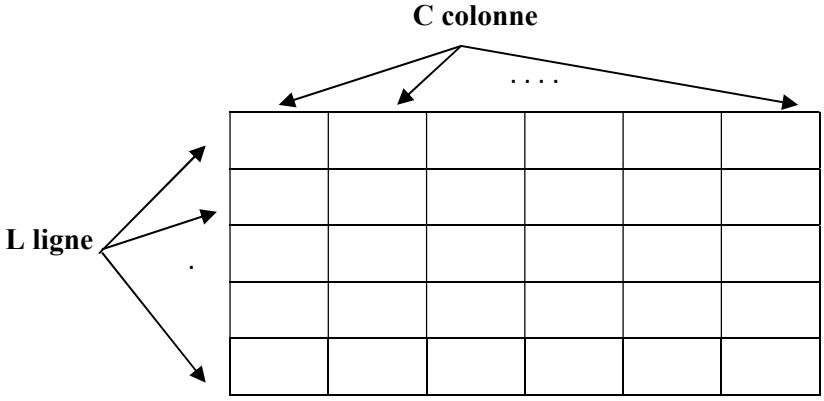


Figure V.2 Tableau à deux dimensions: notion de lignes et de colonnes

V.4.1 Déclaration

La déclaration d'un tableau doit être faite avant son utilisation dans le programme en c, et dans la partie des déclarations en algorithme. Ci-dessous une présentation générale de la syntaxe :

Syntaxe en algorithmique	Syntaxe en langage C
Var nom-tab [L][C] **tableau de Type ;**	**Type** nom-tab [L][C] ;

- **Exemple :** déclaration d'un tableau d'entiers de 10 lignes et 4 colonnes

Var mat [10][4] **tableau d'entiers ;**	int nom-tab [10][4] ;

V.4.2 Manipulation des tableaux à deux dimensions

a. Initialisation

Comme les tableaux à une dimension, l'initialisation des tableaux à deux dimensions peut être séquentielle ou sélective.

- **Initialisation séquentielle:** il s'agit d'initialiser les valeurs ligne par ligne
  - **Exemple:**

En algorithme	En programme C
var tab[3][2] ←{{1 , 2 }, {10 , 20 }, {100 , 200}};	**int** A [3][2] = {{1 , 2 }, {10 , 20 }, {100 , 200}};

- **Initialisation sélective :** il s'agit ici d'initialiser un ou plusieurs éléments bien particuliers, sachant que chaque élément est désigné par deux indices, il représente une intersection entre les lignes et les colonnes.

  - **Exemple :**

En algorithme	En programme C
tab[0][0] ←1 ; tab[0][1] ←2 ; tab[1][0] ←10 ;	tab[0][0]=1 ; tab[0][1]=2 ; tab[1][0]=10 ;

b. Lecture des éléments d'un tableau

L'initialisation peut être effectuée via une lecture des éléments saisis par l'utilisateur, pour cela, il faudrait utiliser les boucles. Nous avons alors besoin d'une boucle pour parcourir les lignes est une autre boucle pour parcourir les colonnes.

- **Exemple:** remplissage d'un tableau Mat à deux dimensions L*C contenant des réels

En algorithme	En langage C

Pour i **de** 0 **à** L **pas** 1 **faire** // parcourir les lignes
Pour j **de 0 à** C **pas** 1 **faire//** parcourir les colonnes
Écrire (" écrire donner un nombre réel ") ;
Lire (Mat[i][j]) ;
Finpour
Finpour
	for (i=0 ;i<L ;i++)
	for(j=0 ;j< C;j++)
	{
	printf (" écrire donner un nombre réel ") ;
	scanf("%f",Mat[i][j]) ;
	}

V.5 Allocation dynamique

L'allocation de la mémoire consiste à réserver de l'espace en mémoire pour une variable donnée durant la session d'exécution d'un programme. L'allocation peut être statique, où cet espace est réservé jusqu'à la fin de l'exécution, comme elle peut être dynamique et par conséquent l'espace est libéré à n'importe quel moment, une fois l'utilisation de la variable en question est terminée.

L'allocation statique se fait par défaut lors de la déclaration de la variable. Pour qu'elle soit dynamique, il faudrait utiliser des fonctions dédiées à ça, en langage C elles sont nombreuses, nous pouvons citer la fonction malloc (memory allocation) ou calloc pour l'allocation et la fonction free pour libérer cet espace alloué.

Or pour manipuler les tableaux de façons dynamiques, il faudrait utiliser un type de variable simple mais bien particulier appelé pointeur, et qui fait l'objet des cours et Tp durant semestre.

V.6 Les chaines de caractères

V.6.1 Définitions

Une chaîne est une séquence de caractères ASCII [1], elle est utilisée pour stocker du texte (mot, phrase, un ou plusieurs paragraphes). En langage C, contrairement à d'autres langages, il n'existe pas un type particulier appelé chaine, par conséquent les tableaux sont utilisés pour les mémoriser et les manipuler. Les chaînes caractères sont alors représentées sous la forme d'un tableau de caractères terminé par un octet nul (c'est-à-dire, un octet dont la valeur est égale à zéro, présenté sous la forme '\0').

Il faudrait noter toutefois, qu'en langage c, il existe plusieurs fonctions prédéfinies dans le fichier string.h, qui sont mises à la disposition du programmeur.

Une chaine est stockée dans un tableau à une dimension, comme c'est indiqué dans l'exemple suivant.

- **Exemple** : HELLO

H	E	L	L	O	\0

V.6.2 Déclaration et manipulation d'une variable du type chaine

a. Décalaration

En algorithme, il existe plusieurs façons de faire, la déclaration en algorithme comme en programme C suit la syntaxe suivante:

Syntaxe en algorithmique	Syntaxe en langage c
Var nom-ch : **chaîne**; Ou **Var** nom-ch : **chaîne [**nombre de car**]**; Ou **Var** nom-ch **tableau** [nombre de car] de **car**	**char** nom-ch[taille];

b. Initialisation

L'initialisation ne peut se faire que lors de la déclaration, selon la syntaxe suivante :

Syntaxe en algorithmique	Syntaxe en langage c
Var nom-ch **chaîne**[nb-car]← "Texte";	**char** nom-ch[nb-car]= " Texte "

- **Exemple**

Var mot **chaîne** [10] ← "Hello" On ne peux pas faire hors la déclaration ceci ~~mot ← "Hello"~~	**char mot[10]=** "Hello ~~**mot[10]=** "Hello"~~

c. L'accès aux éléments d'une chaine

Une chaine de caractères peut être interprétée comme un tableau à une dimension, l'accès à ses éléments se fait via l'indice comme indiqué ci-dessous :

H	E	L	L	O	\0
0	1	2	3	4	

- **Exemple :**

Var mot **chaine** [5]; mot[0] ←'H' ; mot[1] ←'E' ; mot[2] ←'L' ; mot[3] ←'L' ; mot[4] ←'O' ;	**char** mot [5] ; mot[0] ='H' ; mot[1] =E' ; mot[2] ='L' ; mot[3] ='L' ; mot[4] ='O' ;

d. Lecture des chaines de caractères

La lecture se fait de la même façon que les tableaux, en utilisant la fonction de lecture lire ou scanf(), sauf qu'on langage c, il est est possible de lire une chaine d'un coup, sans utiliser les boucles, en mettant le format adéquat %s. Mais il n'est malheureusement pas possible de lire une chaîne de caractère contenant des espaces avec la fonction scanf(), pour cela il serait plus judicieux de faire appel à un autre fonction prédéfinie gets(), que nous allons voir ultérieurement. La lecture d'une chaine de caractéres sans éspace via la fonction scanf() suit la syntaxe suivante ;

Syntaxe en algorithmique	**Syntaxe en langage c**
Lire (nom-char) ;	**scanf**("%s", nom-char) ;// pas de symbole &

