الجمـهورية الجـزائرية الديمقـراطية الشـعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العـالـي والبـحـث العلـمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Relizane
Faculté des sciences et technologies
Département d'Informatique

جامعة غليزان
RELIZANE UNIVERSITY

ـ جامعة غليـــــــــزان ـ
كلية العلـــــــوم والتـكنولـوجـيـا
قسم الاعلام الالي

Support de cours du module

Algorithmique et Structures de données I

1ère Année MI

Dr. Said KHELIFA

Année Universitaire : 2022/2023

AVANT-PROPOS

Ce polycopié "*Algorithmique et Structures de Données 1*" est rédigé à l'intention des étudiants de première année du premier cycle universitaire (*licence*) du département MI (Socle Commun en Mathématiques et Informatique). Il constitue un manuel de cours et d'exercices sur une partie du domaine de programmation. L'objectif principal est de faire apprendre aux étudiants à résoudre un problème de manière informatisée.

Ce polycopié est structuré en six chapitres comme suit :

Dans le premier chapitre nous nous intéressons à quelques définitions comme l'informatique, à l'ordinateur et à l'algorithmique.

Dans le second chapitre, des notions de base sur la structure globale d'un algorithme sont données, ainsi que les différentes parties qui le composent suivie par les instructions de base les plus élémentaires. Ce chapitre se veut une introduction à toutes les structures de données qui vont être les objets de manipulation de toutes les définitions de contrôle qui suivront.

Quant aux structures de contrôle, nous les avons développées séparément dans deux chapitres : Le troisième chapitre décrit en détails les différentes structures conditionnelles. Le quatrième chapitre décrit les différentes structures de contrôles (*boucles*) qui peuvent être utilisées dans un algorithme (*ex. Pour, tant que, ...*).

Le chapitre cinq aborde l'utilisation des tableaux et les chaines de caractères dans la programmation ainsi que dans le cadre de l'algorithmique. Dans le dernier chapitre, l'utilisation des types personnalisés comme les enregistrements et les Enumérations sont expliqués.

Une liste de références bibliographiques est donnée à la fin de ce manuscrit.

À la fin de chaque chapitre, nous avons choisi des exercices et problèmes contigus aux notions de cours présentés, formalisés comme complément à la compréhension des notions.

Enfin, nous espérons que le présent ouvrage aura le mérite d'être un bon support pédagogique pour l'enseignant et un document permettant une concrétisation expérimentale pour l'étudiant.

L'auteur

Saïd Khelifa

FICHE MATIERE

Objectifs généraux :

Il s'agit d'un fascicule du cours/exercices du module *Algorithmique et Structures de Données I*. Il a pour but de :

- Acquérir les notions fondamentales de l'algorithmique.
- Apprendre à concevoir des algorithmes efficaces indépendamment des langages ou environnements d'exécution.
- Étudier les types de données et leurs utilisations courantes, à l'aide d'algorithmes adaptés.
- Acquérir des bases algorithmiques (affectation, entrées - sorties, structures conditionnelles, structures itératives et boucles), notion d'emplacement mémoire (Tableaux), chaines de caractères et les types personnalisés.
- Possibilité de définir de nouveaux types par l'utilisateur de façon plus agréable pour décrire des données complexes ou composées.
- Les mettre en œuvre avec le langage de programmation C.

Pré-requis :

- Mathématiques de l'école primaire.

Volume horaire :

Ce module est présenté, de manière hebdomadaire, comme suit:

- 3h00mn de cours.
- 1h30mn de Travaux diriges pour chaque groupe.
- 3h00mn de Travaux pratiques pour chaque groupe.
- 3h00mn de Travail personnel.

Soit en total : 105h

Moyens pédagogiques

- Tableaux.
- Salle de TDs.
- Salle de TPs dotée de micro-ordinateurs.
- Polycopies de cours /Fiches de TDs et TPs.

Evaluation

- Coefficient : 04.
- Crédits : 06.
- Note du contrôle continu : 40%.
- Note d'examen : 60%.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

Chapitre 1

Introduction à l'informatique et a l'algorithmique

Sommaire

Objectif

Dans ce premier chapitre introductif nous donnons un aperçu historique sur l'évolution de l'informatique ainsi que quelques notions de base sur l'informatique, sur les ordinateurs ainsi que sur l'algorithmique.

I. Introduction

L'homme a toujours eu besoin de compter. Au cours de la préhistoire, il ne savait calculer qu'à l'aide de cailloux ou de ses mains qui furent sans doute les premières calculatrices de poche. Les doigts ont servi à nos ancêtres pour compter et pour effectuer toutes sortes d'opérations arithmétiques.

Après l'apparition des calculateurs numériques, l'ancienne méthode de calcul commence à disparaître. Curieusement, on parle de calcul digital dans la nouvelle science informatique (Science de traitement automatique de l'information).

L'informatique est une discipline qui pénètre chaque jour un peu plus les autres sciences ; est devenu de plus en plus omniprésente dans tous les secteurs d'activité humaine. Cette progression nous amène à accorder plus d'importance à l'algorithmique et par conséquent à la programmation qui font l'origine de cette évolution.

L'algorithmique, est très simplement une méthode ou une façon systématique de procéder pour faire quelque chose. Il est souvent synonyme de logique et de rigueur puisqu'il permet finalement aux développeurs de présenter leurs idées devant une machine neutre.

II. Généralités :

Pour nous introduire dans le domaine de l'informatique et des ordinateurs. Nous avons opté pour une méthode très simple qui consiste à répondre aux questions suivantes :

1. Qu'est-ce que l'informatique ?

Le mot informatique a été créé en 1962 par Philippe Dreyfus pour désigner le traitement automatique des informations. En effet, il s'agit d'une contraction des deux mots « Information » et « Automatique » comme suit :

Informatique = ***INFOR****mation* + *auto****MATIQUE.***

Ce néologisme a été ensuite accepté par l'Académie française en 1966, et l'informatique est devenue officiellement "la science du traitement automatique de l'information", notamment par des machines automatiques, où l'information est considérée comme le support des connaissances et des communications, dans les domaines techniques, économiques et sociaux.

Le mot français Informatique n'a pas un seul véritable mot équivalent en anglais, car aux Etats-Unis on parle de Computer Science (science du calcul) alors que **Informatics** est admis par les Britanniques .

Définition 1.1 : Informatique

L'informatique est la science du traitement automatique de l'information. Autrement dit, c'est l'ensemble des théories, méthodes et techniques utilisées pour le traitement automatique de **l'information** à l'aide des machines électroniques (**Ordinateurs**).

D'après la définition de l'Informatique, on trouve deux nouveaux mots qui sont : **Information** et **Ordinateur**.

2. Qu'est-ce qu'une Information ?

Les informations traitées par un ordinateur sont de différents types mais elles sont toujours représentées sous forme binaire. Une information élémentaire correspond donc à un chiffre 0 ou 1 appelé bit (0 => 0 volt et 1 => 5 volts). Une information plus complexe, telle qu'un caractère ou un nombre, se ramène à un ensemble de bits.

Quels types d'informations sont traités par un ordinateur ? On distingue les **instructions** et les **données**.

- **Instructions** : Elles représentent les opérations (addition, par exemple) effectuée par un ordinateur. Elles sont composées de plusieurs champs, qui sont les suivants :
    - Le code de l'opération ;
    - Les opérandes impliqués dans l'opération.

- **Données** : Elles sont les opérandes sur lesquels portent les opérations, ou produites par celles-ci. On distingue les données numériques, pouvant être l'objet d'une opération arithmétique, et les données non numériques, par exemple, les symboles constituant un texte.

3. Qu'est-ce qu'un ordinateur ?

Le mot ordinateur est un terme générique qui a été créé en 1955 à la demande d'IBM, pour désigner un appareil ou une machine électronique programmable servant au traitement de l'information sous différentes formes (valeurs numériques, texte, image…) selon des séquences d'instructions (les programmes).

L'intérêt d'un ordinateur revient principalement à sa capacité de manipuler **rapidement** et **sans erreur** un grand nombre d'informations, à **mémoriser** des quantités de données numériques ou alphabétiques, à rechercher, comparer ou classer les informations mémorisées.

Tout ordinateur à 3 fonctions principales : La fonction d'**entrée**, la fonction du **traitement**, la fonction de **sortie**.

4. Quels sont Les aspects de l'Informatique ?

L'informatique traite deux aspects complémentaires : les programmes (**logiciel / software**) qui décrivent un traitement à réaliser et les machines (**matériel / hardware**) qui exécutent ce traitement.

Définition 1.2 : Logiciel

Un logiciel ou une application est un ensemble structuré d'instructions (et de programmes), qui permet à un ordinateur ou à un système informatique de traiter des informations, d'assurer une tâche ou une fonction particulière.

Définition 1.3 : Matériel

Le matériel informatique est l'ensemble des éléments physiques (microprocesseur, mémoire, écran, clavier, disques durs…) constituant un ordinateur, utilisés pour le traitement automatique des informations en utilisant des logiciels.

III. Architecture Matérielle d'un Ordinateur

L'architecture matérielle d'un ordinateur repose sur le modèle de Von Neumann qui distingue classiquement 3 parties (Figure 1.1) :

1. **Les périphériques d'entrées/sorties (E/S) ou interfaces :** ce sont toutes les unités destinées à la lecture ou l'écriture de l'information (exemple : clavier, écran, imprimante, souris, …).

2. **Un microprocesseur :** sa tâche est d'effectuer des calculs, de gérer et synchroniser les transferts de données entre les différents organes du système. Il est composé de :

   - **Unité Arithmétique et Logique (U.A.L) :** réalise les opérations arithmétiques de base (+ - * /), de comparaison et les opérations logiques (ET / OU / NON) contenus dans l'instruction et effectue aussi des échanges de données avec la mémoire vive.

   - **Unité de Commande et de Contrôle (U.C.C) :** son rôle est de séquencer les opérations. U.C.C commande et contrôle le fonctionnement de l'UAL, de la mémoire et des E/S. Elle se charge de chercher l'instruction à exécuter dans la mémoire principale (et les données qu'elles utilisent), décode cette instruction, et envoie le cas échéant un signal à l'UAL pour se préparer à l'exécution.

3. **La mémoire :** contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise en :

- **Une mémoire vive (RAM : Random Access Memory) :** pouvant être lue et écrite. Nécessaire pour l'exécution de tout programme (contient programmes et données en cours de fonctionnement). Ce type de mémoire est volatile.

- **Une mémoire morte (ROM : Read Only Memory)** : pouvant être lue mais pas (ou peu de fois) écrite. Elle contient des programmes essentiels au fonctionnement du matériel, surtout lors du démarrage (avant le chargement du système d'exploitation dans la RAM), elle est généralement programmée par le fabricant. Toutefois, il existe des variantes telles que:

    - PROM [Programmable ROM]: pouvant être écrite une seule fois par l'utilisateur.

    - REPROM [REProgrammable ROM]: pouvant être écrite un certain nombre de fois par l'utilisateur.

Les entités citées ci-dessus sont reliés physiquement par des pistes (lignes) électriques appelées « bus ». Il existe plusieurs types de bus, tels que :

- **Le bus de données :** permet la circulation des données entre les organes de la machine.
- **Le bus d'adresse :** véhicule les adresses des cases mémoires et des périphériques sollicitée par le microprocesseur.
- **Le bus de commandes :** véhicule l'ordre à exécuter (ex. une instruction de de lecture ou d'écriture).

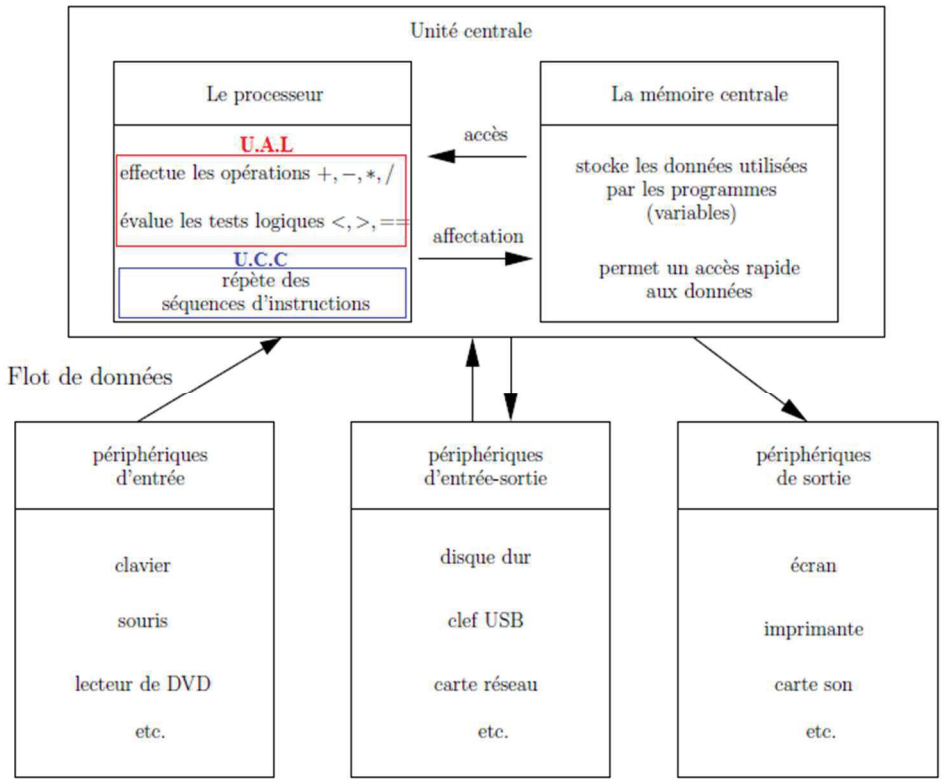


Figure 1.1 Architecture de Von Neumann

Pour exécuter une commande, le microprocesseur a besoin de connaitre certaines informations. Ces données lui sont introduites via une unité d'entrée (clavier, scanner, camera…etc.)

Une fois les données en mémoire, le microprocesseur consulte le programme de traitement relatif au type d'informations chargé et les modifie en conséquence.

Pour suivre et contrôler aussi bien le processus d'introduction des données que celui de leur traitement, l'ordinateur met à la disposition de l'utilisateur un moniteur (écran d'affichage).

A la fin des opérations de traitement, les résultats sont envoyés vers une unité de sortie (imprimante, moniteur…etc.)

Pour conclure, l'ordinateur est une machine capable de :

- Recevoir des données (de l'utilisateur le plus souvent) en mémoire via des périphériques d'entrée,
- Traiter les données grâce à l'unité de traitement
- Fournir les données (résultats) de sa mémoire vers des périphériques de sortie (écran par exemple).

La figure suivante illustre ces fonctions

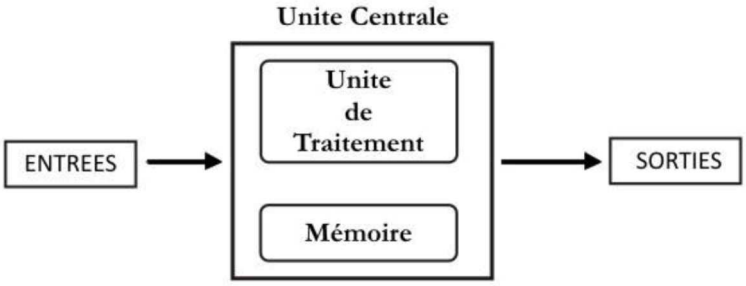


Figure 1.2 Structure simple d'un ordinateur

Pour donner des ordres à l'ordinateur, il est nécessaire de pouvoir communiquer avec lui. Cette communication passe par un **langage de programmation**, dans lequel est écrit les programmes.

IV. Brève historique de l'informatique

L'informatique, comme discipline scientifique et technique, s'est déployée sur deux siècles environs : 19ème et 20ème siècle, elle a connu une évolution extrêmement rapide. L'histoire de l'informatique résulte de la conjonction entre des découvertes scientifiques

de plusieurs disciplines techniques et sociales. Elle est liée à l'apparition des premiers automates et à la mécanisation : un processus de développement et de généralisation des machines qui a commencé au 18ème siècle en Europe avec l'industrialisation.

Dans ce qui suit, nous passons en revue les événements scientifiques les plus importants de l'histoire de l'humanité, qui ont eu le plus grand impact sur l'apparition et l'évolution de l'informatique .

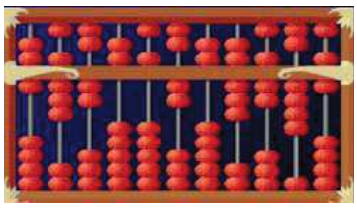
Tableau 1.1 : Histoire et évolution de l'informatique

La préhistoire : de 3000 AC à 1900

- L'ABAQUE : Inventé par les Babyloniens (3000 avant notre ère).

  Tables sur lesquelles étaient posées des petites pierres, utilisés dès le 7e siècle av. J.-C. Rappelons que le mot **calcul** vient du latin *calculus* qui signifie caillou.