- **Exemple : déclaration puis lecture**

Var mot chaine [10] ; **Écrire**("Donner une chaine de caractére"); **Lire**(mot) ;	**char** mot [10] ; **printf**("Donne une chaine de caractéres") ; **scanf**("%s", mot) ;

e. Affichage des chaines de caractères

La syntaxe d'affichage d'une chaine de caractère est comme suit :

Syntaxe en algorithmique	Syntaxe en programme c :
Écrire (nom-char) ;	**printf**("%s", nom-char) ;

- **Exemple : déclaration avec initialisation puis affichage**

Var mot chaine [10]←"Hello" ; **Écrire**(mot) ;	**char** mot [10] = "Hello" ; **printf**("%s",mot) ;

V.6.3 Les fonctions manipulant les chaines de caratères

Afin de manipuler les chaînes de caractères, il existe un ensemble de fonctions prédéfinies dans le langage C, et qui se trouvent dans la bibliothèque string.h , nous allons voir dans ce qui suit quelques une.

a. La taille d'une chaîne (nombre de caractères)

Pour récupérer la taille d'une chaîne, nous utilisons la fonction qui retourne le nombre de caractères de cette chaine. En algorithme la fonction longueur() et en langage c la fonction strlen.

Syntaxe en algorithmique	Syntaxe en programme c
Var L : **entiers;** L=**longueur**(chaine);	**int** L **;** L= **strlen** (chaine);

- **Exemple :**

L ←longueur(mot) ; **Écrire** ("voila la longueur du mot:%d",L);	L =strlen(mot); **printf**("voila la longeur du mot:%d",L);

Résultat : voila la longeur du mot : 5

b) Comparaison de deux chaines (strcmp)

Les chaînes peuvent être comparées en utilisant l'ordre lexicographique, en algorithme on utilise les opérateurs de comparaison; en langage c, la fonction **strcmp** permet de comparer deux chaines. **strcmp** retourne un nombre entier :

- Le nombre =0 si les deux mots sont identiques ;
- Le nombre< 0 si chaine1 < chaine2, c'est dire la chaine1 vien avant la chaine deux dans l'ordre alphabétique
- Le nombre >0 si chaine1 >chaine2

- **Remarque :** < ou > selon l'ordre lexicographique (voir table du code ascii).

La syntaxe de l'utilisation de la fonction de comparaison en algorithme et en programme c est comme suit :

Syntaxe en algorithmique	Syntaxe en programme c
chaine 1 **operateur** chaine 2 operateur est <,> ou =	**strcmp(**chaine1,chaine2**) ;**

- **Exemple 1:** comparaison entre les deux mots "allo" et "bllo", quelle valeur va être affichée ?

Syntaxe en algorithmique	Syntaxe en programme c

Algorithme comp **;**
Var i : **boolean** ;
mot1 **chaîne** [10] ←"allo";
mot2 **chaîne** [10] ←"bllo";
Début
i←(mot1=mot2) ;
Écrire(i) ;
Finalgorithme
	main()
	{
	char mot1[10]="allo";
	char mot2[10]="bllo";
	int i;
	i=strcmp(mot1, mot2);
	printf("%d",i);
	}

Résultat: affichage de faux	**Résultat:** affichage de la valeur - 1

- **Exemple 2 :** comparaison entre deux chaines de caractères initialisées et afficher s'ils sont égaux

Algorithme comparaison ;
Var val : booléen;
mot1 **chaîne** [10] ←" Sujet de test ";
mot2 **chaîne** [10] ←" Sujet de test ";
Début
val←(mot1=mot2) ;
Si (val =true) alors **Écrire** (" Les deux chaines identiques");
Sinon ("Les deux chaines sont differentes ") ;
Finsi
Finalgorithme
	main ()
	{
	char chaine1 [] = " Sujet de test ",
	chaine2 [] = " Sujet de test ";
	if (strcmp (chaine1 , chaine2) == 0)
	{ **printf** (" Les deux chaines sont identiques \n");
	}
	else
	{
	printf (" Les chaines sont differentes \n");
	}
	}

Résultat: affichage du message: Les deux chaines sont identiques

c) Copie d'une chaîne dans une autre

Copier le contenu d'une chaine dans une autre consiste à écraser cette dernière, en algorithme une simple affectation(←) permet de désigner la copie, hélas en langage c, ce n'est pas possible, il faudrait utiliser la fonction strcpy (string copy).

La syntaxe de l'utilisation de la fonction **strcpy** afin de copier le contenu de la chaine2 dans la chaine 1 est comme suit :

Syntaxe en algorithme	Syntaxe en programme c
chaine1 ←chaine2;	strcpy (chaine1 , chaine2);

- **Exemple:** copier le contenu de la chaine C1 dans la chaine C2 initialisée.

Algorithme copie-chaine ;	main ()
Var C1 **chaîne[ ]**← "Samedi"**;**	{
C2 **chaîne[]**← "Dimanche"**;**	**char** C1[]="Samedi";
Début	**char** C2[]=" Dimanche";
Écrire ("Avant copy :",C1) ;	**printf**("Avant copy :%s\n",C1);
//---affichage Samedi	**//----- affichage de Samedi**
C1← C2 ;	**strcpy**(C1,C2);
Écrire ("Apres copy ",C1) ;	**printf**("Apres copy :%s\n",C1);
//------ affichage Dimanche	**// affichage Dimanche**
Fin Algorithme	}

d) Concaténeation des chaines

Une chaîne de caractères peut être construite en lui ajoutant d'autres chaînes à la fin. La concaténation des chaines et le fait d'ajouter une chaîne à la suite d'une autre. En langage c, il existe une fonction qui permet ceci; il s'agit de la fonction **strcat.** En algorithme, il suffit d'utiliser l'opérateur + **.**

La concatenation d'une chaine 2 à la suite d'une autre chaine 1 suit la syntaxe algorithmique et en programme c illustrée ci-dessous:

Syntaxe en algorithme	Syntaxe En programme c
chaine1← chaine1 +chaine2	strcpy (chaine1 , chaine2);

- **Exemple**

Var C1 chaine[10] ←"Je suis" ;	main ()

C2 chaine[10] ←" un" ; C3 chaine[10] ←" " ; C4 chaine[10] ←"etudiant " ; C1←C1+C3+C2+C3+C4 ;	{ **char** C1[]="Je suis"; **char** C2[]=" un"; **char** C3[]=" "; **char** C4[]="etudiant"; **strcat**(C1,C3); **strcat**(C1,C2); **strcat**(C1,C3); **strcat**(C1,C4); }

Résultat : Je suis un etudiant

e) D'autres fonctions en langage C :

Ces fonctions font partie de la bibliothèque stdio.h

- getchar(): qui lit un caractère suite à la saisie au clavier (entrée standard)
  - **Exemple :**

```
char a ;
printf("donner un caractere");
a=getchar() ;
```

- putchar(c) : affichage d'un caractère à l'écran (sortie standard)
  - **Exemple :**

```
putchar(a) ;
```

- gets(**ch**): lit une ligne de caractères du clavier et la copie à l'adresse indiquée par **ch.**
  Elle est plus appropriée que la fonction scanf(format %s) lorsque la chaîne de caractères contient des espaces, car avec la fonction scanf la lecture s'arrête lors premier espace.
  - **Exemple :**

```
char chaine[10] ;
gets(chaine) ;
```

- puts (**ch**) : écrit la chaîne de caractères **ch** sur l'écran et provoque un retour à la ligne. Elle spécifique pour du texte, elle ne peut pas hélas afficher les valeurs d'autres types de variables comme la fonction printf.
  - **Exemple :**

```
char chaine[]= "Informatique" ;
puts(chaine) ;// affichage de la chaine Informatique
```

V.7 Exercices

1) Exercice 01 :

Q1. Ecrire un algorithme permettant de lire un tableau contenant **10** valeurs du type réel.

Q2. Rajouter les instructions permettant d'afficher les éléments du tableau.

Q3. Donner le programme correspondant.

2) Exercice 02:

Q1. Reprendre l'exercice précédent mais cette fois-ci en considérant que la taille du tableau doit être donnée par l'utilisateur

3) Exercice 03 :

Q1. Écrire un algorithme permettant de saisir les 8 notes des examens du S1 d'un étudiant en MI puis calculer sa moyenne et l'afficher ainsi que ces notes.

Q2. Donner le programme correspondant

4)Exercice 04:

Q1. Écrire un algorithme qui permet de saisir les moyennes 1000 étudiants inscrits en MI puis de déterminer combien d'entre eux ont la moyenne supérieures à la moyenne de la classe

Q2. Donner le programme corrspondant

5)Exercice 05 :

Q1. Ecrire un algorithme permettant de lire un tableau d'entiers contenant **3** lignes et **4** colonnes

Q2. Rajouter les instructions permettant d'afficher les éléments du tableau.

Q3. Donner le programme correspondant.

6) Exercice 06 :

Q1. Ecrire un algorithme permettant :

- de lire un tableau de L lignes et C colonnes contenant des valeurs réelles.
- d'afficher le tableau
- de chercher un nombre réel **A** saisi au clavier et dire si le nombre existe dans le tableau ou non, dans le cas où le nombre existe indiquer sa position (indice ligne et colonne).

Q2. Donner le programme correspondant en affichant le tableau suivant l'exemple :

12 4 5
2 5 18
. . .

7) Exercice 07 :

Q1. Ecrire un algorithme permettant :

- de lire un tableau Mat de L lignes et C colonnes contenant des valeurs du type entier.
- De calculer la moyenne de chaque ligne et la mettre dans un tableau T.
- D'afficher les tableaux Mat et T

Q2. Donner le programme correspondant.

8) Exercice 08 :

Ecrire un programme en c permettant de lire et d'afficher une chaine de caractères, sans avoir à utiliser les fonctions printf() et scanf() .

9) Exercice 09 :

Q1.Écrire un algorithme permettant de lire une chaine de caractères, l'algorithme devrait éliminer les caractères non alphabétiques dans la chaine puis l'afficher.

Exemple : "aB'3$ @kj " devient "aBkj

Q3. Rajouter dans votre algorithme les instructions permettant de calculer le nombre d'occurrences d'un caractère donné par l'utilisateur

Q2. Donner le programme correspondant, considérer la fonction **strchr** pour compter le nombre d'occurrence.

V.8 Corrigé type des exercices

1) Exercice 01 :

Q1 et Q2 :	Q3 :

Algorithme lect-aff –tab
Var T : **tableau** [10] **d'entiers** ;
I : **entiers ;**
Début
Pour i **de** 0 à 9 **pas** 1 **faire**
Écrire ("Donner un nombre entier"**) ;**
Lire (T[i]) ;
FinPour
Pour i **de** 0 à 9 **pas** 1 **faire**
Écrire (T[i]) ;
FinPour
	main()
	{
	int i ;
	float t[10];
	for(i=0;i<10;i++)
	{
	printf("Donner un nombre entier");
	scanf("%f",&t[i]);
	}
	for(i=0;i<10;i++)
	{

Finalgorithme	**printf**("T[%d]=%d\n",i,t[i]); } }

2) Exercice02 :

Q1 et Q2 :	Q3 :

Algorithme lect-aff –tab
Var T : **tableau** [] **d'entiers** ;
I,N: **entiers ;**
Début
Écrire ("Donner la taille du tableau"**) ;**
Lire (N) ;
RedimT(N) **;**
Pour i **de** 0 à N-1 **pas** 1 **faire**
Écrire ("Donner un nombre entier"**) ;**
Lire (T[i]) ;
FinPour
Pour i **de** 0 à N-1 **pas** 1 **faire**
Écrire (T[i]) ;
FinPour
Finalgorithme
	main()
	{
	int i,N;
	printf("Donner la taille du tableau");
	scanf("%d",&N);
	int t[N];
	for(i=0;i<=N-1;i++)
	{
	printf("Donner un nombre entier");
	scanf("%d",&t[i]);
	}
	for(i=0;i<=N-1;i++)
	{
	printf("T[%d]=%d\n",i,t[i]);
	}
	}

3) Exercice 03 :

Q1 :	Q2 :

Algorithme moyenne-etud
Var Note **tableau** [8] **réel** ;
Moy, S : **réel ;**
i: **entier** ;
Début
Pour i de 0 à7 pas 1 faire
Écrire ("Donner la note de la matière ",i+1**) ;**
Lire Note[i] ;
Finpour
	main()
	{
	int i ;
	float Note[8] ,S,Moy;
	for(i=0 ;i<=7 ;i++)
	{
	printf("Donner la note de la matière ") ;
	scanf("%f",&Note[i]) ;
	}

S←0 ;
Pour i **de** 0 **à**7 **pas** 1 **faire**
S←S+Note[i] ;
Finpour
Moy←S/8 ;
Écrire ("la moyenne de l'étudiant est", Moy**) ;**
Finalgorithme
	S=0 ;
	for(i=0 ;i<=7 ;i++)
	{
	S=S+Note[i] ;
	}
	Moy=S/8 ;
	printf("la moyenne de l'étudiant est%f",Moy) ;
	}

4) Exercice 04:

Q1 :	Q2 :

Algorithme saisie note
Var M : tableau[1000] **de réel** ;
i,nbr : **entier** ;
som ,nbr, moyc : réel ;
Début
som ←0 ;
Pour i **de** 0 **à** 999 **pas** 1 **faire**
Écrire("Donner la moyenne de l'etudiant", i+1) ;
Lire(M[i]) ;
som←som+M[i] ;
Fin pour
moyc←som/1000 ;
nbr←0 ;
Pour i **de** 0 **à** 999 **pas** 1 **faire**
Si(M[i]>moyc) **alors** nbr ←nbr+1 ;
Fin pour
Écrire("Le nombre des etudiants dont la moyenne est supérieur de la moyenne de classe est ",nbr) ;
Fin algorithme
	main()
	{
	float som,moyc, M[1000];
	int nbr,i;
	for(i=0;i<=999;i++)
	{
	printf("Donner la moyenne de l'etudiant %d", i+1) ;
	scanf("%f",&M[i]);
	}
	som=0;
	for(i=0;i<=999;i++)
	{
	som=som+M[i];
	}
	moyc=som/1000;
	nbr=0;
	for(i=0;i<=999;i++)
	{
	if (M[i]>moyc) nbr=nbr+1;
	}

	printf("la moyenne de la classe=%f",moyc); **printf**("Le nombre des etudiants dont la moyenne est supérieur de la moyenne de classe est %d ",nbr) ; }

5) Exercice 05 :

Q1:	Q2:

Algorithme lirmat
Var Mat **tableau[**3] [4] **d'entier** ;
I,j : **entier** ;
Début
Pour i **de** 0 **à** 3 pas **1 faire**
Pour j **de** 0 **à** 4 pas **1 faire**
Écrire (" donner une nombre entier ") ;
Lire(Mat[i][j]) ;
Finpour
Finpour
Pour i **de** 0 **à** 3 pas **1 faire**
Pour j **de** 0 **à** 4 pas **1 faire**
Écrire (Mat[i][j]) ;
Finpour
Finpour
Fin algorithme
	main
	{
	int i,j;
	float Mat [3][4];
	for(i=0;i<3;i++)
	{
	for(j=0;j<4;j++)
	{
	printf("Mat[%d][%d]=",i,j);
	scanf("%d",&Mat[i][j]);
	}
	}
	for(i=0;i<3;i++)
	{
	for(j=0;j<4;j++)
	{
	printf("Mat[%d][%d]=%d",i,j,Mat[i][j]);
	}
	}
	}