Abaque



- Formalisation du calcul : Al Khawarizmi (IXème)

  Le développement de l'informatique est lié à la recherche fondamentale en mathématiques et plus précisément à la logique et aux algorithmes mathématiques, apparus au début du IXème siècle avec les travaux du mathématicien arabe Abu Jaffar Al Khawarizmi.

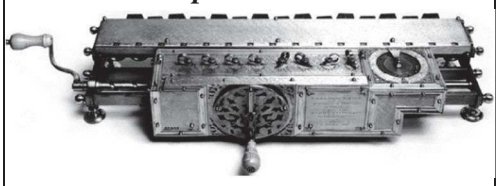
Al Khawarizmi

- Vers 1642, Blaise Pascal développe une machine à calculer mécanique, **la Pascaline**. Considérée comme la première machine à calculer. Elle permettait d'additionner et de soustraire deux nombres d'une façon directe et de faire des multiplications et des divisions par répétitions.

Pascaline

- Gottfried Wilhelm Leibniz en 1671 a construit le **Step Reckoner**. Cette machine pouvait calculer les additions, soustractions, et aussi les multiplications, les divisions et les racines carrées par des séquences d'additions décalées.

Step Reckoner



- En 1834, **Charles Babbage** s'inspire du métier à tisser de Jacquard pour élaborer une machine qui, à l'aide de cartes perforées, évalue les différentes fonctions.

  L'information contenue sur les cartes perforées utilisait un code basé sur la présence ou l'absence d'un trou à des endroits précis sur la carte. Un système mécanique complexe permettait à la machine de Babbage de lire ce qui était contenu sur les cartes. le principe de la carte perforée allait permettre le développement du langage binaire, un code utilisé par l'ordinateur.

Machine de Babbage

De 1900 à 1940

- **Machine de Turing** : En 1936, Alan Turing démontre qu'on ne peut pas tout calculer de manière automatique. Il imagine pour sa démonstration un outil qui inspire encore le fonctionnement de nos ordinateurs. Une machine universelle qui manipule des informations – des lettres ou des chiffres – suivant des règles définies dans une table.

Machine de Turing

Les années 40

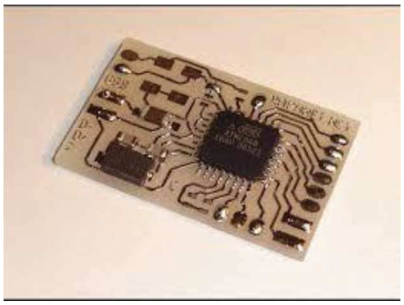
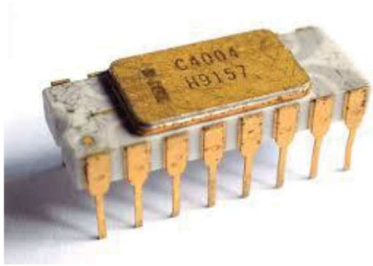
- Travaux en cryptographie (décodage : Machine Enigma, Colossus)

- En 1943, **Howard Aiken** met en place en collaboration avec **IBM** le premier calculateur électromécanique : **Mark I**. C'est une machine qui pèse 5 tonnes et mesure 17 m de long et 2,5 m en hauteur. Cette machine calcule **5 fois plus vite que l'homme**.

- En 1945, l'**ENIAC** *fut inventé par deux ingénieurs américains* ***John Erkert*** *et* ***John Mauchly****, il* devient le premier ordinateur entièrement électronique ne comportant plus **aucune pièce mécanique**. Il est composé de 18 000 lampes et s'étend sur plus 160 m². Il sera utilisé pour mettre au point la bombe H.

- Alors que l'ENIAC n'était programmable que manuellement, l'**EDVAC** permet la **mémorisation**. L'EDVAC est une réelle innovation en 1946 puisqu'il permet de mémoriser 1024 mots en mémoire centrale et 20 000 en mémoire magnétique.

Ordinateur ENIAC

➢ Sous l'influence de John von Neumann, l'ENIAC est à partir de 1947 reconverti en ordinateur à programme enregistré, ➢ L'apparition du **transistor** en 1948 révolutionne l'informatique permettant ainsi de fabriquer des ordinateurs moins encombrants et qui consomment moins d'électricité.	
Les années 50 ➢ Les années 1950 voient apparaître de **nouveaux langages de programmation** - Compilateurs (FORTRAN en 1957) - LISP en 1958 ➢ La naissance des **circuits intégrés** en 1959. Le circuit intégré relie plusieurs transistors sans utiliser de fil électrique. Il permet ainsi de réduire encore la taille et le coût de l'ordinateur.	**circuit intégré** 
Les années 60 ➢ Systèmes d'exploitation ➢ Basic en 1964 ➢ Automates – Langages formels - Correction de programmes ➢ Micro-processeurs ➢ En 1962, création d'un réseau de communication militaire par l'US Air Force. ➢ En 1969, création du réseau expérimental ARPANET par l'ARPA, qui est aujourd'hui considéré comme le réseau précurseur d'internet.	**Micro-processeurs** 
Les années 70 ➢ Base de Données Relationnelles ➢ Unix et C (Thompson et Richie) ➢ Pascal et Ada ➢ En 1972, Ray Tomlinson mit au point le courrier électronique. ➢ 1972, le réseau ARPANET fut présenté au public ➢ En 1976, création du protocole TCP. ➢ En 1978, le protocole TCP fut fragmenté en deux protocoles : TCP et IP.	

Les années 80 ➢ Micro-ordinateur personnel (Apple – MacIntosh en 84) ➢ Premiers virus en 1988 ➢ En 1984, création du système de nommage DNS. ➢ Fin 1990, création du protocole HTTP et du langage HTML par Tim Berners-Lee.	
Les années 90 ➢ Sortie du Windows ➢ La fin de la Micro - l'ère du compatible PC	
Les années 2000 ➢ Ressort des OrdiPhone (Smart Phone et iPAD) et lancement de la programmation Mobile.	

V. Introduction à l'algorithmique

La notion d'algorithme est antérieure à l'apparition des ordinateurs. L'homme depuis longtemps essaye de déterminer des procédés suffisamment précis pour résoudre ses problèmes et pour organiser ses activités. Avec l'avènement de l'informatique, cette notion est devenue plus restrictive avec un objectif supplémentaire est de pouvoir traduire un algorithme en un programme.

- **Historique :**

Le mot « algorithme » est un terme d'origine arabe qui vient de la transcription latinisée du nom d'un célèbre mathématicien musulman perse du 9ième siècle, Al Khawarizmi (780-850) auteur d'un ouvrage décrivant des méthodes de calculs algébriques, et du mot grec ***arithmos*** qui signifie « nombre » .

Le mot algorithme se référait à l'origine uniquement aux règles d'arithmétique utilisant les chiffres indo-arabes numérales mais cela a évolué par la traduction en latin européen du nom Al-Khawarizmi en algorithme au 18ième siècle. L'utilisation du mot a évolué pour inclure toutes les procédures définies pour résoudre un problème ou accomplir une tâche.

- **Qu'est-ce qu'un Algorithme ?**

On appelle un algorithme l'ensemble d'opérations et de règles définissant « ce qu'il faut faire » et « dans quel ordre » pour résoudre un problème (ou une classe de problème) .

Définition 1.4 : Algorithme

Un algorithme est une suite finie d'instructions, qui une fois exécutée correctement dans un ordre bien déterminé donne une solution au problème posé. Cette définition se base sur le principe de décomposition d'une tâche quelconque en un ensemble de tâches élémentaires exécutés en séquence suivant un enchaînement strict.

Exemples d'algorithmes (de la vie courante)

En principe, la notion d'algorithmes est très générale et ne se limite pas au cadre de l'informatique. En effet, nous utilisons dans notre vie courante, des méthodes algorithmiques. Exemple :

1. Une recette de cuisine : contient deux parties : Les ingrédients et le mode de préparation. La recette précise clairement, les ingrédients nécessaires à la préparation d'un repas ou d'un gâteau ainsi que les opérations de préparation par ordre chronologique.

2. Une notice de montage d'un meuble livré en kit (Bureau, armoire, …). Cette notice montre comment monter le meuble livré en montrant les étapes, en général, par des schémas évolutifs.

3. Manuel d'utilisation d'un appareil électronique, électroménager, …

4. Résolution d'une équation du second degré ($\mathbf{Ax^2 + Bx + C = 0}$).

Exemples :

L'algorithme qui calcul et affiche la surface et le périmètre d'un cercle de rayon R, peut-être d'écrit comme suit :

- Introduire le rayon R
- Calculer la surface S suivant la loi $S = \pi * R^2$
- Calculer le périmètre P suivant la loi $P = 2 * \pi * R$
- Afficher les valeurs de : S et P

Cet algorithme nécessite le rayon R et fournie la surface et le périmètre du cercle.

Caractéristiques d'un algorithme :

1. **Lisibilité** : un algorithme doit être lisible et compréhensible même par un non-informaticien.

2. **De haut niveau***:* L'intérêt d'un algorithme est la possibilité d'être codé dans un langage algorithmique et pouvoir être traduit en n'importe quel langage de programmation, il ne doit donc pas faire appel à des notions techniques relatives à un programme particulier ou bien à un système d'exploitation donné.

3. **Complet:** Il faut que le programme considère tous les cas possibles et donne un résultat dans chaque cas.

4. **Efficace:** Il faut que le programme exécute sa tâche avec efficacité de telle sorte qu'il se déroule en un temps minimal et qu'il consomme un minimum de ressources.

5. **Concis:** un algorithme ne doit pas dépasser une page. Si c'est le cas, il faut décomposer le problème en plusieurs sous-problèmes

6. **Structuré:** un algorithme doit être composé de différentes parties facilement identifiables à savoir : un entête où apparaissent le nom de l'algorithme ainsi que les déclarations des objets manipulés puis le corps de l'algorithme qui commence par "Début" et se termine par "Fin. Entre ces deux mots clés se trouvent les actions ordonnées à exécuter par la machine.

VI. Conclusion

L'informatique est devenue omniprésente dans tous les secteurs d'activités de notre vie quotidienne. Nous avons vu dans ce chapitre comment l'informatique à évoluer rapidement, et comment nous pouvons l'utiliser pour résoudre nos problèmes à travers l'algorithmique et la programmation. Dans le chapitre suivant, nous allons aborder les notions de base concernant un algorithme séquentiel simple. Ces notions constituent les ingrédients indispensables dans n'importe quel algorithme.

Série d'exercices 1

Questions à choix multiples

Pour réaliser un traitement, l'ordinateur exécute		**Q1**
1.	Des données	
2.	Des instructions	
3.	Un programme	
4.	Des résultats	

Pour sauvegarder les informations de manière permanente, j'utilise :		**Q2**
1.	Disque Dur	
2.	RAM	
3.	ROM	
4.	Processeur	

Laquelle des mémoires suivantes est non volatile ?		**Q3**
1.	RAM	
2.	Disque Dur externe	
3.	ROM	
4.	REPROM	

Une clé USB est basée sur une mémoire		**Q4**
1.	Non volatile	
2.	Volatile	
3.	Permanente	
4.	Souple	

Dans le micro-ordinateur le traitement de l'information est réalisé par :		**Q5**
1.	Processeur.	
2.	RAM.	
3.	ROM	
4.	RAM et ROM	

Pour prendre connaissance des données à traiter, on effectue :		**Q6**
1.	une écriture	
2.	une lecture	
3.	Une sauvegarde	
4.	un traitement numérique	

Un Ecran Tactile est un périphérique de :		**Q7**
1.	Entrée.	
2.	Sortie.	
3.	Entrée et Sortie	
4.	Traitement	

Chapitre 2

Algorithme séquentiel simple

Sommaire

Objectif

Dans ce chapitre, on présente les fondamentaux qui vont guider l'étudiant pour concevoir et analyser des algorithmes.

L'objectif de ce chapitre est de présenter la structure d'un algorithme séquentiel simple avec ses composants et ses opérations de base.

Une traduction en langage C est donnée à la fin de ce chapitre pour permettre à l'étudiant d'aborder le domaine de développement en réalisant son premier programme.

I. Introduction

On a noté dans le premier chapitre qu'un ordinateur sert à faire des traitements automatiques sur les informations. Autrement dit, l'activité de programmation permet à un ordinateur de donner des solutions à des problèmes réels. De façon générale, la programmation consiste, à partir d'un problème donné, à réaliser un programme dont l'exécution apporte une solution satisfaisante au problème posé. Le cœur du processus de résolution d'un problème est la conception d'algorithmes.

Un algorithme permet d'expliciter clairement les idées de la solution indépendamment d'un langage de programmation. L'intérêt d'un algorithme est la possibilité d'être codé dans un langage algorithmique et pouvoir être traduit en n'importe quel langage de programmation, en un programme exécutable par un ordinateur.

II. Notion de langage et langage algorithmique

Définition 2.1 : Langage algorithmique

Le langage algorithmique est un pseudo-langage qui tient en compte des caractéristiques de la machine, tout en étant plus souple qu'un langage de programmation. C'est, donc, un compromis entre le langage naturel et un langage de programmation.

Ce langage utilise un ensemble de **mots clés** et de **structures** permettant de décrire de manière complète et claire, les objets manipulés par l'algorithme ainsi que l'ensemble des **instructions** à exécuter sur ces objets pour résoudre un problème.

Définition 2.2 : Mots clés

Un mot clé est un identificateur (mot) qui a une signification particulière au langage, comme : *Début, Fin, Si, Pour, Répéter, Tantque, …etc*

Définition 2.3 : Programme

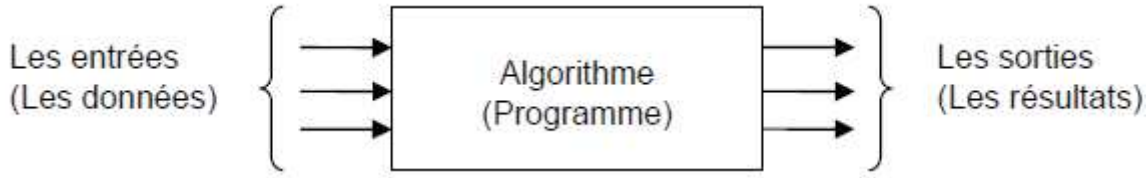
Un programme est la traduction d'un algorithme dans un langage de programmation compréhensible par la machine (en langage C, par exemple).

Un programme informatique, aussi bien qu'un algorithme, réalise en général trois choses :

1- **Introduire les données nécessaires (lire des données en entrée) :** Le programme doit en effet savoir à partir de quoi travailler. Par exemple, pour utiliser une calculatrice, on doit lui donner des nombres et lui dire quelles opérations effectuer. Pour cela, on utilise souvent un *clavier*, mais le programme peut aussi tirer les données d'un *disque dur* ou encore d'un *autre ordinateur via* un réseau ou autre.

2. **Le traitement** : exécuter séquentiellement des instructions sur ces données. À partir des données en entrée, le programme va appliquer automatiquement des méthodes pour traiter ces données et produire un résultat. Par exemple, une calculatrice va appliquer l'opération d'addition ou de multiplication.

3. **Afficher les résultats obtenus (écrire les résultats en sortie)** : Lorsque le programme a obtenu un résultat, il doit écrire ce résultat quelque part pour qu'on puisse l'utiliser. Par exemple, une calculatrice va afficher un résultat à l'*écran* ou stocker le résultat en *mémoire*.



Le travail d'un programmeur consiste à créer des programmes informatiques. Le programmeur doit pour cela expliquer à l'ordinateur dans un certain langage, appelé *langage de programmation*, quelles sont les données et quelles sont les méthodes à appliquer pour traiter ces données.

Définition 2.4 : Langage de programmation

Un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent. D'une manière similaire à une langue naturelle, un langage de programmation est composé d'un alphabet (lettres, chiffres, symboles, caractère spéciaux…), d'un vocabulaire, d'un ensemble de règles (syntaxe ou grammaire) et de significations qui permettent de réunir les éléments du vocabulaire pour former des phrases (ligne de programme) correctes.

Le langage de programmation est l'intermédiaire entre l'humain et la machine, il permet d'écrire dans un langage proche de la machine mais intelligible par l'humain les opérations que l'ordinateur doit effectuer. Ainsi, étant donné que le langage de programmation est destiné à l'ordinateur, il doit donc respecter une syntaxe stricte.

III. Cycle de développement d'un programme

L'utilisation d'un ordinateur pour la résolution d'un problème consiste à réaliser un programme informatique qui reçoit les données en entrée et fournit les résultats en sortie. L'élaboration d'un programme passe généralement par les étapes suivantes :

Problème --> Analyse --> Algorithme --> Programme --> Compilation --> Exécution

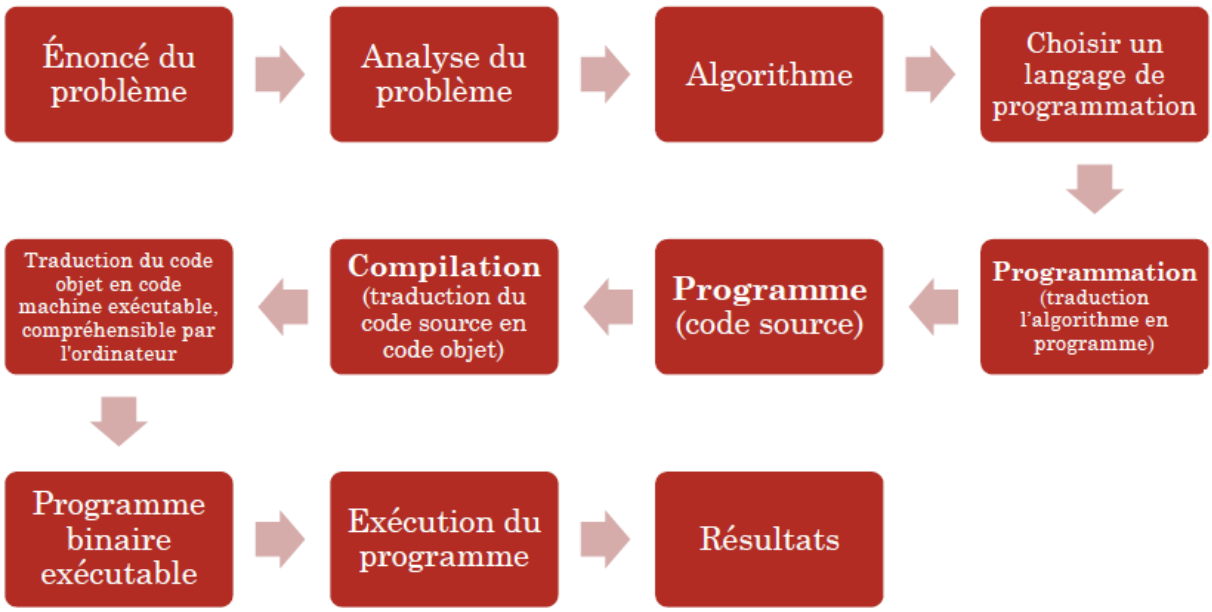


Figure 2.1. Résolution d'un problème informatique

Etape 1 : Définition du problème

Il s'agit de déterminer toutes les informations disponibles et la forme des résultats désirés.

Etape 2 : Analyse du problème

C'est une phase de réflexion qui permet d'identifier les caractéristiques du problème à traiter. Elle consiste à trouver le moyen de passer des données aux résultats.

Selon la complexité du problème P à résoudre, cette étape peut être décomposée en trois parties :

1) Décomposer P en N sous-problèmes (P_1, P_2,..., P_N) de complexité inférieure. Si un ou plusieurs sous-problèmes apparaissent encore trop complexes on les décompose à leurs tours.

2) Identifier les objets nécessaires à la formulation du modèle qui permet de définir le processus de résolution.
   - Objets relatifs aux données : les entrées.
   - Objets relatifs aux résultats : les sorties.
   - Objets intermédiaires qui résultent de la phase de décomposition.

3) Définir les relations qui existent entre ces objets en termes de règles, de formule, d'équations mathématiques et de méthodes de traitements.

Etape 3 : Algorithme

Une fois qu'on trouve le moyen de passer des données aux résultats, il faut être capable de rédiger une solution claire et non ambiguë.

Il s'agit d'une description compréhensible par un être humain de la suite des opérations à effectuer pour résoudre le problème analysé en respectant un formalisme (un ensemble de règles d'écriture) bien déterminé.

Les étapes 1, 2 et 3 se font sans le recours à la machine. Si on veut rendre l'algorithme concret ou pratique, il faudrait le traduire dans un langage de programmation.

Etape 4 : Programme

Cette étape consiste à traduire les instructions de l'algorithme, développé antérieurement, dans un langage de programmation choisi par l'utilisateur. Pour notre cours, nous traduirions en langage C.

Une fois le programme écrit, il va falloir le vérifier et le corriger en lançant la compilation.

Etape 5 : Compilation

Chaque langage de programmation contient un compilateur intégré. Le compilateur est un logiciel qui détecte les erreurs de syntaxe du programme, mais ne détecte pas les erreurs de logique.

- Si le programme est syntaxiquement correct, le compilateur crée ce qu'on appelle programme objet, c'est un programme prêt à être exécuté.

- Si le programme contient des erreurs de syntaxe, le compilateur affiche la liste des erreurs à l'écran.

Etape 6 : Exécution

L'ordinateur exécute les instructions d'un programme en langage "binaire". Ainsi, l'utilisateur "lance" l'exécution de programme compilé pour savoir quel est le résultat.

IV. Structure générale d'un algorithme

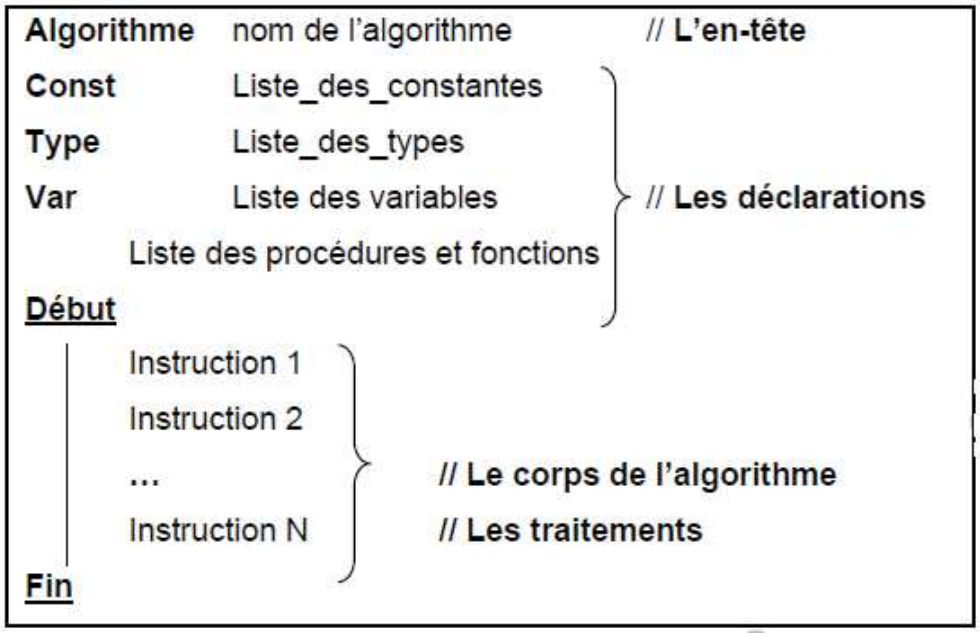
Un algorithme est composé de trois parties indissociables :

1) **Entête :** permet d'identifier l'algorithme. On spécifie le nom de l'algorithme par exemple : Algorithme tri ;

2) **Déclaration :** contient la déclaration de de tous les objets (constantes, variables, …) avec leurs types utilisés et manipulés dans le corps de l'algorithme.

3) **Corps :** contient la séquence d'instructions entre les deux mots clés : Début et Fin

La structure générale d'un algorithme est la suivante :



Exemple : Calculer la surface d'un carré

```
ALGORITHME TEST ;
 VAR
   LONGUEUR, SURFACE : RÉEL ;
DÉBUT
      ECRIRE ("ENTREZ SVP LA LONGUEUR DU CARRÉ") ;
       LIRE (LONGUEUR) ;
       SURFACE ← LONGUEUR*LONGUEUR ;
      ECRIRE ("LA SURFACE DU CARRÉ EST ", SURFACE) ;
FIN.
```

La succession d'instructions n'est pas toujours détaillée explicitement au sein de l'algorithme, mais parfois dans une autre partie, à travers ce que l'on appelle des fonctions ou des procédures. Cela permet de découper l'algorithme en plusieurs sous-algorithmes et de le rendre plus facile à comprendre.

V. Les données : variables et constantes

Lors de la préparation d'un algorithme, on a toujours besoin de stocker provisoirement des valeurs. Il peut s'agir des données fournies par l'utilisateur, comme il peut s'agir des résultats obtenus par le programme (intermédiaires ou définitifs). Pour cela, on utilise des objets.

Un objet est de nature ***variable***, si sa valeur peut changer pendant l'exécution des actions de l'algorithme. Comme il est de nature ***constante*** si sa valeur est invariable.

Tous les objets qui deviennent partie intégrante de l'exécution d'un algorithme doivent être déclarés avant leurs utilisation; ceci constitue la partie déclaration d'un algorithme. Un objet peut être décrit par un **nom (identificateur**), un **type** et une **valeur**.

- **Nom (identificateur**) : est utilisé pour donner un nom à un objet, afin de l'identifier et de le distinguer des autres objets dans l'algorithme

  Remarques :
  - Un identificateur doit être significatif (représentatif) pour faciliter la compréhension de l'algorithme.
  - Un identificateur est une suite de lettres non accentuées et/ou de chiffres, commençant par une lettre.
  - En l'algorithmique, il n'y a pas de différence entre minuscules et majuscules.
  - Le caractère « _ » peut être utilisé pour construire des noms d'identificateurs composés de plusieurs mots.
  - Un identificateur doit être différent des mots clés (algorithme, début, fin, ….)

- **Type :** caractérise l'intervalle ou l'ensemble des valeurs que peut prendre cet objet ainsi que les opérations qui lui sont autorisées tout au long de l'algorithme.

  Une fois qu'un type de données est associé à une variable le contenu de cette variable doit **obligatoirement** être du même type.

- **Valeur :** La valeur d'un objet est celle qui lui a été assignée pour représenter la grandeur que contient cet objet.

  Dans la partie déclaration, les variables n'ont pas de valeur. Attribuer une valeur à une variable se fait par le biais d'une instruction. Initialiser une variable, c'est lui donner une première valeur.

Définition 2.5 : Variables

Une variable est une case mémoire qui sert à stocker l'information traitée par un algorithme. Cette variable associe un nom à une valeur qui peut varier au cours du temps de l'exécution d'un algorithme.

En effet, la variable est représentée dans la mémoire par un nombre de mots mémoire correspond à leur type (Figure 2.2).

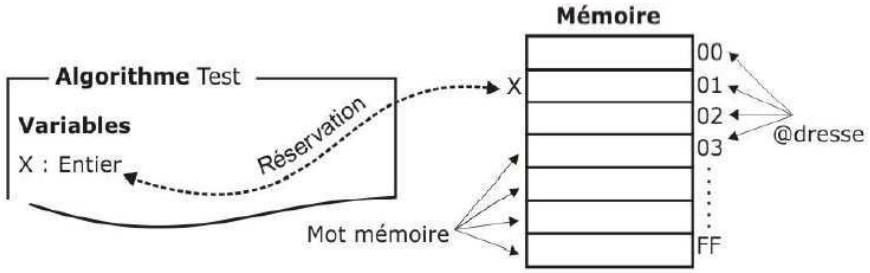


Figure 2.2. Représentation d'une variable dans la mémoire

Un objet de nature variable est déclarée à l'aide du mot **VAR** (ou Variables). L'expression de déclaration de variables s'écrit suivant la règle suivante :

```
VAR
        Nom_variable : Type_variable ;
```

Exemple :

```
VAR
    X : Entier ;          // déclaration d'une variable X de type Entier
    Y, Z : réel ;         // déclaration d'une variable Y et Z de type réel
```

Définition 2.6 : Constantes

Une constante est objet informatique similaire à une variable en nom et type. Elle se diffère d'une variable seulement dans sa valeur qui ne varie pas au cours de l'exécution de l'algorithme, fixée en début de l'algorithme.

Une constante est déclarée à l'aide du mot **CONST** (ou Constante). Un objet de nature constante suit la déclaration suivante :

```
CONST
        Nom_constante = Valeur_constante ;
```

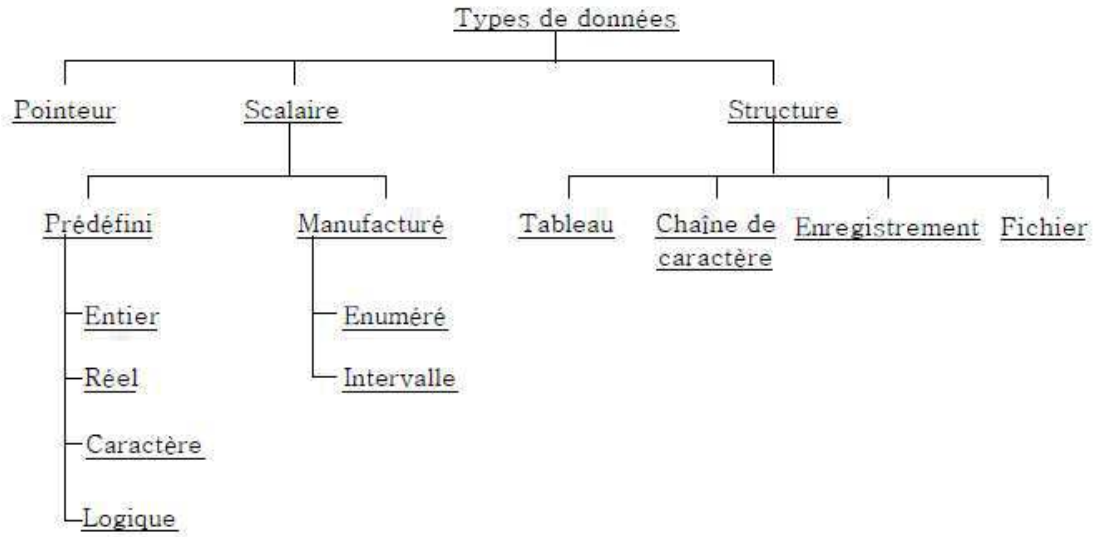
Exemple :

```
CONST
    Pi = 3.14              // déclaration d'une constante Y et sa valeur = 3.14
    ville = 'BISKRA'       // déclaration d'une constante ville de type chaine de
                              caractère et sa valeur = 3.14.
```

VI. Types de données

Un type définit l'ensemble des **valeurs** que peut prendre un objet. Il définit également les **opérations**, généralement appelées opérateurs, qui pourront être appliquées sur les données de ce type.

Les types de données sont résumés dans le diagramme suivant :



Dans cette partie du cours, on s'intéresse au type prédéfini. Ces types sont des types de base fréquemment utilisés, ils n'ont pas besoin d'être définis par l'utilisateur. Ils sont :

- **Type entier :** Ce type représente le domaine des nombres entiers (Z).
- **Type réel :** Ce type représente le domaine des nombres réels (R).
- **Type logique (booléenne):** Ce type représente le domaine logique qui contient deux valeurs logiques : {FAUX, VRAI},
- **Type caractère :** Ce type représente le domaine des caractères qui contient les lettres alphabétiques, les caractères numériques, les caractères spéciaux (., ?, !, <, >, =, *, +, ...etc), et le caractère espace (ou blanc).
- **Type chaîne de caractères** permet de représenter des mots ou des phrases.

Remarques :

- Une variable de type réel peut avoir une valeur de type entier, car le type entier est inclus dans le type réel, mais l'inverse est totalement faux.
- Une variable de type chaîne de caractères peut avoir une valeur de type caractère, car le type caractère est inclus dans le type chaîne de caractères, mais l'inverse est totalement faux.

VII. Opérations de base

Pour pouvoir comprendre et utiliser correctement les opérations de base en algorithmique, on doit tout d'abord aborder les notions d'opérateur, d'opérande et d'expression.

Définition 2.7 : Opérateur

Un opérateur est un symbole d'opération qui permet d'agir sur des variables ou de faire des "calculs". Il existe plusieurs types d'opérateurs :

- **Les opérateurs arithmétiques** qui permettent d'effectuer des opérations arithmétiques entre des opérandes numériques :
  Les opérateurs élémentaires sont : «+», «-», «x», «/», «[/]» (division entière) et l'opérateur de changement de signe «-» (qui est similaire à l'opérateur de soustraction).

- **Les opérateurs de comparaison** («<», «>», «≤», «≥», «=» et «≠») qui permettent de comparer deux opérandes et produisent une valeur booléenne, en s'appuyant sur des relations d'ordre :
  - Ordre naturel pour les entiers et les réels
  - Ordre lexicographique ASCII pour les caractères

- **Les opérateurs logiques** qui combinent des opérandes booléens pour former des expressions logiques plus complexes :
  - Unaire : «non» (négation)
  - Opérateurs binaires : «et» (conjonction), «ou» (disjonction), «ou_exclusif»

Définition 2.8 : Opérande

Un opérande est une entité (variable, constante ou expression) utilisée par un opérateur

Définition 2.9 : Expression

Une expression est une combinaison d'opérateur(s) (arithmétiques, logiques) et d'opérandes (constantes, variables), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type

Exemple :

Dans a + b

a est l'opérande gauche.

+ est l'opérateur.

b est l'opérande droit.

a + b est appelé une expression.

par exemple a vaut 2 et b 3, l'expression a + b vaut 5.

Si par exemple a et b sont des entiers, l'expression a + b est un entier.

L'évaluation d'une expression faisant intervenir plus d'un opérateur repose sur des règles de priorité entre opérateurs. Pour lever toute ambiguïté lors de l'évaluation d'une expression, il est nécessaire d'associer une priorité à chaque opérateur pour définir son ordre d'exécution. Comme il est possible de modifier cet ordre par l'utilisation des parenthèses.