6) Exercice 06:

Q1:	Q2:
Algorithme Mat ; **Var** i,j,L,C,iL,iC ,trouv: **entier** ; Mat **tableau[L] [C] de réels** ;	main() { **int** i,j,L,C,iL,iC;

```
    A : réel ;
Début
 Écrire ("Donner le nombre de lignes et de
 colonnes");
 Lire (L, C) ;
 Redim(mat[L][C]) ;
 Pour i de 0 à L-1 pas 1 faire
   Pour j de 0 à C-1 pas 1 faire
    Écrire (" donner une nombre réel " ) ;
     Lire(Mat[i][j]) ;
   Finpour
 Finpour
 Pour i de 0 à L-1 pas 1 faire
     Pour j de 0 à C-1 pas 1 faire
        Écrire ( Mat[i][j] ) ;
      Finpour
Finpour
Écrire (" Donner le nombre réel") ;
Lire(A) ;
I=0 ; trouv=0 ;
Tant que(i<=L-1 et trouv=0 )
   Tant que(j<=C-1 et trouv=0)
     Si (Mat[i][j]=A)
       Trouv ←1 ;
       iL←i ;
       iC←j ;
     Finsi
     j←j+1 ;
  Fin tant que
   i=i+1 ;
Fin tant que
Si (trouv=1)
    Écrire ("Le nombre",A, "existe dans le
```

```
printf("donner le nombre de lignes et de
colonnes");
scanf("%d %d",&L,&C);
float A,Mat [L][C];
for(i=0;i<=L-1;i++)
 {
 for(j=0;j<=C-1;j++)
    {
     printf("Mat[%d][%d]=",i,j);
     scanf("%f",&Mat[i][j]);
      }
  }
  printf("donner un nombre réel");
  scanf("%f",&A);
  int trouv=0;
  i=0;
  while(i<=L-1 && trouv==0)
  {
    j=0;
    while(j<C-1 && trouv==0 )
     {
       if (Mat[i][j]==A)
           {
                 trouv=1;
                 iL=i;
                 iC=j;
           }
        j=j+1;
      }
  i=i+1;
  }
  for(i=0;i<=L-1;i++)
   {
```

tableau dans la ligne",iL, "et la colonne ",iC) ;
Sinon
Écrire (A," n'existe pas dans le tableau")
Finsi
Fin algorithme
	for(j=0;j<=C-1;j++)
	{
	printf("Mat[%d][%d]=%f",i,j,Mat[i][j]);
	}
	printf("\n");
	}
	i**f**(trouv==1)
	printf("%f existe dans le tableau , dans la ligne %d, et la colonne %d",A,iL,iC);
	else
	printf ("%f n'existe pas dans le tableau",A);
	}

7) Exercice 07 :

Q1.	**Q2.**

Algorithme som ligne-Mat ;
Var
L,C,i,j : **entier** ;
s : **réel ;**
M **tableau[L] [C] d'entiers** ;
T **tableau[L] [C] de réels** ;
Début
Écrire ("Donner le nombre de lignes et de colonnes");
Lire (L, C) ;
Redim(mat[L][C]) ;
Redim(T[L]) ;
Pour i **de** 0 **à** L pas 1 **faire**
Pour j **de** 0 **à** C pas 1 **faire**
Écrire (" donner une nombre entier**")** ;
Lire(Mat[i][j]) ;
Finpour
	main()
	{
	int L,C,i,j;
	float s;
	printf("donner la de ligne et le nombre de colonne");
	scanf("%d %d",&L,&C);
	int M[L][C];
	float T[L];
	for(i=0;i<=L-1;i++)
	for(j=0;j<=C-1;j++)
	{
	printf("M[%d][%d]",i,j);
	scanf("%d",&M[i][j]);
	}
	for(i=0;i<=L-1;i++)
	{

Pour i **de** 0 **à** L-1 pas 1 **faire**
s←0 ;
Pour j **de** 0 **à** C -1pas 1 **faire**
s←s+M[i][j];
Finpour
T[i] ←s/C ;
Finpour
Pour i **de** 0 **à** L-1 pas 1 **faire**
Écrire (T[i]) ;
Finpour
Fin algorithme ;
	s=0;
	for(j=0;j<=C-1;j++)
	s=s+M[i][j];
	T[i]=s/C;
	}
	for(i=0;i<=L-1;i++)
	printf("T[%d]=%f\n",i,T[i]);
	}

8) Exercice 08 :

```
main()
{
   char c;
   char ch[100];
   puts("Donner une chaine");
   gets(ch);
   puts("la chaine est ");
   puts(ch);
   puts("Donner un caractere");
   c=getchar();
   puts("le caractère est");
   putchar(c);
}
```

09) Exercice 09 :

Q1 et Q2	Q3
Algorithme trait-chaine ; **Var** Ch **chaine** [300] ; Cara : **car** ; L,nb,i,j : **entier** ;	#include<string.h> main () { **char** chaine[100]; **puts**("donner une chaine de caractères");

```
Début
  Écrire ("donner une chaine de caractères" );
  L←longeur(Ch) ;
  Lire (Ch) ;
  i←0 ;
  Tant que( i <=L-1)
    Si ((Ch[i]>='a' et Ch[i] <='z') ou( Ch[i]>='A' et
    Ch[i] <='Z')) alors
     i←i+1 ;
    Sinon
      Pour j de i à L-2 pas 1 faire
         Ch[j]←Ch[j+1] ;
      Finpour ;
      Ch[j+1] ← "/0" ;// vider les cases en plus
                        // suite au déclage
     L←L-1 ;
    Finsi
  Fin Tant que
//Q2
Écrire("Donner un caractère") ;
Lire(cara) ;
nb←0 ;
Pour i de 0 à L-1 pas 1 faire
      si(Ch[i]=cara )alors nb←nb+1;
      Finsi ;
 FinPour ;
Écrire ("Le nombre d'occurrence du caractère",
cara, "est :",nb) ;
Finalgorithme
```

```
  gets(chaine);
  int L,i,j;
  L=strlen(chaine);
  i=0;
  while(i<L)
    {
      if ((chaine[i]>='a' && chaine[i]<='z') ||
      (chaine[i]>='A' &&chaine[i]<='Z'))
         {
         i=i+1;
         }
       else
        {
        for(j=i;j<=L-2;j++)
             chaine[j]=chaine[j+1];
              chaine[L-1]="/0";
              L=L-1;
         }
    }
 puts(chaine); //q2
 char *suit,cara;
printf("donner un caractere");
scanf("%c",&cara);
suit=strchr(chaine,cara);
int nb=0;
while( suit!=NULL)
    {
     nb++;
     suit=strchr(suit+1,cara);
     }
printf("le nombre de caractere est %d",nb)
}
```

V. 9 Conclusion

Nous avons pu voir durant ce chapitre un type de variables composé et indicé, il s'agit des tableaux à une et à deux dimensions, l'utilisation de ce type de variable est sollicitée lorsque nous avons plusieurs données concernant une entité (une dimension) ou même plusieurs (deux dimensions), à condition que ces données soient du même type (tous des entiers, tous des réels, tous des caractères,…). Il est possible d'utiliser une déclaration statique, ou l'espace mémoire occupé par le tableau, n'est libéré que lorsque le programme a terminé son exécution. Une initialisation dynamique permet justement de libérer cet espace mémoire une fois l'utilisation du tableau est terminée, qui peut être bien avant la fin du programme, permettant ainsi une meilleure gestion de l'espace mémoire.

Dans un certain type de problème, les données concernant une même entité peuvent être de types différents, pour cette raison l'utilisation des tableaux n'est plus possible, or il existe un moyen de personnaliser sa structure donnée pour aboutir à une bonne résolution du problème, ceci fera l'objet du chapitre suivant.