- **Ordre de priorité décroissante des opérateurs arithmétiques**

Symbole	Opération
«-»	changement de signe
«x», «/», «[/]»	Multiplication, division et division entière
«+», «-»	Addition et Soustraction

- **Ordre de priorité décroissante des opérateurs logiques**

Symbole	Opération
«non»	négation logique (fonction inverse)
«et»	pour réaliser la conjonction logique entre deux valeurs booléennes
«ou»	pour réaliser la disjonction logique entre deux valeurs booléennes
«Xou»	pour réaliser le ou exclusif entre deux valeurs booléennes

Les opérations élémentaires

- **Sur les entiers** : Les opérations utilisables sur les entiers sont :
  - Les opérateurs arithmétiques classiques.
  - La division entière, notée DIV, telle que n DIV p donne la partie entière du quotient de la division entière de n par p.
  - Le MODulo, noté MOD, telle que n MOD p donne le reste de la division entière de n par p.
  - Les opérateurs de comparaison classiques.

- **Sur les réels :** Les opérations utilisables sur les réels sont :
  - Les opérations arithmétiques classiques.
  - Les opérateurs de comparaison classiques.

- **Sur les booléens :** Les opérations utilisables sur les booléens sont réalisées à l'aide des opérateurs logiques

- **Sur le caractère :** Les opérations élémentaires réalisables sont les opérateurs de comparaisons selon l'ordre lexicographique

- **Sur une chaîne de caractères :** Les opérations utilisables sur chaîne de caractères sont :
  - Les opérateurs de comparaison classiques selon l'ordre lexicographique.
  - La Concaténation représenter par un + .

VIII. Instructions de base

Les instructions de base permettent le transfert d'informations entre objets, ou une communication entre un algorithme et l'un des périphériques d'entrées ou de sorties. On distingue trois genres d'instructions élémentaires : l'affectation, la lecture et l'écriture.

A. Instruction d'affectation

C'est l'action par laquelle nous pouvons charger une valeur dans une variable X. Cette valeur peut elle-même être une variable, une constante ou le résultat de l'évaluation d'une expression arithmétique/logique.

L'affectation s'effectue en utilisant l'opérateur d'affectation représenté par le symbole ←

L'affectation est faite selon la syntaxe suivante :

- Affectation d'une valeur à une variable : **Variable ← valeur ;**
- Affectation d'une variable à une variable : **Variable1 ← Variable2**
- Affectation du résultat de calcul à une variable : **Variable← Variable1 opérateur Variable2 ;**
  Ou encore l'affectation du résultat de plusieurs calculs à une variable :
  Variable←Variable1 opérateur1 Variable2 ... opérateur-N VariableN ;

Le type de la partie droite de l'instruction doit être compatible avec le type de la variable X.

Exemple :

```
Age ← 24 ;          // Initialiser (attribuer) la valeur 24 à la variable Age.
A   ← ((A+B)*2) ;
```

Remarque :

L'instruction d'affectation permet de :

- Forcer le contenu d'une variable (initialisation de la variable).
- D'écraser la valeur contenue précédemment dans la variable.

B. Instruction de lecture

Cette instruction permet d'introduire des valeurs par l'utilisateur via un périphérique d'entrée dans des cases mémoire bien définies. La lecture est faite selon la syntaxe suivante :

Lire (variable) ; ou **Lire (variable1, variable2,... variableN) ;**

L'exécution de cette instruction consiste à affecter une valeur à la variable en prenant cette valeur sur le périphérique d'entrée. Avant l'exécution de cette instruction, la variable avait ou n'avait pas de valeur. Après, elle a la valeur prise sur le périphérique d'entrée. La valeur à introduire doit être de même type que la variable réceptrice.

Exemple : Supposons x une variable de type entier alors :

Lire(x) ; // ***Affectera une valeur de type entier taper au clavier à la variable X.***

La fin de la valeur est reconnue en tapant la touche ENTREE

Remarque :

L'instruction de lecture bloquera l'exécution de programme jusqu'à ce que la touche ENTREE soit tapée.

C. Instruction d'écriture

L'écriture permet d'afficher un résultat ou un message à l'utilisateur sur un périphérique de sortie. L'écriture est faite selon la syntaxe suivante :

Ecrire (variable);

Ecrire (var1, var2,... varN) ***;*** ***// pour afficher les valeurs des variables var1, var2,... varN***

Ecrire ('Bonjour') ; ***// pour afficher un message à l'utilisateur sur l'écran.***
// dans ce cas, le message doit être écrit entre simples quottes.

Remarque :

Il est possible de regrouper aussi un message et une variable dans la même instruction d'écriture comme suit :

Ecrire ('Résultat = ' , x1)

Exemple : supposons x=0, y=0 alors:

Ecrire (x, y+2, "bonjour") ; //***Affiche sur l'écran : 0 2 bonjour***

IX. Construction d'un algorithme simple

Un algorithme se présente en général sous la forme suivante :

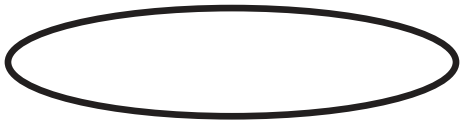
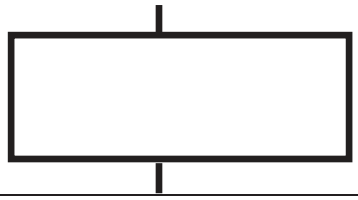
- ***Déclaration des variables :*** On décrit dans le détail les éléments que l'on va utiliser dans l'algorithme (variables, constantes et structures),
- ***Initialisation ou Entrée des données :*** On récupère les données et/ou on les initialise (par lecture ou par affectation),
- ***Traitement des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

Exemple : Écrire un algorithme qui permet de calculer la somme de deux entiers.

```
ALGORITHME CALCUL_SOMME ;
VAR
        A, B, C: ENTIER ;
DÉBUT
    ECRIRE ("ENTREZ VOTRE PREMIERE VALEUR ") ;
    LIRE (A) ;
    ECRIRE ("ENTREZ VOTRE DEUXIEME VALEUR ") ;
    LIRE (B) ;
    C ← A+B;
    ECRIRE ("LA SOMME = ", C ) ;
FIN.
```

X. Représentation d'un algorithme par un organigramme

En général, on peut représenter un algorithme sous forme structurée ou sous forme graphique. Un **organigramme** est une représentation graphique d'un algorithme. Pour cela, on utilise des symboles normalisés dont les principaux sont les suivants :

Début et fin 	On l'utilise pour commencer et pour terminer un algorithme.
Entrée / Sortie 	On l'utilise pour lire ou écrire des données.
Symbole de traitement 	On l'utilise pour le reste des opérations hors la lecture et l'écriture.
Symbole de liaison	On l'utilise pour connecter les autres symboles

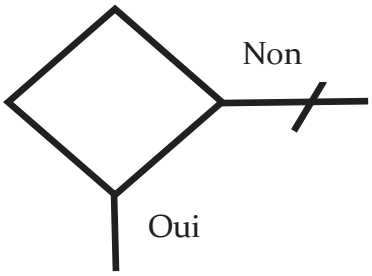
Symbole de test (branchement)	
	Choix avec condition : on l'utilise pour l'exploitation de conditions variables impliquant le choix d'une voie parmi plusieurs.

Tableau 2.1. Symboles d'un organigramme

Exemple : L'algorithme de l'exemple précédent (somme de deux entiers) peut être représenté par l'organigramme suivant :

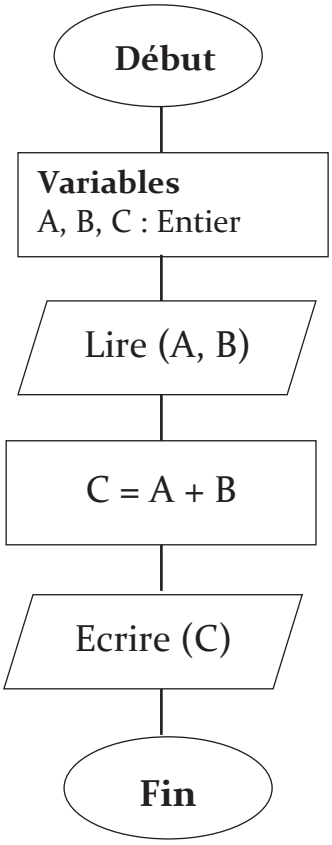


Figure 2.3 : Structure d'un organigramme

XI. Traduction en langage C

Dans notre ouvrage nous présentons la traduction des algorithmes en langage C.

Le langage C est un langage évolué et structuré. C'est un langage souple et puissant, assez proche du langage machine utilisé pour des projets tels que les systèmes d'exploitation, des applications de contrôle de processus (gestion d'entrées/sorties, applications temps réel …), des applications de traitements de textes, des graphiques ou même des compilateurs pour d'autres langages. Inventé au début des années 1970, C est devenu un des langages les plus utilisés.

Le langage C possède assez peu d'instructions, il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombre avec le compilateur.

Exemples:

- **math.h** : bibliothèque de fonctions mathématiques de base.
- **stdio.h** : bibliothèque d'entrées/sorties standard utilisée avec les unités classiques d'entrées-sorties, qui sont respectivement le clavier et l'écran.

La programmation par le langage C est basée sur :

1. La rédaction du texte du programme (codage) en respectant la syntaxe précise du langage, à l'aide d'un éditeur de texte.
2. La compilation : transformation du code C en langage machine par un compilateur C;
3. Exécution du fichier binaire obtenu.

Premier programme :

```
#include <stdio.h>
main()
{
int a,b,c;      /* Déclaration de variables */
printf (" Bonjour \n");
a = 1;          /* initialisation */
b = 2;
c = a + b;     /* affectation */
printf (" Résultat = %d \n ",c);
printf (" Au revoir !\n");
}
```

La fonction ***main()*** est la fonction principale du programme. Elle est la seule à être exécutée au lancement du programme ; elle fait appel éventuellement à d'autres fonctions. Dans l'exemple ci-dessus, elle appelle la fonction ***printf ().***

La fonction ***printf ()*** est une fonction standard définie dans le fichier ***stdio.h*** inclus en début de programme. Elle affiche une chaıne de caractères à l'′écran ainsi que certains types de variables prédéfinis.

Le corps des fonctions est toujours délimité par des **{…}**, et chaque instruction doit se terminer par un **" ; ".**

Les commentaires sont mis entre **/* … */,** ainsi le compilateur les ignore.

XI.1 Variables et Constantes en langage C

Le C est un langage typé. Cela signifie en particulier que toute variable, constante ou Fonction est d'un type précis.

En langage C, les variables doivent être déclarées en début de programme et leurs types doivent être explicités. Le tableau ci-dessous englobe les types les plus utilisés en langage C avec leurs spécificités :

Nom du Type	Description	Nombre d'octets	Domaine Min	Domaine Max
char	Caractère	*1*	-128	127
int	Entier standard signé	2	-32 768	32 767
short	entier court	2	-32 768	-32 768
long	entier long	4	-2147483648	2147483647
unsigned int	entier positif	2	0	65535
float	Réel à virgule flottante simple précision	4	$3.4 * 10^{-38}$	$3.4 * 10^{38}$
double	Réel à virgule flottante double précision	8	$1.7 * 10^{-308}$	$1.7 * 10^{308}$

Tableau 2.2. Principaux types de variables en langage C

En langage C, le type char est un cas particulier du type entier : ***Un caractère est un entier de 8 bits***

La déclaration en langage C se fait à l'aide du mot-clé correspondant au type souhaité, suivi d'un nom de variable

Exemple :

- int Var1 ; // ***Var1 est une variable de type entier.***
- float Y ; // ***Y est une variable de type réel.***
- char W ; // ***W est une variable de type caractère.***

En langage C, l'opérateur d'affectation est "=". Le langage C permet l'initialisation des variables dans la zone des déclarations:

char c; est équivalent à ***char*** c = 'A';
c = 'A';

int i; est équivalent à ***int*** i = 50;
i = 50;

Le langage C distingue les minuscules des majuscules (Exemple : a est différente de A). Les mots réservés du langage C doivent être écrits en minuscules.

La déclaration et l'initialisation d'une variable peuvent commencer par le qualificatif ***const*** qui indique que la valeur de cette variable ne pourra plus être modifiée.

```
const int N = 30;
```

Il est également possible de déclarer des constantes à l'aide de la commande du préprocesseur ***#define*** :

```
#define NMAX 10
```

Avant la compilation, le préprocesseur (une partie du compilateur) remplacera toutes les occurrences de NMAX par le chiffre 10.

XI.2 Les opérateurs en langage C

- **Les opérateurs arithmétiques**

  x + y addition
  x - y soustraction
  x * y multiplication
  x / y division
  x % y reste de la division entière (modulo)

- **Les opérateurs de comparaison**

  Les opérateurs de comparaison sont : **>** , **>=** , **<** , **<=**
  Les opérateurs d'égalité : == , **!=** .
  Attention, == est différent de l'opérateur d'affectation **=.**

- **Les opérateurs logiques**

  Les opérateurs logiques **&&** et || signifient "et" et "ou" .
  L'opérateur de négation : **!** .

Ces opérateurs donnent un résultat "***Vrai***" ou "***Faux***" selon l'expression où ils figurent. En réalité ils retournent la valeur **0** pour "Faux" et **1** pour "Vrai".

En général, toute valeur non nulle est ′évaluée comme "Vrai".

```
int a,b;
a = 2;
b = 3;
a > b;                    /* vaut 0 */
a < b;                    /* vaut 1 */
!b;                       /* vaut 0 */
a == b;                   /* vaut 0 */
b != a;                   /* vaut 1 */
a > b && a < b;           /* vaut 0 */
a > b || a < b;           /* vaut 1 */
```

- **Les opérateurs d'incrémentation et de décrémentation**

```
int i = 0;
int a;
i++;          /* equivalent à  i = i + 1 */
i--;          /*equivalent à   i = i - 1 */
```

- **Les opérateurs d'affectation**

```
a = i;          /* a prend la valeur de i */
a += i;         /* equivalent à   a = a + i */
a -= i;         /* equivalent à   a = a - i */
a *= i;         /* equivalent à   a = a * i */
a /= i;         /* equivalent à   a = a / i */
```

XI.3 Instruction de lecture en langage C

La lecture en langage C s'effectue avec l'utilisation de la fonction ***scanf***. Il s'agit d'une fonction de la librairie standard ***stdio.h*** .

La fonction ***scanf*** permet de saisir des données au clavier et de les stocker aux adresses spécifiées par les arguments de la fonction. Les variables à saisir sont formatées, le nom de la variable est précédé du symbole & désignant l'adresse de la variable. La syntaxe de scanf est la suivante :

Scanf ("%format ", & nom_de_variable)

Le tableau ci-dessous indique les Formats de saisie les plus utilisés .

Format	Type attendu
%c	Caractère
%d	int
%hd	short
% ld	long
%u	unsigned int
%f	float
% lf	double

Tableau 2.3. Principaux formats de saisie

Exemples:
```
char alpha;
int i;   float r;
scanf("%c",&alpha);          // saisie d'un caractère .
scanf("%d",&i);              // saisie d'un nombre entier en décimal .
scanf("%f",&r);              // saisie d'un nombre réel .
```

XI.4 Instruction d'écriture en langage C

L'écriture en langage C s'effectue avec l'utilisation de la fonction ***printf***. Il s'agit d'une fonction d'affichage formatée de la librairie standard ***stdio.h***, ce qui signifie que les données sont converties selon un format particulier choisi.

On utilise ***printf*** de la même manière que pour afficher un texte, sauf que l'on rajoute un symbole spécial à l'endroit où l'on veut afficher la valeur de la variable. La fonction ***printf*** exige l'utilisation de formats de sortie, avec la structure suivante:

printf ("%format",nom_de_variable);

Pour un affichage multiple, on utilise la structure suivante:

printf ("format1 format2 .... formatN",variable1,variable2, ...,variableN);

Exemples: affichage d'un texte:

printf("BONJOUR"); // **pas de retour à la ligne du curseur après l'affichage.**

printf("BONJOUR\n"); // **affichage du texte, puis retour à la ligne du curseur.**

On peut éventuellement préciser certains paramètres du format d'impression, qui sont spécifies entre le % et le caractère de conversion dans l'ordre suivant :

- ***Largeur minimale du champ d'impression*** : %10d spécifie qu'au moins 10 caractères seront réservés pour imprimer l'entier.

- ***Précision :*** %.2f signifie qu'un flottant sera imprimé avec 2 chiffres après la virgule. Lorsque la précision n'est pas spécifiée, elle correspond par défaut à 6 chiffres après la virgule. Pour une chaîne de caractères, la précision correspond au nombre de caractères imprimés : %30.4s signifie que l'on réserve un champ de 30 caractères pour imprimer la chaîne mais que seulement les 4 premiers caractères seront imprimés (suivis de 26 blancs).

Trace d'exécution :

L'exécution d'un algorithme (programme) se fait instruction par instruction selon l'ordre indiqué par l'algorithme. Elle commence à partir de la première instruction qui suit le mot clé DEBUT et se termine à la dernière instruction qui précède juste le mot clé FIN.

Au début les valeurs des variables sont inconnues « ? ». Quand une variable reçoit une valeur, cette valeur est sauvegardée jusqu'à ce qu'une autre instruction change cette valeur.

Exemple : donnez la Trace d'exécution de l'algorithme suivant

```
Algorithme trace ;
   x : entier ;
   y : entier ;
Début
      x ← 12 ;
      y ← x + 4 ;
      x ← 3 ;
Fin.
```

Instructions	x	y
(1)	12	?
(2)	12	16
(3)	3	16

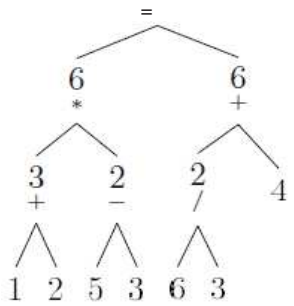
Tableau 2.4. Trace d'exécution d'un algorithme

XII. Conclusion

Nous avons présenté à travers ce chapitre les premiers pas pour construire un algorithme séquentiel simple. Par conséquent, les notions de variables, d'affectation, de lecture et d'écriture sont présentées algorithmiquement et techniquement avec le langage de programmation C.

Série d'exercices 2

Exercice 2.1 : Combien ça vaut ?

Évaluer les expressions suivantes :	**Exemple** : (1 + 2) * (5 - 3) = ((6/3) + 4)
• (2 + 4) / (3 * (5 - 3)) • 2 + 4 / 3 * 5 – 3 • (9/3) - (12 mod 5) * (5 div 3) + 1 • (17 mod 3) = (12 div 5) • $(x \geq 3) \wedge ((y \leq 5) \vee (y \geq 8))$ pour x = 1 et y = 3 • $((x \geq 3) \wedge (y \leq 5)) \vee (y \geq 8)$ pour x = 1 et y = 3	**Vrai** 

Exercice 2.2 : Quel est mon type ?

Pour chaque expression ci-dessous, donner le type et le résultat, sinon dire si elle n'est pas bien formée.

• 5 * 2 • 6 + 7.4 • '6' + 7.4 • (3 * 4) + 5 = 32 • '0' • 'zero' • 3 > 10 ∨ 10 < 3	**Exemple :** L'expression (1 + 3.6) > (4 - 2) est évaluée comme suit : $(\underbrace{\underbrace{1}_{Entier} + \underbrace{3.6}_{Réel}}_{Réel\ 4.6} > \underbrace{\underbrace{4}_{Entier} - \underbrace{2}_{Entier}}_{Entier\ 2})$ *Booléen Vrai*

Exercice 2.3 : Combien ça vaut ?

Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

Algorithme EXO2 ; **Variables** A, B, C : Entier ; **Début** A ← 5 ; B ← 3 ; C ← A + B ; A ← 2 ; C ← B – A ; **Fin**	**Algorithme** EXO3 ; **Variables** A, B, C : Entier ; **Début** A ← 3 ; B ← 10 ; C ← A + B ; B ← A + B ; A ← C ; B ← B * 2 ; **Fin**

Exercice 2.4 : Que fait mon algorithme ?

Compléter les différents commentaires devant chaque instruction.

Algorithme 2 ;
Variables
X, Y : Entier // **deux variables de type Entier**
Z : Réel // **une variable de type réel**
Début
1: **Écrire** ('Donner la valeur de X') // ....
2: **Lire** (X) // ....
3: **Écrire** ('Donner la valeur de Y') // ....
4: **Lire** (Y) // ....
5: Z ← X=Y // ....
6: **Écrire** ('La valeur de Z est :', Z) // ....
Fin.

Exercice 2.5 : Le carré

Ecrire un algorithme qui demande un nombre à l'utilisateur, puis calcule et affiche le carré de ce nombre.

Traduire cet algorithme en programme C.

Exercice 2.6 : La résistance

Ecrire un algorithme qui calcule et affiche la résistance d'un composant électronique en utilisant la loi d'Ohm :

$$U = R \times I \quad \text{avec} \quad \begin{cases} U : \textit{Tension en } V \\ R : \textit{Resistance en } \Omega \\ I : \textit{Intensite en } A \end{cases}$$

Traduire cet algorithme en programme C.

Chapitre 3

Les structures conditionnelles

Sommaire

Objectif

L'objectif de ce chapitre est de construire des algorithmes comportant des traitements conditionnels.

I. Introduction

Une structure séquentielle est une suite d'instructions. Les instructions sont exécutées dans l'ordre l'une après l'autre.

Syntaxe :

```
Début
        Instruction 1 ;
        Instruction 2 ;
        …..
        Instruction n ;
Fin.
```

Exemple : l'algorithme qui affiche l'inverse d'un nombre réel.

```
ALGORITHME  INVERSE ;
VAR
        X, Y : REEL ;
DEBUT
        ECRIRE ("ENTREZ UNE VALEUR SVP ") ;
        LIRE (X) ;
        Y ← 1 /X ;
        ECRIRE ("L'INVERSE DE " , X, "EST", Y) ;
FIN.
```

Généralement, un algorithme ne comprend pas que des instructions séquentielles, mais il comprend aussi des instructions dites conditionnelles. L'algorithme ci-dessus calcule l'inverse d'un nombre réel. Si l'utilisateur tape la valeur 0 il y a une erreur d'exécution (division par 0), ce problème ne peut pas être résolu que par l'utilisation des structures conditionnelles.

Une instruction conditionnelle permet à un programme de modifier son traitement en fonction d'une condition. Les structures conditionnelles permettent de déterminer l'ordre dans lequel les instructions sont exécutées.

Dans les structures conditionnelles on se base sur ce qu'on appelle ***prédicat*** ou ***condition***. Ce dernier est un énoncé ou proposition qui peut être ***vrai*** ou ***faux.***

Dans ce qui suit, nous allons étudier les quatre formes d'instructions conditionnelles qui sont:

- Les structures conditionnelles simples
- Les structures conditionnelles composées
- Les structures conditionnelles imbriquées
- Les structures conditionnelles de choix multiple
- Les branchements

II. Structure conditionnelle simple : L'instruction « if..... »

Une structure de contrôle conditionnelle est dite à forme simple (la plus basique) lorsque le traitement dépend d'une condition. Si la condition est évaluée à « vrai », le traitement est exécuté. Dans cette structure, la non-satisfaction de la condition ne correspond à aucune action à faire.

Cette structure est subdivisée en deux parties : la condition et l'action.

Syntaxe :

Algorithme	langage C

SI <Condition> **ALORS**
< Bloc d'actions >
FINSI
	if (condition)
	{
	< Bloc d'actions >
	}

Où : <Condition> est une expression de condition dont l'évaluation donne une valeur logique, et le < Bloc d'actions > est un groupe d' instructions élémentaires.

- Si la condition est vérifiée (condition = vrai), les instructions du < Bloc d'actions > sont exécutées et on continue l'exécution des actions (instructions) situées après le Finsi.

- Si la condition n'est pas vérifiée (condition = faux), la partie < Bloc d'actions > à l'intérieur du Si n'est pas exécuté et on poursuit l'exécution de l'algorithme directement à partir de l'instruction qui suit le FinSi.

En langage C, si on ne met pas les accolades, le compilateur va considérer uniquement la première instruction comme instruction subordonnée à la structure if.

Exemple :

SI ((A-B) * C) = 12 **ALORS**
B ← 3;
C ← (X+2+B);
FINSI;

	<condition> = VRAI			**<condition> = FAUX**	
	Avant l'action	**Après l'action**		**Avant l'action**	**Après l'action**
A	5	5		2	2
B	2	3		2	2
C	4	8		4	4
X	3	3		5	5

Exemple en langage C:

```
if ( note >= 10)
   {
      printf (" vous êtes admis  ") ;
   }
```

Le formalisme de cette structure dans un organigramme est comme suit :

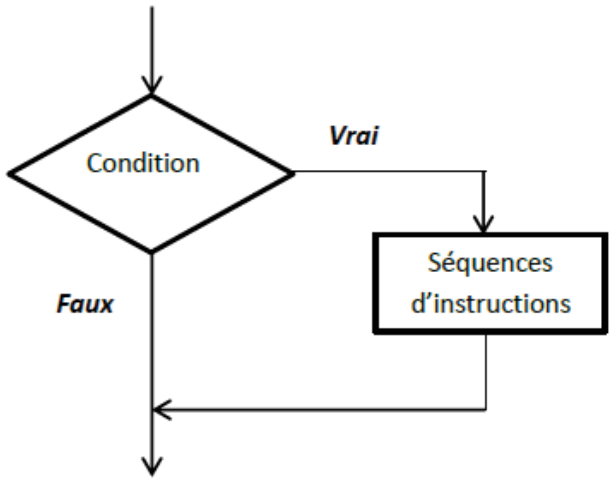


Figure 3.1 : Structure conditionnelle simple

III. Structure conditionnelle composée : L'instruction « if ... else »

Une structure de contrôle conditionnelle est dite composée ou à forme alternative lorsque le traitement dépend d'une condition à deux états: Si la condition est évaluée à « vrai », le premier traitement est exécuté, si la condition est évaluée à « faux », le second traitement est exécuté.

Syntaxe :

Algorithme	langage C

SI <Condition> **ALORS**
< Bloc d'actions 1 >
SINON
< Bloc d'actions 2 >
FINSI
	if (Condition)
	{
	Bloc d'actions 1
	}
	else
	{
	Bloc d'actions 2
	}

Elle s'exécute comme suit :

- Si la condition est vérifiée, les instructions du < Bloc d'actions1 > sont exécutées puis poursuite de l'exécution de l'algorithme à partir de l'instruction qui suit le Finsi.

- Si la condition n'est pas vérifiée, les instructions du < Bloc d'actions2 > sont exécutées puis poursuite de l'exécution de l'algorithme à partir de l'instruction qui suit le Finsi.

Exemple :

SI (A+B = 0) **ALORS**
A ⟵ (C-2);
SINON
B ⟵ (X*3);
A ⟵ (B-4);
FINSI;

	<condition> = VRAI (exécuté action(s)1)			<condition> = FAUX (exécuté Action (s)2)	
	Avant l'action	**Après l'action**		**Avant l'action**	**Après l'action**
A	3	0		5	2
B	-3	-3		2	6
C	2	2		4	4
X	1	1		2	2

Exemple en langage C:

```
int main ()
        {
        int valeur1 ;
        int valeur2 ;
        printf (" Entrez une 1ere valeur : ");
        scanf ("%d" ,& valeur1 );
        printf (" Entrez 2eme valeur : ");
        scanf ("%d" ,& valeur2 );
        if ( valeur1 < valeur2 )
                printf ("%d > %d\n",valeur2 ,
        valeur1 );
        else
                printf ("%d >= %d\n",valeur1 ,
        valeur2 );
        return 0;
        }
```

Le formalisme de cette structure dans un organigramme est comme suit :

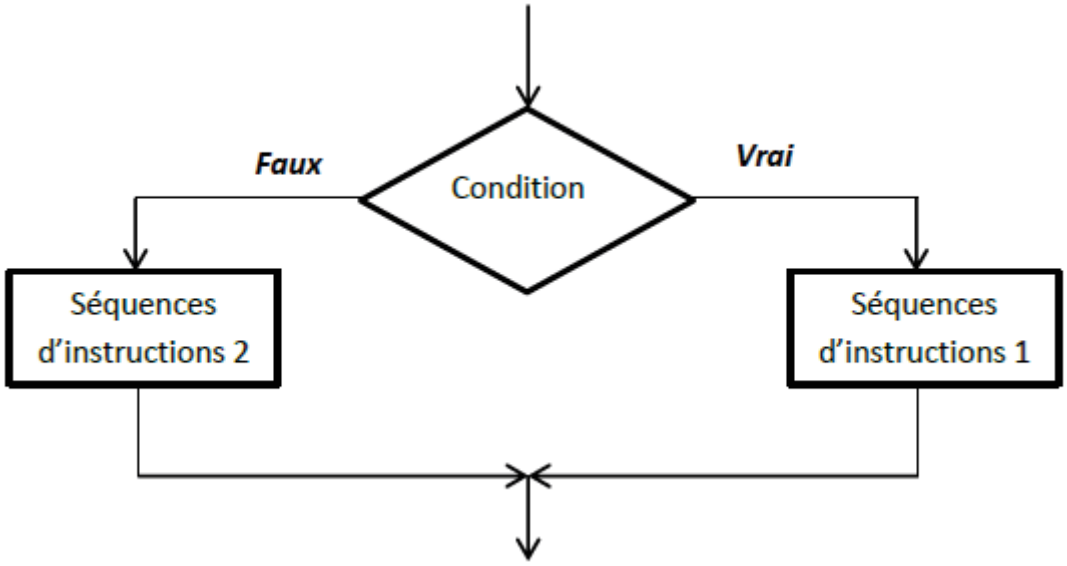


Figure 3.2 Structure conditionnelle composée

IV. Structures conditionnelles imbriquées

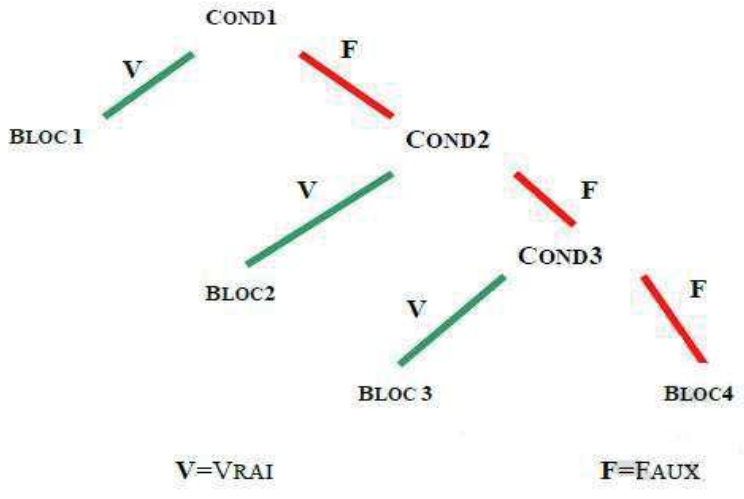
Il faut bien comprendre que les blocs d'instructions de ***(SI)*** et de ***(SINON)*** sont des séquences d'instructions. Ces blocs peuvent donc comporter des structures conditionnelles.

Par exemple, la deuxième branche d'une structure conditionnelle peut déboucher sur une nouvelle structure à deux branches, créant ainsi une structure conditionnelle à trois branches.

Syntaxe :

```
SI    <Cond 1 >   ALORS
                      < Bloc d'actions 1 >
SINON   SI <Cond 2 > ALORS
                   < Bloc d'actions 2 >
        SINON  SI  <Cond 3 >  ALORS
                          < Bloc d'actions 3 >
                SINON
                          < Bloc d'actions 4 >
                FINSI
          FINSI
FINSI
```

Fonctionnement :



Exemple en langage C:

```
int main ()
 {
int a = 0;
printf (" Saisie de a : ");
scanf ("%d" ,&a);
if (a < 0)    /* Strictement negative ? */
printf ("la variable a est negative .\n");
else
{
if (a > 0) /* Strictement positive ? */
printf ("la variable a est positive \n");
else /* Sinon a est nulle */
printf ("la variable a est nulle \n");
}
return 0;
  }
```

V. Structure conditionnelle de choix multiple : L'instruction « switch »

C'est une extension du si ... alors ... sinon. Elle permet une programmation plus claire en évitant une trop grande imbrication de Si successifs. Une structure de contrôle conditionnelle est dite à choix multiple lorsque le traitement dépend de la valeur que prendra le sélecteur (variable).

Cette structure conditionnelle est appelée aussi ***sélective*** car elle sélectionne entre plusieurs choix à la fois, et non entre deux choix alternatifs (permet de faire plusieurs conditions sur une même variable). La structure globale est la suivante :

Syntaxe :

Algorithme	langage C
SELON Variable **FAIRE** **Cas val 1 :** liste d'instructions 1 **Cas val 2 :** liste d'instructions 2 ..... **Cas val n :** liste d'instructions n **SINON** traitement par défaut ; **FIN SELON**	**switch (** Variable **) {** **case Valeur1 : // Liste d' instructions ;** **break ;** **case Valeur2 :// Liste d' instructions ;** **break ;** **.....** **case Valeurs n ... : // Liste d' instructions;** **break ;** **default : // Liste d'instructions ;** **break ;** **}**

Le sélecteur doit être de type scalaire (dont le type est un entier, un caractère, un booléen, mais pas un réel ni une chaîne de caractères).

Les **CAS X** sont des constantes ordinales du même type que le sélecteur (sont des valeurs qui peuvent être prise par la variable sélecteur).

CAS X peut être :

- une valeur
- une suite de valeurs : 1, 3, 5, 7, 9
- une fourchette : 0 à 9
- une plage : >= 10

Si la valeur de « Variable » est égale à l'une des **CAS X**, la liste d'instructions correspondant est exécutée puis le programme sort de la structure. Sinon la liste d'instructions correspondant à « Sinon ou default » est exécutée. Au cas où l'action sinon ou default n'existe pas alors aucune action n'est exécutée.