CHAPITRE VI

Les types personnalisés

Plan du chapitre :

VI.1 Introduction

Nous avons pu voir durant les chapitres précédents, qu'il existe deux grandes catégories de variables. La première, les types simples, comprend les entiers (longs, courts, ...), les réels (long, double, ...), les caractères ainsi que les pointeurs qui feront l'objet des cours du semestre 2. La seconde porte sur les types composés, plus particulièrement les tableaux à une et à deux dimensions, et l'intérêt d'utiliser ce type de variables dites indicées, sollicités lorsque nous avons plusieurs données du même type. Malheureusement, dans certaines situations, ces données peuvent être de types différents, telle que les données concernant une voiture par exemple. Cette dernière possède un numéro d'immatriculation de type entier, une marque et une couleur qui sont de type chaine de caractères. Aussi, d'autres problèmes nécessitent d'avoir une liste de valeur pour vérifier la conformité des données. Dans ces situations-là, les types de variables vues précédemment, ne suffisent pas à définir ce type de données. Il est par conséquent nécessaire de faire appel à des structures de données personnalisées telles que les enregistrements et les types énumérés, le sujet principal de ce chapitre.

VI.2 Les types énumérés

Un **type énuméré** est un type permettant de représenter des objets pouvant prendre leur valeur dans **une liste finie** et ordonnée de noms.

- **Exemple** :

  Semaine= **(**lundi**,** mardi**,** mercredi**,** jeudi, vendredi**,** samedi**,** dimanche**)**

  Couleur= **(**rouge**,** vert**,** bleu**)**

VI.2.1 Manipulation

a. Déclaration d'un type énuméré (définition)

La définition d'un nouveau type, permet d'enrichir les types de variables. Nous avons justement l'habitude de déclarer des variables en utilisant des types déjà prédéfinis (entier, réel, booléen,), il s'agit maintenant de voir comment créer son propre type personnalisé. Cette définition en algorithmique doit être effectuée avant la déclaration des variables, donc avant le mot clés **var,** et en dehors de la fonction main, juste âpres les directives en langage C.

La syntaxe de la définition d'un nouveau type en algorithmique, comme en langage C se fait comme suit :

Syntaxe en algorithmique	**Syntaxe en langage C**
Type nom-type= (val1,… valeur n) ;	**typedef enum** {valeur1,…., valeur2} nom-type;

- **Exemple:**

Type SEMAINE**=** **(**lundi**,** mardi**,** mercredi**,** jeudi, vendredi**,** samedi**,** dimanche**) ;**	**typedefenum{**lundi**,** mardi**,** mercredi**,** jeudi, vendredi**,** samedi**,** dimanche**)}** SEMAINE **;**

b. Déclaration d'une variable du type énuméré défini

Une fois le type défini, il est possible maintenant de déclarer la variable, ceci peut être réalisé comme suit :

Syntaxe en algorithmique	Syntaxe en langage C
Var nom-var:**nom-type**;	**nom-type**nom-var;

- **Exemple:**

Var S1,S2 **:** SEMAINE ;	SEMAINE S1, S2 **;**

- **Remarque :**

  En langage C, il est possible de considérer une autre notation lors de la définition d'un type et par conséquent aussi lors de la déclaration d'une variable de ce type-là. Ci-dessous la syntaxe:

  enum nom-type {val1, val2, …,valn} ;

  enum nom-type nom-var ;

- **Exemple:**

  enum SEMAINE jour ;

c. Affectation dans les variables de type énuméré

L'affectation est une instruction permettant de stocker une valeur dans une variable, or il est possible d'affecter une valeur dans une variable de type énuméré, cette valeur doit figurer dans la liste de valeur déclarée, en réalité elle correspond à un rang dans la liste, qui est de type entier.

Syntaxe en algorithmique	Syntaxe en langage C
nom-var ← valeur;// parmi ceux déclarés	nom-var = valeur

- **Exemple:**

jour←samedi; **correcte**	jour =samedi; **correcte**
jour←x; **incorrect;**	jour =x; **incorrect;**

VI.2.2 Caractéristiques des types énumérés

Les types énumérés ont certaines caractéristiques nécessaires à leur manipulation :

1) Une variable de type énuméré ne peut prendre pour valeur que l'une des étiquettes associées au type.

2) Relation d'ordre : l'ordre dans lequel les valeurs sont listées a une importance

- Les étiquètes sont affectées par défaut aux éléments du type énuméré déclaré
- Chaque élément est représenté par une valeur entière selon son ordre dans la liste (de gauche à droite)
- Les étiquettes commencent par défaut par 0, elles peuvent être modifiées par le programmeur.

▪ **Exemple:**

enum mois {JANVIER, FÉVRIER,…, DÉCEMBRE} ;

➢ Cette définition du type énuméré implique que :

JANVIER=0< FÉVRIER=1< …<DÉCEMBRE=12.

➢ On peut forcer un élément d'un type à prendre une valeur donnée grâce au signe = suivi d'une constante entière)

▪ **Exemple:**

enum jour {lundi, mardi, **mercredi = 20**, jeudi, vendredi, samedi, dimanche} ;

3) Les éléments de la liste ne peuvent être lus (ils sont définis)

4) Pour afficher les éléments de la liste, il faut prévoir un sous-algorithme (sous-programme)

5) Pour récupérer les éléments de la liste sous formes de chaine de caractères, il faut aussi prévoir un sous-algorithme (sous-programme)

VI.3 Les enregistrements

Les enregistrements sont des structures de données permettant de regrouper plusieurs variables qui sont de types différents et qui définissent une seule entité ou objet. Un enregistrement est donc caractérisé par un nom qui doit être significatif à l'objet qu'il représente, et un ensemble de variables qui le définit, appelé champs [9].

▪ **Exemple :** un enregistrement représentant **une personne (nom de l'enregistrement),** il peut être caractérisée par :

➢ Un nom: type caractère;

➢ Un prénom: type caractère;

➢ Age: entier;

➢ Adresse : (type chaine de caractères)

VI.3.1 Déclaration d'un type (définition)

Comme les types énumérés, les enregistrements sont des types personnalisés, ils doivent alors être déclarés. En langage C, la déclaration doit être à l'extérieur de toute fonction, y compris la fonction principale, juste après les directives. En algorithme, avant la déclaration des variables, juste après les types énuméré s'ils existent.

Ci-dessous les deux syntaxes, en algorithmique et en langage C.

Syntaxe en algorithmique	Syntaxe en langage C
Nom_enregistrement**: Enregistrement** Nom_champ1 **: Type -champ1;** Nom_champ2 **: Type -champ2;** **…** Nom_champn **: Type -champ n;** **Fin Enregistrement;**	**typedef struct** nom-enregistrement **{** **type_champ1** nom_champ1**;** **type_champ2** nom_champ2**;** **…** **};**

▪ **Exemple :** déclaration de l'enregistrement personne caractérisé par le nom, le prénom, l'âge et l'adresse:

Syntaxe en algorithmique	Syntaxe en langage C
Personne: **Enregistrement** nom : **Chaine;** prenom : **Chaine;** age : **réel;** adresse : **chaine** ; **Fin enregistrement ;** **Var** **…..**	**typedef struct** **{** **char** nom[20]; **char** prenom [30]**;** **float** age**;** **char** adresse[200] **;** **}** Personne**;** **main()** **{** **….** **}**

VI. 3.2 Déclaration d'une variable de type enregistrement

Une fois le type enregistrement est créé, il est maintenant possible de déclarer une ou plusieurs variables de ce type-là. Chaque nouvelle variable déclarée aura la même structure de l'enregistrement, c'est-à-dire elle va avoir les mêmes champs (voir figure 6.1).

La syntaxe de déclaration est comme suit :

Syntaxe en algorithmique	Syntaxe en langage C
Var nom-var **: Nom-enregistrement**	**nom-enregistrement** nom-var ;

- **Exemple:** déclaration de deux variables du type Personne déclarées dans l'exemple précédent :

Var P1 P2:**Personne;**	**Personne** P1, P2**;**

Suite à cette déclaration, un espace mémoire est réservé pour chaque champ de cette variable de type **Personne** comme illustré dans la figure V.1**.**

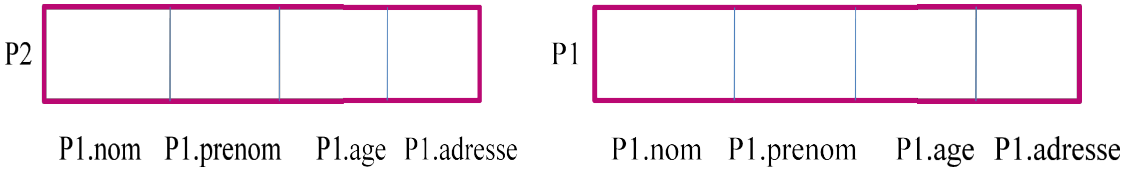


Figure V.1 Représentation des champs des variables P1 et P2 de type Personne

- **Remarque :**

  Il existe une autre notation en langage C qui consiste à ne pas utiliser le mot clé **typedef**, dans la déclaration du type et par conséquent la déclaration de la variable doit être précédé par le mot clés **struct,** comme illustré par l'exemple suivant :

- **Exemple** :

```
struct Personne
    {  char nom[50];
       char prenom [30];
       float age;
       char adresse[200] ;
    };
    main ()
    { struct Personne p1,p2; ….
```

VI. 3.3 Manipulation des variables de type enregistrement

Pour manipuler les variables du type enregistrement, il faudrait connaitre les champs de cet enregistrement et comment accéder à ces champs-là. L'accès, justement se fait en utilisant la notation **nom-enregistrement.nom-champ.**

a. Lecture

Récupérer les valeurs saisies par l'utilisateur, dans les champs adéquats de la variable de type enregistrement nécessite l'utilisation de la fonction de lecture. Puisque nous somme dans le dernier chapitre, à ce niveau on rappelle qu'il est question d'interaction avec l'utilisateur, par

conséquent cette instruction de lecture est généralement précédé par un message qui doit être affiché à l'écran, afin de demander à l'utilisateur de saisir les données en question .

- **Exemple:** lecture d'une variable de type Personne en entier (remplissage de tous les champs)

Syntaxe en algorithmique	Syntaxe langage C
Var p:**Personne;** **Écrire**("donner le nom de la personne "); **Lire** (p.nom) ; **Écrire**("donner le prénom de la personne "); **Lire** (p.prenom) ; **Écrire**("donner l'agede la personne "); **Lire** (p.age) ; **Écrire** ("donner l'adresse de la personne "); **Lire**(p.adresse) ;	**Personne** p1**;** **printf**("donner le nom de la personne "); **scanf**("%s",&p.nom) ; **printf**("donner le prenom de la personne "); **scanf**("%s",&p.prenom) ; **printf**("donner l'agede la personne "); **scanf**("%f",&p.age) ; **printf**("donner l'adresse de la personne "); **scanf**("%s",&p.adresse) ;

b. Affichage (écriture)

L'affichage d'une variable de type enregistrement consiste à afficher tous ces champs, il est notamment possible d'afficher un ou plusieurs de ses champs particuliers. Il se fait via l'instruction d'écriture telle qu'elle est illustrée dans l'exemple suivant :

- **Exemple:** affichage du champ nom d'une variable du type de l'enregistrement Personne, déclaré dans un exemple précédent :

Syntaxe en algorithmique	Syntaxe en langage C
Var p:**Personne;** **Début** **Écrire** ("Le nom de la personne : ", p.nom);	**Personne** p**;** **main(){** **printf**("le nom de la personne:%s",p.nom) ;

c. Initialisation

Il est possible d'initialiser une variable de type enregistrement, cette initialisation ce fait lors de la déclaration, voici un **exemple** :

Var p:**Personne=(**"Benzi","Sarah", 38, "Toto"**);**	**Personne** p= {"Benzi","Sarah", 38, "Toto"} ;

d. Plusieurs variables du même type d'enregistrement

Dans le cas ou nous avons un nombre assez important de variables du même type enregistrement, tel que l'exemple des employés d'une entreprise, les élèves d'une classe ou même différentes voitures, il sera plus judicieux de faire appel à la structure des tableaux, puisque il s'agit de plusieurs variables du même type. L'utilisation des boucles et la notion d'indice permettront d'automatiser les différentes manipulations, à savoir la lecture, l'écriture, ainsi que d'autres traitements possibles. Mais il est notamment nécessaire de comprendre comment se fait l'accès aux champs de chaque variable qui constituera un élément du tableau (voir figure VI.2).

- **Exemple :** 30 personnes, donc 30 variables de type Personne

Var TPer: **tableau** [30] de **Personne;**	**Personne** TPer[30]**;**

L'accès à un champ d'une variable enregistrement stockée dans un tableau se fait en spécifiant l'indice puis le champ en question

- **Exemple:** afficher l'âge de la première personne

Écrire("l'age de la premiere personne est : ", TPer[0].age**) ;**	**printf ("l'age de la premiere personne est :%f",**TPer[0].age**) ;**

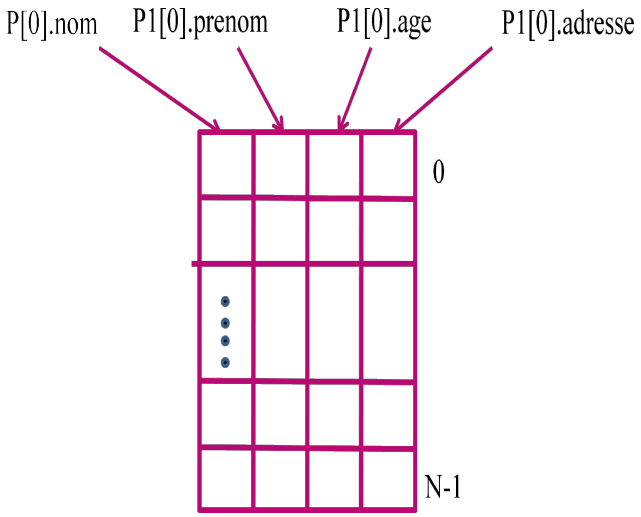


Figure VI.2 représentation par un tableau de N enregistrements de type personne

VI.4 Exercices

1) Exercice 1:

Q1. Déclarer en algorithmique puis en langage C un type permettant de représenter les valeurs vrai ou faux.

Q2. Écrire un algorithme puis le programme C qui permet d'initialiser une variable avec la valeur vrai puis de l'afficher

2) Exercice 2 :

Q1. Déclarer en C un type permettant de représenter la météo, caractérisé par ces états soleil, pluie et neige.

Q2. Écrire le programme qui permet d'afficher les éléments de la liste en chainecaractères.

Q3. Rajouter les instructions permettant de récupérer la météo dujour en chaine de caractère et d'afficher sa valeur en une variable du type météo.

3) Exercice 3:

Q1.Écrire un algorithme qui permet la saisie des données concernant deux personnes pers1 et pers2, puis affiche la différence d'âge entre ces deux personnes

4) Exercice 4 :

En langage C, il n'y a pas de type prédéfini permettant de manipuler des nombres complexes. On souhaite créer un type Complexe en utilisant une structure avec deux champs de type double qui représentent la partie réelle et la partie imaginaire. On rappelle qu'un nombre complexe **z** est défini par ses parties réelle **a** et imaginaire **b**.

z = **a** + i**b**

Q1. Définir le type Complexe en C.