Exemple en langage C:

```
void main ()
{
int choixMenu ;
printf (" ===== Menu =====\ n\n");
printf ("1. Royal Cheese \n");
printf ("2. Big Burger \n");
printf ("3. Complet Poulet \n");
printf ("4. Panini Thon \n");
printf ("\n Votre choix ? ");
scanf ("%d", & choixMenu );    /* Saisie du choix de l' utilisateur */
printf ("\n");
switch ( choixMenu ) /* Tester le choix de l' utilisateur */
{
case 1: printf (" Vous avez choisi un Royal Cheese !");
break ;
case 2: printf (" Vous avez choisi un Big Burger !");
break ;
case 3: printf (" Vous avez choisi un Complet Poulet !");
break ;
case 4: printf (" Vous avez choisi un Panini Thon !");
break ;
default : printf (" Choix incorrect . Vous ne mangerez rien !");
break ;
}
return 0;
}
```

Remarques :

- L'instruction default est facultative.
- Le mot clé break indique la sortie de la structure conditionnelle. N'oubliez pas d'insérer des instructions break entre chaque test, ce genre d'oubli est difficile à détecter car aucune erreur n'est signalée
- Ceci peut être utilisé judicieusement afin de faire exécuter les mêmes instructions pour différentes valeurs consécutives

VI. Le branchement

A priori, dans un programme, les instructions sont exécutées séquentiellement, c'est à dire dans l'ordre où elles apparaissent. Mais, avec les instructions de branchement nous pouvons casser cette règle en permettant à un programme d'interrompre un bloc d'instruction pour sortir ou pour passer à un autre bloc. En langage C, cette idée peut être réalisée à travers les instructions : ***break, continu* et *goto***.

A. L'instruction break

L'instruction break permet d'interrompre le déroulement d'une boucle, et passe à la première instruction qui suit la boucle. En cas de boucles imbriquées, break fait sortir de la boucle la plus interne. Par exemple, le programme suivant

```
#include<stdio.h>
int main ()
{
int i;
for (i = 0; i < 5; i++)
{
printf("i = %d\n",i);
if (i == 3)
break;}
printf("valeur de i à la sortie de la boucle = %d\n",i);
}
```

Le programme va sortir de la boucle for avec i=3, c'est-à-dire avant l'expiration de la boucle en imprimant :

i = 1
i = 2
i = 3
valeur de i à la sortie de la boucle = 3

B. L'instruction continue

L'instruction ***continue*** permet de passer directement au tour de la boucle suivante, sans exécuter les autres instructions de la boucle. Ainsi le programme :

```
#include<stdio.h>
int main ()
{int i;
for (i = 0; i < 5; i++)
{
if (i == 3)
continue;
printf("i = %d\n",i);
}
printf("valeur de i à la sortie de la boucle = %d\n",i);
}
```

imprime

i = 0
i = 1
i = 2
i = 4
valeur de i à la sortie de la boucle = 5

C. L'instruction goto

L'instruction ***goto*** permet d'effectuer un saut jusqu'à l'instruction étiquette correspondant. Elle est à proscrire de tout programme C digne de ce nom. Ainsi le programme :

```
#include<stdio.h>
int main ()
{
int i;
for (i=1 ;i<=10 ;i++)
{
printf("i = %d\n",i);
if(i==3)goto sortie ;
}
sortie: printf ("fin de tour avec i=%d\n",i) ;
return 0 ;
}
```

imprime :

i = 1
i = 2
i = 3
fin de tour avec i=3

VII. Une façon plus courte de faire un test :

Il est possible de faire un test avec une structure beaucoup moins lourde en langage C grâce à la structure suivante :

(condition) ? instruction si vrai : instruction si faux

Remarques :

- La condition doit être entre des parenthèses
- Lorsque la condition est vraie, l'instruction de gauche est exécutée
- Lorsque la condition est fausse, l'instruction de droite est exécutée
- En plus d'être exécutée, la structure ? : renvoie la valeur résultant de

Exemple :

```
(moyenne >=10) ? printf (" Admis ") : printf (" Ajourne ");

admis = (( moyenne >=10) ? 1 : 0);
```

VIII. Conclusion

Dans ce chapitre, nous avons présenté les structures conditionnelles, qui vont nous permettre de faire des tests et d'exécuter une ou plusieurs instructions dans un cas, d'autres instructions dans un autre cas. Selon la situation et le positionnement de problème, le développeur peut choisir entre plusieurs structures conditionnelles à savoir : les structures simples, les structures composées, les structures imbriquées, les structures de choix multiples ou les branchements.

Série d'exercices 3

Exercice 3.1 : Pair ou impair ?

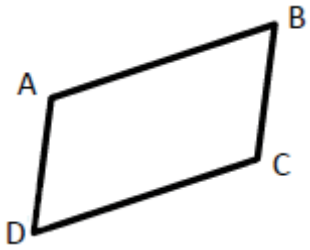
Écrire un algorithme/programme qui demande à l'utilisateur un nombre entier N et affiche si le nombre N est pair ou impair suivant sa parité.

Exercice 3.2 : Parallélogramme ?

Dans un repaire Euclidien, quatre points du plan A, B, C et D sont connus par leurs coordonnées. Ecrire un algorithme/programme qui permet de déterminer si le quadrilatère ABCD est un parallélogramme ou non.

Un parallélogramme est distingué par deux choses :

- (XB –XC) = (XA–XD) et
- (YB – XA) = (YC – YD)



Exercice 3.3 : Maximum

Écrivez un algorithme/programme qui lit trois variables au clavier et affiche le maximum des trois..

Exercice 3.4 : Equation second degré

Ecrire un algorithme/programme permettant de résoudre dans R une équation du second degré de la forme $ax^2 + bx + c = 0$, en envisageant tous les cas particuliers.

Exercice 3.5 : Jour de la semaine

Ecrivez un algorithme/programme qui permet de saisir un entier entre 1 et 7 et affiche en toutes lettres le nom du jour de la semaine qui lui correspond.

Exercice 3.5 : Catégories

Proposer deux variantes d'algorithme/programme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :

- "Poussin" de 6 à 7 ans
- "Pupille" de 8 à 9 ans
- "Minime" de 10 à 11 ans
- "Cadet" après 12 ans

Chapitre 4

Les structures itératives (Les boucles)

Sommaire

Objectif

L'objectif de ce chapitre est de construire des algorithmes comportant des traitements itératifs, qui consiste à répéter les mêmes instructions un certain nombre de fois à travers la boucle tant que, la boucle répéter et la boucle pour.

I. Introduction

Il arrive souvent des situations où on veut répéter une instruction ou un bloc d'instructions plusieurs fois. Pour ce faire on utile les structures répétitivités (boucles).

La plupart des langages de programmation proposent trois types de structures répétitives courantes :

- La boucle Pour
- La boucle Tant que
- La boucle Répéter

II. La boucle Pour (La boucle « for »)

La boucle ***Pour*** permet de répéter un bloc d'instructions un nombre de fois donné, dans ce cas on utilise un compteur et on s'arrête lorsque le compteur atteint sa valeur finale connue au préalable. Cette boucle a la structure suivante :

Syntaxe :

POUR I= *valeur_initiale* **à** *valeur_finale* **pas**=*val* **FAIRE**

Ensemble des instructions à répéter

FINpour

Propriétés en langage algorithmique :

- La boucle POUR est utilisée lorsque le nombre de répétition est connu à l'avance.
- Pour chaque valeur de la variable de contrôle qui varie de la *valeur initiale* à *la valeur finale* avec un pas égale à *val*, le bloc sera exécuté.
- Le bloc est répété ***(valeur finale - valeur initiale+1)*** fois.
- La sortie de cette boucle s'effectuer lorsque le nombre souhaité de répétition est atteint.
- Dans cette boucle l'incrémentation (ajouter 1) est faite automatiquement.
- Lorsque le pas est Omis, il est supposé égal à (+1).

Syntaxe en Langage C :

```
for ( initialisation ; condition ; pas )
{
// Bloc d'instructions
}
```

Propriétés en langage C :

Il y a trois instructions condensées, chacune séparée par un point-virgule :

- La première est l'**initialisation** : cette première instruction est utilisée pour préparer notre variable compteur.
- La seconde est la **condition :** c'est la condition qui dit si la boucle doit être répétée ou non. Tant que la condition est vraie, la boucle for continue. Si la condition est fausse, on sort de la boucle et on continue avec l'instruction qui suit l'accolade fermante.
- Enfin, il y a **l'incrémentation :** cette dernière instruction est exécutée à la fin de chaque tour de boucle pour mettre à jour la variable compteur.

Exemple : Calculer la somme de N entiers

Algorithme	langage C

Algorithme Somme **;**
Var
N, X, SOM, i **: entier ;**
Debut
Lire (N) ;
SOM ← 0 ;
Pour i = 1 **à** N **Faire**
Lire (X) ;
SOM← SOM + X ;
Fin Pour
Ecrire ("LA SOMME =", SOM) ;
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, i;
	printf("donner le nombre des entiers à additionner: ");
	scanf("%d",&N);
	SOM=0;
	for (i =1; i <N+1; i ++)
	{
	printf ("donner la valeur du nombre %d : " , i);
	scanf("%d", &X);
	SOM=SOM+X;
	}
	printf ("LA SOMME = %d \n", SOM);
	return o;
	}

La plupart du temps on fera une **incrémentation**, mais on peut aussi faire une **décrémentation (variable --)** ou encore n'importe quelle autre opération (variable += 2 ; pour avancer de 2 en 2 par exemple).

Il est fortement déconseillé de modifier la valeur du compteur (et/ou la valeur de finale) à l'intérieur de la boucle. En effet, une telle action :

- Perturbe le nombre d'itérations prévu par la boucle Pour
- Présente le risque d'aboutir à une boucle infinie

III. La boucle Tant que (La boucle « While »)

Une boucle ***Tant que*** permet de répéter plusieurs fois le même bloc d'instructions tant qu'une certaine condition reste vraie

Syntaxe :

```
TANT QUE COND FAIRE
    Bloc d'instructions
FIN TANT QUE
```

Propriétés en langage algorithmique :

- Le nombre de répétitions n'est pas connu à l'avance il dépend de la valeur de la condition **COND.**
- Au début de chaque itération, la condition est évaluée.
- Si **COND** est Vraie le bloc est exécuté sinon on sort de la boucle
- Le Bloc est répété tant que **COND** est Vraie
- Si **COND** est fausse au début on n'exécute aucune instruction. Donc on peut avoir des boucles qui ne sont jamais exécutées.

Syntaxe en Langage C:

```
while ( Condition )
{
  // Bloc d'instructions
}
```

Propriétés en langage C :

- La condition (dite condition d'arrêt) est évaluée avant chaque itération
- Si la condition est vraie, on exécute le bloc d'instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution, ...
- Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui se trouve juste après l'accolade fermante « } ».

Exemple 1 : On souhaite écrire un programme qui permet de contrôler la saisie d'un entier positif.

```
int entierPositif = 0;
while ( entierPositif <=0)
{
printf (" Tapez un entier positif ! ");
scanf ("%d", & entierPositif );
}
```

Exemple 2 : Calculer la somme de N entiers

Algorithme	langage C

Algorithme Somme **;**
Var
N, X, SOM, i **: entier ;**
Debut
Lire (N) ;
SOM ← 0 ;
i ← 1 ;
 	
Tant que (I <= N) Faire
Lire (X) ;
SOM← SOM + X ;
i←i+1 ;
FinTantque
Ecrire ("LA SOMME =", SOM) ;
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, i;
	printf("donner le nombre des entiers à additionner: ");
	scanf("%d",&N);
	SOM=0;
	i = 1 ;
	while (i <= N)
	{
	printf ("donner la valeur du nombre %d : " , i);
	scanf("%d", &X);
	SOM=SOM+X;
	i++ .
	}
	printf ("LA SOMME = %d \n", SOM);
	return o;
	}

IV. La boucle Répéter (La boucle « do...while »)

Cette structure est très similaire à la structure ***Tant que*** du fait que leurs exécutions dépendent d'une condition. Elle permet de répéter une série d'instructions jusqu'à la vérification d'une certaine condition.

La différence entre ***Tant que*** et ***Répéter*** est que la boucle ***Tant que*** peut ne jamais être exécutée car la condition peut être non vérifiée au démarrage alors que le contenu de la boucle ***Répéter*** est exécuté au moins une fois pour arriver à la condition.

Syntaxe :

Répéter
Bloc d'instructions
jusqu'à COND

Propriétés en langage algorithmique:

- Le Bloc d'instructions est répété jusqu'à ce que la condition **COND** égale à Vraie.
- En fin de chaque itération l'expression logique **COND** est évaluée.
- Avec cette boucle le Bloc est répété au moins une seule fois.

Syntaxe en Langage C :

```
do
{
  // Bloc d'instructions
} while ( Condition );
```

Propriétés en langage C :

- La boucle « do...while » est très similaire à while
- La seule chose qui change par rapport à while, c'est la position de la condition. Au lieu d'être au début de la boucle, la condition est à la fin.
- Cette boucle s'exécutera donc toujours au moins une fois.
- Dans la boucle « do...while », n'oubliez pas de mettre un point-virgule à la fin.

Exemple : Calculer la somme de N entiers

Algorithme	langage C

Algorithme Somme **;**
Var
N, X, SOM, i **: entier ;**
Debut
Lire (N) ;
SOM ← 0 ;
i ← 1 ;
 	
Repeter
Lire (X) ;
SOM← SOM + X ;
i←i+1 ;
Jusqu'à (i > N)
Ecrire ("LA SOMME =", SOM) ;
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, i;
	printf("donner le nombre des entiers à additionner: ");
	scanf("%d",&N);
	SOM=0;
	i = 1 ;
	do
	{
	printf ("donner la valeur du nombre %d : " , i);
	scanf("%d", &X);
	SOM=SOM+X;
	i++ .
	} **while** (I <= N);
	printf ("LA SOMME = %d \n", SOM);
	return o;
	}

V. Les boucles imbriquées

Le bloc d'instructions d'une boucle peut contenir lui-même une autre boucle. C'est ce qu'on appelle des boucles imbriquées

Le principe des boucles imbriquées est :

1. On rentre dans la boucle mère (qui englobe la deuxième boucle).
2. On rentre dans la boucle inclue et on la parcours jusqu'à arriver à sa condition de sortie.
3. On sort de la boucle inclue et on revient donc au niveau de la boucle mère.
4. On itère sur la boucle englobant et l'on revient donc sur la boucle inclue.
5. Et ainsi de suite.

Exemple 1 :

```
int i;
int j=1;
for (i = 1 ; i <= 3 ; i++)
 {
    j=1
    while (j <= 4)
    {
     printf ("i=%d et j=%d!\n", i, j);
     j++;
    }
 }
```

Historique d'exécution

Inst.	i	j	Affichage
1	1	1	i=1 et j=1
2	1	2	i=1 et j=2
3	1	3	i=1 et j=3
4	1	4	i=1 et j=4
5	2	1	i=2 et j=1
6	2	2	i=2 et j=2
7	2	3	i=2 et j=3
8	2	4	i=2 et j=4
9	3	1	i=3 et j=1
10	3	2	i=3 et j=2
11	3	3	i=3 et j=3
12	3	4	i=3 et j=4

Exemple 2 : Programme d'affichage des tables de multiplication de 1 à 9

Algorithme	langage C

Algorithme Table_Multipliction ;
Var
i, j, resultat : entier ;
Pour i Allant de 1 A 9 Faire
Pour j Allant de 10 A Faire
resultat ← i*j ;
Ecrire (i,"*",j, "=",resultat) ;
FinPour
Fin pour
Fin.
	#include<stdio.h>
	int main ()
	{ int i, j;
	int resultat = 0;
	for (i = 1; i < 10; i++)
	for (j = 1; j <= 10; j++)
	{
	resultat = i * j;
	printf ("%d x %d = %d\n", i,j,resultat);
	}
	return 0;
	}

Quelle boucle puis-je utiliser pour mon programme ?

Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser :

- Si le bloc d'instructions ne doit pas être exécuté si la condition est fausse, alors utilisez while ou for.
- Si le bloc d'instructions doit être exécuté au moins une fois, alors utilisez do - while.
- Si le nombre d'exécutions du bloc d'instructions est connu à l'avance alors utilisez de préférence for.
- Si le bloc d'instructions doit être exécuté aussi longtemps qu'une condition extérieure est vraie alors utilisez while.

Le choix entre for et while n'est souvent qu'une question de préférence ou d'habitudes.

VI. Conclusion

Les structures répétitives, également appelées structures itératives ou encore boucles, sont introduites dans ce chapitre. Techniquement, si le nombre de répétitions est connu à l'avance on préfère utiliser la boucle for. Sinon, dans le cas où le nombre de répétition est imprévisible et dépend de l'évaluation d'une condition il faut utiliser un while ou un do..while.

Série d'exercices 4

Exercice 4.1 : **Diviseurs**

Ecrire un algorithme\programme qui lit un entier positif N puis affiche tous ses diviseurs.

Exercice 4.2 : **Chiffre exacte**

Ecrire un algorithme\programme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Exercice 4.3 : **Factorielle**

Écrire un algorithme\programme qui calcule la factorielle N! d'un entier N .

(Rappel : N! = 1 × 2 × · · · × *n*).

Exercice 4.4 : **Emprunt**

Un individu a emprunté à un ami une somme de 25000 Dinars (prêt sans intérêts). Pour rembourser son ami, il prévoit de lui remettre 1100 Dinars par mois. Il se demande quel sera le nombre de mois pour le remboursement et quel est le montant qui reste dans le dernier mois. Écrire un algorithme\programme pour résoudre ce problème.

Exercice 4.5 : **Série harmonique**

Calculez la somme des N premiers termes de la série harmonique S ; n est un entier positif lu à partir du clavier .

$$S = 1 + 1/2 + 1/3 + ... + 1/N .$$

Exercice 4.6 : **Fonction exponentielle**

Ecrire l'algorithme approprié pour calculer l'exponentielle d'un nombre X lu au clavier:

$$e^x = 1 + X + \frac{X^2}{2!} + \ldots\ldots\ldots + \frac{X^n}{n!}$$

Chapitre 5

Les tableaux et Les chaînes de caractères

Sommaire

Objectif

- Comprendre l'utilisation du type tableau.
- Manipuler des tableaux à une ou deux dimensions.
- Manipuler des chaînes de caractères.
- Manipuler les fonctions sur les chaines de caractères.

I. Introduction

Jusqu'à présent, nous n'avons utilisé que des variables de type prédéfini simple (entier, réel, caractère ou logique). Souvent, il est nécessaire de mémoriser une suite de valeurs numériques, de mots ou de phrases dans des variables. Mais, utiliser autant de variables n'est réellement pas rationnel. Le plus pratique est de créer des tableaux ou des chaînes de caractères, ce qui allégera les déclarations des variables et simplifiera leur lecture, leur écriture et leur accès.

Dans le présent chapitre, nous allons voir un autre type de données, c'est le type structuré "Tableau" qui nous permet de regrouper des données de même type en mémoire, ainsi que le type " chaînes de caractères".

II. Le type tableau

Exemple introductif :

Écrire un programme permettant de stocker les 200 notes du S1 des étudiants en première année MI puis les afficher avec la moyenne générale de la promotion.

Solution :

```
void main ()
{
int note = 0, somme = 0, moyenne = 0, i= 0;
for (i=0 ; i <200  ; i++)
{
printf (" donnez la note numero %d : ", i+1) ;
scanf ("%d", & note ) ;
somme +=M;
}
moyenne = somme /200 ;
printf ("La moyenne générale est : %d", moyenne );
}
```

Supposons que nous souhaitions déterminer, à partir de ces 200 notes fournies en données, combien d'étudiants ont une note supérieure à la moyenne de la classe. Pour parvenir à un tel résultat, nous devons :

- Déterminer la moyenne des 200 notes, ce qui demande de les lire toutes.
- Déterminer combien, parmi ces 200 notes, sont supérieures à la moyenne précédemment obtenue.

Vous constatez que si nous ne voulons pas être obligé de demander deux fois les notes à l'utilisateur, il nous faut les conserver en mémoire. Pour ce faire, il paraît peu raisonnable de prévoir 200 variables différentes (méthode qui, de toute manière, serait difficilement transposable à un nombre important de notes). Le type ***tableau*** va nous offrir une solution convenable à ce problème, à savoir :

- Par des déclarations appropriées, nous choisirons un nom de variable unique (par exemple *Notes*) pour repérer notre ensemble de valeurs (les notes d'un étudiant).
- Nous pourrons accéder individuellement à chacune des valeurs de ce tableau, en la repérant par un indice (ou index) précisant sa position dans le tableau.

Définitions 5.1 : Tableau

- Un tableau d'ordre n, peut être défini comme une suite de n cases mémoire consécutives (les unes à coté des autres) possédant un nom global et tel que chaque élément est caractérisé par sa position par rapport au premier arbitrairement de rang 1.

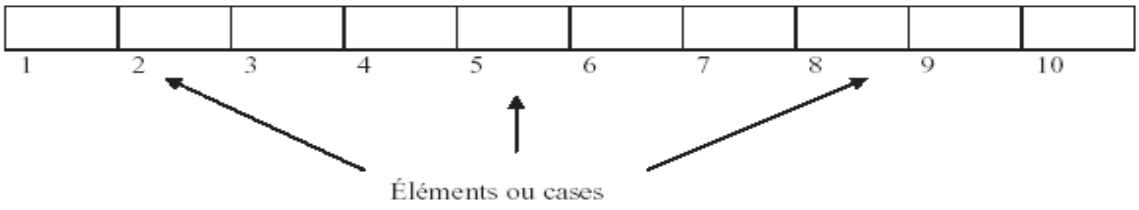
- Le nombre qui, au sein d'un tableau, sert à repérer chaque valeur s'appelle l'indice.

- Un tableau est une structure de données permettant de ranger un nombre fini d'éléments de même type. Un tableau est caractérisé par :
  - ✓ Son nom
  - ✓ Sa dimension (longueur)
  - ✓ Son type (Peut être l'un des types vus précédemment).

- Pour accéder à chaque élément du tableau, il faut écrire le nom du tableau suivi de l'indice de l'élément concerné entre crochets.

Schématiquement un tableau T de 10 éléments peut être représenté comme suit :

Tableau T



Où T : est le nom du tableau
T[1] : réfère le contenu du 1er élément du tableau T.
T[2] : réfère le contenu du 2ème élément du tableau T.
Plus généralement, si i est une variable scalaire alors T[i] réfère le contenu du ième élément du tableau T.

Pour bien utiliser un tableau on doit maitriser les tâches de déclaration, d'accès, de lecture et d'affichage.

A. Déclaration d'un tableau

La déclaration d'un tableau permet de préciser à l'ordinateur le nom du tableau ainsi que le type des éléments qui seront contenus dans le tableau. Chaque tableau à une dimension et peut être déclaré comme suit :

Syntaxe :

Algorithme	langage C
var Nom_Tableau : tableau [1.. nbr d'éléments] : <type du tableau>	Type Nom_Tableau [nbr d'éléments];

Où ***nbr d'éléments*** est le nombre des éléments dans le tableau (taille), et ***type du tableau*** indique le type des valeurs qui seront contenues dans les cases du tableau.

Un tableau doit avoir une taille fixe connue à la compilation. Cette taille peut être un nombre ou une variable constante, mais pas une variable.

Exemple :

Var
Notes : tableau [1..10] : réel ;
ou bien **// Note : est tableau de 10 éléments réels.**
Notes : tableau [10] : réel

En langage C: float Notes [10] ;

Notes	11.75	12.5	10.5	7	9.25	17	8.75	13.00	4.75	16

On déclare ici un tableau contenant 10 valeurs de type float. "[10]" représente le nombre de cases du tableau, on aura donc ici un tableau de type float de 10 cases, ces cases étant remplies de données de type **float uniquement**.

En langage C, un tableau commence à l'indice numéro 0. Notre tableau Notes de 10 float a donc les indices 0, 1, 2,... et 9. Il n'y a pas d'indice 10 dans un tableau de 10 cases. C'est une source d'erreurs très courantes pour les débutants.

B. Manipulation des composantes d'un tableau

Les éléments d'un tableau (Notes par exemple) peuvent être manipulés comme toutes autres types de variables comme suit :

```
Notes [ 1 ]  ← x                    // x est une variable avec une valeur définie
Notes [ 2 ] ← 3.5                   // mémorise le réel 3.5 dans la seconde case
Notes [ 3 ] ← Notes[1] + Notes[2]
Lire (Notes [1])                    // range la valeur saisie dans la 1ere case
Écrire (Notes [3])                  // affiche le réel contenu dans la 3eme composante
```

C. Lecture (Remplissage) d'un tableau

On peut remplir un tableau T de n éléments par une suite d'instructions de lecture comme suit : Lire (T [1]) , Lire(T[2]), ... Lire (T[n])

On remarque que l'instruction Lire est répétée n fois, et afin d'éviter cette répétition, on va utiliser une des structures répétitives pour la lecture des éléments d'un tableau.

Algorithme	langage C

Var
T : Tableau [10] d'entiers ;
i : entier ;
Début
Pour i =1 **à** 10 **Faire**
Écrire ("T [", i, "] = ") ;
Lire (T[i]) ;
Fin pour
FIN
	#include<stdio.h>
	int main ()
	{
	int T[10],i;
	for (i=0;i<=9;i++)
	{
	printf("T[%d] = ",i);
	scanf("%d",&T[i]);
	}
	return o; }

D. Affichage d'un tableau

De la même façon, on peut afficher les m éléments d'un tableau T, comme suit :

Algorithme	langage C

Var
T : Tableau [10] d'entiers ;
i : entier ;
Début
Pour i =1 **à** 10 **Faire**
Écrire (T[i]) ;
Fin pour
FIN
	#include<stdio.h>
	int main ()
	{
	int T[10],i;
	for (i=0; i<=9; i++)
	{
	printf ("%d\n", T[i]); }
	return o; }

E. Initialisation d'un tableau

Maintenant que l'on sait parcourir un tableau, nous sommes capables d'initialiser toutes ses valeurs à 0 en faisant une boucle

Exemple :

```
void main ()
{
int tableau [4] , i = 0;     // Initialisation du tableau
for (i = 0 ; i < 4 ; i++)
{
tableau [i] = 0;
}
```

Il faut savoir qu'il existe une autre façon d'initialiser un tableau un peu plus automatisée en langage C. Elle consiste à placer les valeurs une à une entre accolades, séparées par des virgules.

Exemple :

```
void main ()
{
int tableau [4] = {0, 0, 0, 0} ;
}
```

On peut également définir les valeurs des premières cases du tableau, toutes celles que vous n'aurez pas renseignées seront automatiquement mises à 0.

Exemple :

```
void main ()
{
int tab1 [4] = {0, 0, 0, 0};          // 0, 0, 0, 0
int tab2 [6] = {10 , 23};             // 10, 23, 0, 0, 0, 0
int tab3 [4] = {0};                   // 0, 0, 0, 0
int tab4 [5] = {1};                   // 1, 0, 0, 0, 0, 0,
}
```

Voici la solution complète résolvant le problème posé dans l'exemple introductif :

Algorithme	langage C
Algorithme exemple_intro ; **Var** Notes : tableau [1..200] : réel ; Somme, moyenne : réel ; i , N : entier ;	# include <stdio .h> main () { int T [200] , i , N ; double Somme, moyenne ;

Début
Pour i = 1 à 200 faire
Ecrire (‘donner la note de l’élève N° ‘, i) ;
Lire (Notes[i]) ;
Fin pour
/* calcul de la moyenne */
Somme ← 0 ;
Pour i = 1 à 200 faire
Somme ← Somme + Notes[i] ;
Fin pour
moyenne ← Somme/200 ;
N ← 0 ;
Pour i = 1 à 200 faire
Si (Notes[i] > moyenne) **alors**
N ← N +1 ;
Fsi
Fin pour
Ecrire (‘moyenne de ces 200 notes = ‘ , moyenne);
Ecrire (N , ‘ élèves ont plus de cette moyenne ‘);
Fin
	for (i=0 ; i <200 ; i++)
	{
	printf ("donnez la note de l’élève N° %d : ", i+1) ;
	scanf ("%d", &t[i]) ;
	}
	Somme = 0 ;
	for (i=0 ; i <200 ; i++) Somme += T[i] ;
	moyenne = Somme/200 ;
	N = 0 ;
	for (i=0 ; i <200 ; i++)
	if (T[i] > moyenne) N ++ ;
	
	printf ("moyenne de ces 200 notes = %f\n", moyenne) ;
	
	printf ("%d élèves ont plus de cette moyenne ", N) ;
	}

III. Les tableaux multidimensionnels

Dans cette section on ne garde que les tableaux à deux dimensions (matrices) vu leur utilité pratique, notamment en mathématiques.

Exemple introductif :

Considérons un tableau NOTES_ ASD1 à une dimension pour mémoriser les notes de 200 étudiants du module ASD1:

Var

NOTES_ ASD1 : tableau [200] : réel ;

14	13	9.50	12	11	10.50	08	…..	…..	…..	…..	……	16

Pour mémoriser les notes des étudiants dans les 08 matières d'un trimestre, au lieu d’utiliser 08 tableaux à une dimension de taille 200 (ce qui est énorme), nous pouvons rassembler plusieurs de ces tableaux uni-dimensionnels dans un seul tableau NOTES à deux dimensions

Var

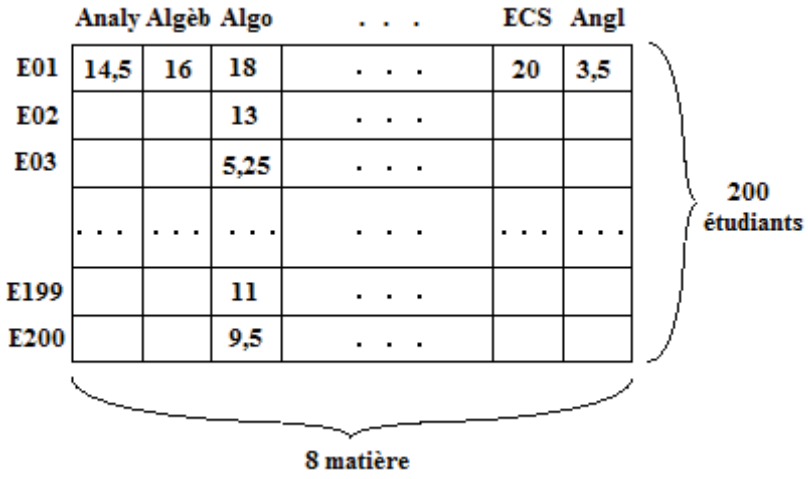
NOTES : tableau [1..8,1.. 200] : réel ;

- Dans une ligne nous retrouvons les notes de tous les étudiants dans une matière.
- Dans une colonne, nous retrouvons toutes les notes d'un étudiant.

Définitions 5.2 : Tableaux à deux dimensions

Un tableau à deux dimensions est à interpréter comme un tableau (uni-dimensionnel) de dimension N dont chaque composante est un tableau (uni-dimensionnel) de dimension M.

On appelle N le ***nombre de lignes*** du tableau et M le ***nombre de colonnes*** du tableau. N et M sont alors les deux ***dimensions*** du tableau. Un tableau à deux dimensions contient donc ***N*M composantes***.



On dit qu'un tableau à deux dimensions est ***carré***, si N est égal à M.

A. Déclaration d'un tableau à deux dimensions

Comme pour les tableaux à une dimension, une matrice est caractérisée par:

- Le nom
- La taille de la matrice (nombre de ligne x nombre de colonnes)
- Le type des éléments de la matrice.

Chaque élément dans une matrice est caractérisé par le numéro de la ligne et le numéro de la colonne

Syntaxe :

Algorithme	langage C
Var Nom_Tabl : tableau[1..<DimLigne>,1..<DimCol>]: type	Type Nom_Tabl [DimLigne] [DimCol] ;

Exemple:

Mat : tableau [8, 200] des entiers ;
int Mat [8] [200] ;

B. Accès aux éléments d'un tableau à deux dimensions

On accède à chaque élément du tableau par deux indices, un indice pour préciser le numéro de ligne et le second pour préciser le numéro de colonne.

Exemple : Considérons un tableau M MAT de dimensions 5 et 4.

MAT

5	6	3	22
0	0	1	8
4	2	8	-1
7	50	78	0
19	27	5	15

MAT [3,2] = 2 : l'élément de la 3ème ligne et la 2ème colonne

Remarque : le premier indice représente toujours la ligne et le second représente toujours la colonne

En langage C :

- *Les indices du tableau varient de 0 à N-1, respectivement de 0 à M-1.*
- *L'élément de la i ème ligne et j ème colonne est noté : A[i-1][j-1]*

C. Lecture d'un tableau à deux dimensions

On utilise deux boucles « Pour » imbriquées. L'indice i est utilisé pour parcourir les lignes et l'indice j pour parcourir les colonnes.