Q2. Écrire un sous-programme qui permet de

- De lire un nombre complexe.
- Puis affiche ce nombre complexe.

5) Exercice 5 :

Q1. Définir en algorithmique un enregistrement (structure) permettant de représenter des étudiants caractérisés par leurs noms, leurs prénoms, leurs dates de naissance et leurs états.
Sachant que l'état d'un étudiant peut être : nouveau, répétitif ou endetté.

Q2. Écrire un l'algorithme permettant de saisir les informations de N étudiants avec N donné par l'utilisateur.

Q3. Rajouter des instructions permettant de séparer les étudiants en trois catégories selon leurs états, puis les afficher par catégories.

Q4.Écrire le programme C équivalent à cet algorithme

VI.5 Corrigé type des exercices:

1) Exercice 1:

Algorithme	Programme

Algorithme vrai-faux ;
Type booleen= (faux, vrai);
Var b :**booleen ;**
Debut
b ←vrai ;
Si(b=1) **alors**
écrire(" vrai");
Sinon
écrire ("faux");
Finsi ;
Fin algorithme
	typedef enum {faux, vrai}booleen;
	main()
	{
	booleen b;
	b = vrai;
	if(b==1)
	printf(" vrai");
	else
	printf(" faux");
	}

2) Exercice 2 :

Q1.Q2.

```
# include<stdio.h>
# include<stdbool.h>
typedefenum {soleil,pluie,neige}meteo;
main()
{
    int i;
    meteo   meteoj;
    char mj[6];
    for(i=0;i<3;i++)
        {
          if (i==0) printf("soleil");
          if (i==1) printf("pluie");
          if (i==2) printf("neige");
        }
    bool correct=false;
```

```
        while(correct==false)
            {
        printf("Donner la météo du jour soleil, pluie ou neige");
        scanf("%s",&mj);
        if (strcmp(mj,"soleil")==0) {meteoj=soleil; correct=true;}
        else if (strcmp(mj,"pluie")==0) { meteoj=pluie; correct=true;}
        else if (strcmp(mj,"neige")==0) { meteoj=neige; correct=true;}
        else ("erreur");
        }
        printf("meteo du jour est %d",meteoj);
}
```

Q3.

```
#include <stdio.h>
#include <conio.h>
typedefenum {dimanche,lundi,mardi, mercredi, jeudi,vendredi, samedi} jour;
main()
{
    float temp[7];
    int min,max,j;
    /* *********Saisie des temperatures**********************/
    printf("Releve des temperatures de la semaine \n");
    for(int j=0;j<7;j++)
        {
          printf("Quelle est la temperature du");
          if (j == 0) printf("dimanche");
          if (j == 1) printf("lundi");
          if (j == 2) printf("mardi");
          if (j == 3) printf("mercredi");
          if (j == 4) printf("jeudi");
          if (j == 5) printf("vendredi");
          if (j == 6) printf("samedi");
          printf(" : ");
          scanf("%f", &temp[j]);
```

```
        }
min = 0;
max =0;
   /*chercher le jour le plus froid et le jour plus chaud*/
for (j =0 ; j<7  ; j++)
   {
      if (temp[j] < temp[min]) min = j;
      if (temp[j] > temp[max]) max = j;
   }
//-------------------------jour et température du jour le plus froid-------------------------------
printf("\n Le jour le plus froide de la semaine a été constaté le ");
if (min== 0) printf("dimanche");
if (min == 1) printf("lundi");
if (min == 2) printf("mardi");
if (min== 3) printf("mercredi");
if (min == 4) printf("jeudi");
if (min == 5) printf("vendredi");
if (min== 6) printf("samedi");
printf(". \n d'une température  de %3.1fdegres C.", temp[min]);
//-------------------------jour et température du jour le plus froid-------------------------------
printf("\nLe jour le plus chaud de la semaine a été constaté le ");
if (max== 0) printf("dimanche");
if (max == 1) printf("lundi");
if (max == 2) printf("mardi");
if (max== 3) printf("mercredi");
if (max == 4) printf("jeudi");
if (max == 5) printf("vendredi");
if (max== 6) printf("samedi");
printf(". \n d'une température  de %3.1fdegres C.\n", temp[max]);
getch();
}
```

3) Exercice 3 :

Q1.Algorithme principale;

pers: **enregistrement**

```
        nom:char[20];
        prenom:char[20];
        age:entier;
    Fin enregistrement
    Var p1,p2: perso;
    diff:entier;
    Début
// ----------saisi des données de la première personne----------
    Écrire ("Donner le nom de la première personne ");
    Lire (p1.nom);
    Écrire ("Donner le prénom de la première personne");
    lire(p1.prenom);
    Écrire ("Donner l'âge de la première personne");
    lire(p1.age);
    // ----------saisi des données de la deuxième personne----------
    Écrire ("Donner le nom de la deuxième personne ");
    lire(p2.nom);
    Écrire ("Donner le prénom de la deuxième personne");
    lire(p2.prenom);
    Écrire ("donner l'âge de la deuxième personne"); lire(p2.age);
    diff ←p1.age-p2.age;
    Si (diff<0) alors diff←-diff;
    Écrire("La différence d'âge entre les deux personnes est:",diff);
Fin algorithme
```

Q2 .

```
    struct pers {
        char    nom[20];
        char    prenom [30];
        float   age;
        char    adresse [200]
                };
    main()
        {
```

```
        struct pers p1,p2;
        float diff;
        printf("donner le nom de la première personne ");  scanf("%s",&p1.nom);
        printf("donner le prénom de la première personne");  scanf("%s",&p1.prenom);
        printf("donner l'âge de la première personne"); scanf("%f",&p1.age);
        printf("donner le nom de la deuxième personne ");  scanf("%s",&p2.nom);
        printf("donner le prénom de la deuxième personne");  scanf("%s",&p2.prenom);
        printf("donner l'âge de la deuxième personne"); scanf("%f",&p2.age);
        diff= p1.age-p2.age;
        if (diff<0) diff=-diff;
        printf("la différenced'âge entre les deux personnes est:%f",diff);
       getch();
      }
```

4) Exercice 4 :

```
#include<stdio.h>
#include<conio.h>
struct complexe
   {
        doublereel;     /* partie reelle */
        doubleimag;  /* partie imaginaire */
   };
main(){
 struct         complexe C1, C2;
                printf("Saisie du premier complexe : ");
                printf("\nReel: ");
                scanf("%lf", &C1.reel);
                printf("Imaginaire: ");
                scanf("%lf", &C1.imag);
                printf("Saisie du second complexe : ");
                printf("\nReel: ");
                scanf("%lf", &C2.reel);
                printf("Imaginaire: ");
                scanf("%lf", &C2.imag);
                printf("Affichage premier complexe : ");
```

```
        printf("%lf + %lf i", C1.reel, C1.imag);
        printf("\nAffichage second complexe : ");
        printf("%lf + %lf i", C2.reel, C2.imag);
    }
```

5) Exercice 5 :

Q1,Q2, et Q3

Algorithme principale;

```
Type etat=(NOUVEAU , REPETITIF , ENDETTE) ;
Date enregistrement
    jj : entier ;
    mm :entier ;
    aa : entier ;
Fin enregistrement ;
//-------------------------------------------------------------------------------------------------------
étudiant: enregistrement
    nom: chaine;
    prenom: chaine;
    dn :date;
    et :etat ;
Fin enregistrement ;
//-------------------------------------------------------------------------------------------------------
Var  E,Tab.nv,Tab.rep,Tab.det : tableau [N] de etudiant;
     N,NV,RP,DET,i :entier ;
Début
Écrire("donner le nombre des étudiants ");
Lire (N);
//---------------- saisir les données des étudiants : remplissage du tableau des enregistrements
Pour i de 0 à N-1 pas de1 faire
    Écrire ("donner respectivement  le nom, le prénom, la date de naissance en jj/mm/aa
et l'état  de l'étudiant",i+1);
    lire( E[i].nom,  E[i].prénom, E[i].dn.jj, , E[i].dn.mm, , E[i].dn.aa, , E[i].et);
Fin pours ;
//-------- ----------séparer les données en trois catégories----------
```

RP←0 ;NV←0 ;DET←0 ;

Pour i **de** 0 **à** N-1 **par pas de**1 **faire**

Si (E[i].et=0) **alors** tabnv[NV]←E[i] ; NV=NV+1 ;

Sinon si (E[i].et=1) **alors** tabrp[RP]←E[i] ; RP=RP+1 ;

Sinion si (E[i].et=2) **alors** tabdet[DET]←E[i] ;DET=DET+1 ;

Finsi ;

Finsi ;

Finsi ;

Finpour ;

Écrire ("le nombre des etudiants nouveaux", NV ,"le nombre des etudiantsrepetitifs",RP," le nombre des etudiantsendette",DET);

Finalgoithme

Q4.

```
#include<stdio.h>
 #include<conio.h>
enumetat {NOUVEAU , REPETITIF , ENDETTE};
struct  date
    {
     int jj;
     int mm;
     int aa;
     };
struct etudiant
    { char nom[30];
      char prenom[30];
      struct date dn ;
     };
main()
{
 int N,NV,RP,DET,i;
 printf("Donner le nombre d'etudiant");
```

```
scanf("%d",&N);
structetudiant E[N], Tabnv[N],Tabrep[N],Tabdet[N];
//------ saisir les données des étudiants : remplissage du tableau des enregistrements ---
for(i=0;i<n;i++)
{
 printf("donner  respectivement le nom, le prénom et la date de naissance en jj/mm/aa
 et l'étatde l'etudiant %d",i+1);
 scanf("%s%s%d%d%d%d",&E[i].nom,&E[i].prenom,&E[i].dn.jj,&T[i].dn.mm,&E[i].dn.aa,&
 E[i].et);
 }
//----------------------séparer les données en trois catégories----------------------------------------
NV=0; RP=0; DET=0;
for(i=0;i<n;i++)
   {  if(E[i].et==0)
          {
               tabnv[NV]=E[i];
               NV=NV+1;
          }
      else if(E[i].et==1)
          {
               tabrp[RP]=E[i];
               RP=RP+1;
          }
        else if(E[i].et==2)
         {
               tabdet[DET]=E[i];
               DET=DET+1;
          }
    }
printf("le nombre des étudiants  nouveaux=%d \n le nombre des étudiantsrepetitifs=%d\n,
le nombre des étudiants endette=%d",NV,RP,DET);
}
```

VI.6 Conclusion

Avec ce chapitre nous cloturons ce cours, nous avons présenté les quelques structures de données de base à partir de simples variables telles que les entiers, les réels , les caracteres et les booléens, puis les types composés, plus précisément les tableaux à une et à deux dimensions ainsi que les chaines de caractères, jusqu'aux types personnalisés à savoir les types énumérés et les enregistrements, sujet principale de ce chapitre. Les types personnalisés sont utilisés pour justement créer de nouveaux types selon les besoins du programmeur, alors que les enregistrement permettent de representer des objets (entités) ayant plusieurs informations de type différents. Lorsque nous avons plusieurs enregistrements, il faudrait penser à profiter pleinement des structures répétitives et des vraiables indicées, effectivement l'utilisation des tableaux d'enregistrements permettra de faciliter la manipulation d'un grand nombre d'enregistrements.

Conclusion générale

Ce polycopié de cours contient des concepts de base pour commencer l'algorithmique et la programmation en c , il est dédié aux etudiants du L1, du socle commun mathématiques, mathématiques appliquées et Informatique, il est conforme au programme de la matière algorithmique et structure de donnée du semestre 1. Il peut notamment être utilisé par tout étudiant ayant besoin de ces notions la, spécialement ceux des domaines ST et SM. Une présentation en algorithmique ainsi qu'on langage C permet aux étudiants de leur faciliter l'apprentissage d'autres langages par la suite.

Dans ce polycopié, nous avons introduit dans un premier temps, le principe de fonctionnement d'un ordinateur, la notion de langage machine ainsi que la chaine de réalisation et d'exécution d'un programme à partir de l'étape de l'analyse du problème, l'écriture de l'algorithme jusqu'à la compilation et l'exécution.

Nous avons vu par la suite les différents types de données, constantes et variables simples, qui sont essentiels dans n'importe quel algorithme ou programme (sauf le cas d'un algorithme d'affichage), nous avons également expliqué le lien entre ces données et la mémoire.

Il était cependant indispensable de parcourir les instructions de base d'un algorithme voir un programme séquentiel (linéaire), ainsi que les principales structures non linéaires à savoir les structures conditionnelle et répétitives. Ces dernières utilisées justement pour manipuler les tableaux à une et à deux dimensions, un autre type de variable dite composée et indicée, utilisé lorsque nous avons besoin de manipuler de multiples variables du même type. Il est possible de procéder par déclaration statique ou dynamique de ce type de structure. Nous avons clôturé ce chapitre avec un cas particulier de tableau, il s'agit des chaines de caractères, qui sont dotées d'un ensemble de fonctions et prédéfinies plus particulièrement dans la bibliothèque du langage c string.h.

Il est notamment possible d'avoir des variables de types différent, rattachées à une même entité. Personnaliser des structures selon le problème en question est désormais possible grâce aux structures personnalisées.

Ce polycopié contient un bon nombre d'exercices accompagnés de solutions types, afin de permettre aux étudiants une meilleure compréhension du cours et une application des connaissances.

Références bibliographiques

[1] Breton, P. (1987). L'informatique comme discipline existe-t-elle? Histoire d'un clivage qui sépare les informaticiens. Réseaux. Communication-Technologie-Société, 5(24), 65-75.

[2] Le dictionnaire Larousse, https://www.larousse.fr/dictionnaires/francais/ordinateur/56358.

[3] Delannoy, C. (2002). Le livre du C: premier langage: . Eyrolles.

[4] Haro, C. (2009). Algorithmique: raisonner pour concevoir. Editions ENI.

[5] Ritchie, D. M., Johnson, S. C., Lesk, M. E., & Kernighan, B. W. (1978). The C programming language. Bell Sys. Tech. J, 57(6), 1991-2019.

[6] Kernighan (B.W.) et Richie (D.M.) (1988). *The C programming language*. – Prentice Hall, 1988, seconde édition.

[7] Berthet, D., & Labatut, V. (2014). Algorithmique & programmation en langage C-vol. 1 (Doctoral dissertation, Université Galatasaray).

[8] Guerid, A., Breguet, P., & Röthlisberger, H. (2002). Algorithmes et structures de données avec Ada, C++ et Java. PPUR presses polytechniques.

[9] Cormen, T., Leiserson, C. , Rivest, R., Stein, C.(2002). " Introduction à l'algorithmique : Cours et exercices " 2ième édition, Dunod

[10]Courtin, J., Kowarski, I., & Arsac, J. (1995). Initiation à l'algorithmique et aux structures de données, Dunod.

[11]Malgouyres, R., Zrour, R., & Feschet, F. (2014). Initiation à l'algorithmique et à la programmation en C-3e éd. Cours avec 129 exercices corrigés, Dunod.

[12]Le Bars, J. M. (2016). Quelques études de l'aléatoire en informatique (Doctoral dissertation, Normandie Université).

[13] Garreta, H. (1992). C: langage, bibliothèque, applications. InterEditions.

[14] ROHAUT, S. (2007) « Algorithmique Techniques fondamentales de programmation » ENI, 2007.