Algorithme	langage C

Var
MAT : Tableau [n, m] d'entiers ;
i, j: : entier ;
Début
Pour i =1 **à** n **Faire**
Pour j =1 **à** m **Faire**
lire (MAT[i, j]) ;
Fin pour
Fin pour
FIN
	#include<stdio.h>
	int main()
	{
	int MAT[3][2] ;
	int i, j;
	for (i = 0 ; i < 3; i++)
	{
	for (j = 0; j < 2; j++)
	{
	printf("MAT [%d,%d]= ",i,j);
	scanf("%d",&MAT[i][j]);
	}
	}
	return o;
	}

On peut remplir la matrice de deux manières différentes ou bien par lignes ou par colonnes. Pour lire colonne par colonne, on inverse juste i et j

D. Affichage d'un tableau à deux dimensions

De la même façon, on peut afficher les éléments de la matrice MAT comme suit :

Algorithme	langage C

Var
MAT : Tableau [n, m] d'entiers ;
i, j: : entier ;
Début
Pour i =1 **à** n **Faire**
Pour j =1 **à** m **Faire**
Ecrire (MAT [i, j]) ;
Fin pour
Fin pour
FIN
	#include<stdio.h>
	int main()
	{
	int MAT[3][2] ;
	int i, j;
	for (i = 0 ; i < 3; i++)
	{
	for (j = 0; j < 2; j++)
	{
	printf("tab[%d,%d]= %d \n", i, j, tab[i][j]);
	}
	}
	return o;
	}

E. Initialisation d'un tableau à deux dimensions

Lors de la déclaration d'un tableau, on peut initialiser les éléments du tableau, en indiquant la liste des valeurs respectives entre accolades. À l'intérieur de la liste, les éléments de chaque ligne du tableau sont encore une fois comprises entre accolades.

Pour améliorer la lisibilité des programmes, on peut indiquer les composantes dans plusieurs lignes.

Exemple :

```
int A[3][10] ={{ 0,10,20,30,40,50,60,70,80,90},
               {10,11,12,13,14,15,16,17,18,19},
               { 1,12,23,34,45,56,67,78,89,90}};
```

- Lors de l'initialisation, les valeurs sont affectées ligne par ligne en passant de gauche à droite.
- Nous ne devons pas nécessairement indiquer toutes les valeurs : Les valeurs manquantes seront initialisées par zéro.

Revenons à l'exemple introductif, le tableau notes contient les notes de 200 étudiants dans 8 matières, l'algorithme suivant permet de calculer la moyenne générale de chaque élève.

```
Algorithme moy_gen ;
Var
      notes : tableau[1..8,1..200] : réel ;
      i, j : entier ;
      Som , Moy : réel ;

 Début
     Pour j = 1 à 200 faire
         Ecrire ( ' donner les 08 notes de l'étudiant N° ', j )
         Pour i = 1 à  8  faire
            Ecrire (' note N° ', i ) ;
            Lire ( notes [i , j]) ;
         Fin pour
     Fin pour
     Pour j = 1 à 200 faire
         Som ← 0 ;
         Pour i = 1 à  8  faire
            Som ← Som +notes[i , j]) ;
         Fin pour
         Moy ← Som/10 ;
         Ecrire ('l'étudiant N° ', j ,' a pour moyenne ',Moy )
     Fin pour
 Fin .
```

IV. Les chaînes de caractères

Les chaînes de caractères sont en fait des tableaux de caractères. Leur manipulation est donc analogue à celle d'un tableau à une dimension, mais elles sont enrichies par d'autres fonctions spéciales facilitant leur manipulation.

A. Les caractères

Comme nous avons noté précédemment, ce type représente le domaine des caractères qui contient les lettres alphabétiques, les caractères numériques, les caractères spéciaux (., ?, !, <, >, =, *, +, ...etc), et le caractère espace. Si une variable est déclarée de type caractère, il va occuper un octet en mémoire.

Comme la mémoire ne peut stocker que des nombres, on a inventé une table qui fait la conversion entre les nombres et les lettres (la table ASCII En C,). Ainsi le nombre 65 par exemple équivaut à la lettre A. il suffit d'écrire cette lettre entre apostrophes, comme ceci : 'A' et 'A' sera donc remplacé par la valeur correspondante : 65.

Exemple :

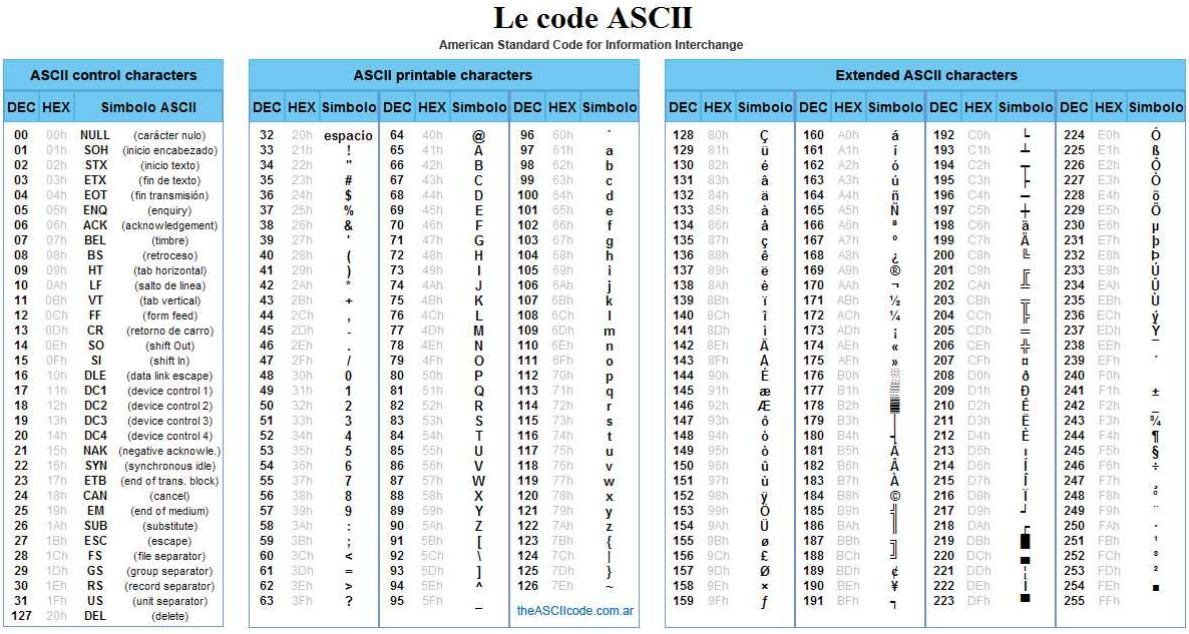
int main ()
{
char lettre = 'A';
printf ("%c\n", lettre);
return 0;
}
	int main ()
	{
	char lettre = 'A';
	printf ("%d\n", lettre);
	return 0;
	}

Affichage à l'écran : A	***Affichage à l'écran :*** 65

Qu'est-ce que le code ASCII ?

Le code ASCII (*American Standard Code for Information Interchange*) est un code standardisé qui permet d'unifier la représentation des caractères ainsi que la communication entre les ordinateurs.

Le code ASCII, ou table ASCII, est basé sur un principe assez simple où chaque caractère (chiffre, lettre, etc.) possède un code numérique pour pouvoir être stocké et interprété par un ordinateur.



Le code ASCII

American Standard Code for Information Interchange

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Figure 5.1 : La table ASCII

Dans sa première version, le code ASCII représente les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127). Ensuite, il a été étendu pour utiliser 8 bits (256 caractères, de 0 à 255) afin de permettre le codage des caractères nationaux (non seulement anglais tels que les caractères accentués comme : ù, à, è, é, â,...etc) et les caractères semi-graphiques.

B. Déclaration des chaînes de caractères.

Syntaxe :

Algorithme	langage C
Var Nom_variable : chaîne de caractère [dim]; *Ou bien* Nom_variable : chaîne de caractère ;	char Nom_variable [dim];

Exemple :

texte : chaîne de caractère [10];
char texte[10];

En langage C, le compilateur réserve (dim-1) places en mémoire pour la chaine de caractères. Une chaîne de caractère doit impérativement contenir un caractère spécial à la fin de la chaîne, appelé « *caractère de fin de chaîne* ». Ce caractère s'écrit '\0'.

- Le caractère '\0' permet tout simplement d'indiquer la fin de la chaîne.
- Par conséquent, pour stocker le mot " Salut " (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 char, mais de 6 .

Exemple :

char texte [6] = " Salut ";

'S'	'a'	'l'	'u'	't'	'\0'

- En tapant entre guillemets la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère '\0'.
- Il écrit ensuite une à une les lettres du mot _ Salut _ en mémoire et ajoute le '\0'.

Exemple :

```
int main ()
{
char chaine [] = " Salut ";
printf ("%s", chaine );
return 0;
}
```

Il y a toutefois un défaut : ça ne marche que pour l'initialisation ! Vous ne pouvez pas écrire plus loin dans le code : chaine = "Salut".

C. Lire une chaîne de caractères

On peut lire une chaîne de caractères entrée en utilisant la fonction ***scanf*** et le format %s. Une chaîne étant un pointeur, on n'écrit pas le symbole &.

Exemple :

scanf ("%s",texte);

Remarque: *scanf* ne permet pas la saisie d'une chaîne de caractères comportant des espaces. Les caractères saisis à partir de l'espace ne sont pas pris en compte.

On peut aussi lire une chaîne de caractères en utilisant la fonction ***gets*** non formatée (n'utilise pas le format %s) .

Exemple :

gets (texte);

D. Afficher une chaîne de caractères

On peut utiliser la fonction ***printf*** et le format %s, comme on peut aussi utiliser la fonction ***puts*** non formatée.

Exemple :

puts (texte); *est équivalent à* **printf**("%s", texte);

E. Fonctions sur les chaines de caractères

La bibliothèque <***string.h***> fournit une multitude de fonctions pratiques pour le traitement et la manipulation des chaînes de caractères. Voici quelques fonctions les plus fréquemment utilisées:

- **strlen : calculer la longueur d'une chaîne**

strlen(s) est une fonction qui calcule la longueur de la chaîne de caractère « ***s*** » sans compter le caractère '\0' final.

Exemple :

```
int main ()
{
char chaine [] = " Salut ";
int longChaine = 0;
longChaine = strlen ( chaine );
printf ("La chaine %s fait %d caractères de long ", chaine , longChaine );
return o;
}
```

- **strcpy : copier une chaîne dans une autre**

La fonction ***strcpy(s,t)*** permet de copier une chaîne « ***t*** » à l'intérieur de la chaîne « ***s*** ».

Exemple :

```
int main ()
{
char chaine1 [] = " Texte ", chaine2 [50];
strcpy (chaine2 , chaine1 ); // On copie " chaine1 " dans " chaine2 "
printf ("chaine1 vaut : %s\n", chaine1 );
printf ("chaine2 vaut : %s\n", chaine2 );
return 0;
}
```

La fonction ***strncpy(s,t,n)*** copie au plus « n » caractères d'une chaîne « ***t*** » vers une autre chaîne « ***s*** ».

- **strcat : concaténer 2 chaînes**

La fonction ***strcat(s,t)*** ajoute une chaîne « ***t*** » à la fin d'une autre chaîne « ***s*** ». On appelle cela la concaténation.

Exemple :

```
int main ()
{
char chaine1 [100] = " etudiants ", chaine2 [] = " MI ";
strcat ( chaine1 , chaine2 );
printf (" chaine1 vaut : %s\n", chaine1 );
printf (" chaine2 vaut toujours : %s\n", chaine2 );
return o;
}
```

La fonction ***strcat(s,t,n)*** ajoute au plus « n » caractères d'une chaîne « ***t*** » à la fin d'une autre chaîne « ***s*** ».

- **strcmp : comparer 2 chaînes**

La fonction ***strcmp(s,t)*** compare la chaîne « ***s*** » et la chaîne « ***t*** » lexicographiquement et fournit un résultat :

zéro si « s » et « t » sont identiques.
négatif si « ***s*** » précède « ***t*** ».
positif si « ***s*** » suit « ***t*** ».

Exemple :

```
int main () {
char chaine1 [] = " Texte de test ", chaine2 [] = " Texte de test ";
if ( strcmp ( chaine1 , chaine2 ) == 0)
      printf ("Les chaines sont identiques \n");
else
   printf ("Les chaines sont differentes \n");
return o;    }
```

Remarques :

- Comme le nom d'une chaine de caractères représente une adresse fixe en mémoire, on ne peut pas 'affecter' une autre chaine au nom d'un tableau :

  Il faut bien copier la chaine, caractère par caractère, ou utiliser la fonction **strcpy** : **strcpy(A, "Hello");**

- La concaténation de chaines de caractères en C ne se fait pas par le symbole '+' comme en langage algorithmique. Il faut ou bien copier la deuxième chaine caractère par caractère ou bien utiliser la fonction **strcat**.

- La fonction **strcmp** est dépendante du code de caractères et peut fournir différents résultats sur différentes machines.

V. Conclusion

Nous avons vu à travers ce chapitre comment utiliser des tableaux et des chaînes de caractères. Dans la pratique, les types de tableaux les plus utilisés sont les vecteurs (tableaux à une seule dimension) et les matrices (tableaux à deux dimensions).

Série d'exercices 5

Exercice 5.1 : **Manipulation d'un tableau**

Ecrire un algorithme\programme permettant de remplir un tableau de 10 éléments entiers et les affiche par la suite dans le sens inverse (du dernier élément au premier élément).

Exercice 5.2 : **Addition**

Ecrire un algorithme\programme qui fait l'addition de deux tableaux entier de 10 éléments en stockant le résultat dans un troisième tableau.

Exercice 5.3 : **Minimum / Maximum**

Ecrire un algorithme\programme permettant de trouver le MAX et le MIN dans un tableau de 20 éléments réels.

Exercice 5.4 : **Nombre d'occurrences**

Ecrire un algorithme\programme permettant de calculer le nombre d'occurrences d'un élément donné dans un tableau entier de 20 éléments

Exercice 5.5 : **Compter**

Ecrire un algorithme\programme qui comptabilise le nombre de caractères, de mots et de phrases dans un texte. Les mots sont séparés par des espaces et les phrases séparées par un point. (Sans compter les séparateurs : espace et point)

Exercice 5.6 : **Anagrammes**

Ecrire un algorithme\programme qui lit deux mots et qui détermine s'ils sont Anagrammes. Sachant qu'un mot est dit anagramme d'un autre mot s'ils utilisent (sont formés par) les mêmes lettres.

Exemples :

CHIEN anagramme de CHINE, NICHE.
AIMER anagramme de MAIRE, MARIE.

Chapitre 6

Les types personnalisés

Sommaire

Objectif

Parfois, aucun des types disponibles en langage de programmation ne permet de représenter nos données complexes ou composées. Dans ce cas-là, il faut définir de nouveaux types de données appelés types personnalisés. L'objectif de ce chapitre est de voir comment définir et utiliser ces types.

I. Introduction

Jusqu'à présent, nous n'avons vu que quelques types de base (entier, réel, caractère et logique). Ces types ont été utilisés pour décrire des structures composées de plusieurs éléments de même type, telles que les tableaux et les chaînes de caractères. Néanmoins, ils nous ne permettent pas de regrouper des informations de différent type liées au même objet dans une seule structure, par exemple: comment décrire un nombre complexe ? Ou bien les informations sur un étudiant ?

Par conséquent, dans le présent chapitre, nous allons approfondir nos connaissances dans deux directions, à savoir:

- La possibilité de définir de nouveaux types par l'utilisateur de façon plus agréable pour décrire ses objets.
- Présenter les instructions complémentaires et nécessaires pour faciliter le traitement de ces nouveaux types.

Déclaration des types

Afin de pouvoir utiliser un nouveau type dans un algorithme, il faut d'abord le déclarer dans la partie de déclaration. Cette partie commence par le mot réservé "**Type**" et comprend la déclaration de tous les types définis par l'utilisateur qui seront utilisés dans l'algorithme.

II. Enumérations

Les énumérations, ou bien les types énumérés, constituent des types de données qui peuvent être définis par l'utilisateur/programmeur. Un type énuméré est un type permettant de représenter des objets pouvant prendre leur valeur dans une liste finie et ordonnée. En d'autres mots, une énumération est une liste de valeurs définie par l'utilisateur.

La définition d'un type énuméré consiste à déclarer une liste de valeurs possibles associées à un type.

A. Déclaration d'un type énuméré

La déclaration des types énumérés se fait avec une syntaxe spéciale.

Syntaxe :

Algorithme	langage C
TYPE Nom_Enumeration = (valeur1, valeur 2,..., valeur n) ;	**enum** Nom_Enumeration { val1, val-2,..., val-n };

Une fois un type énuméré déclaré, les variables de ce type énuméré peuvent être déclarés.

Exemple :

TYPE
SEMAINE= (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche) ;
ANNEE = (Jan, Fev, Mar, Avr, mai, Juin, Jui, Aout, Sept, Oct, Nov, Dec) ;
VAR
jour1, jour2 : SEMAINE ;
mois : ANNEE ;

En langage C / Algorithme, les objets de type ***enum*** sont représentés comme des ***int***. Chaque valeur de la liste (type énuméré) possède un rang associé de 0 à n-1.

Exemple :

```
#include <stdio.h>
enum week { sunday, monday, tuesday, wednesday, thursday, friday, saturday };
int main()
{
enum week today;
today = wednesday;
printf("Day %d",today+1);
return 0;
}
```

B. Manipulation d'un type énuméré

Les variables d'un type énuméré ne pourront pas avoir d'autres valeurs que celles que l'on a listées. On peut les manipuler dans un algorithme grâce à :

- **l'Affectation :**

  jour1 ← Vendredi ;
  jour2 ← jour1 ;

- **Fonctions prédéfinies :**

  En principe, il existe trois fonctions qui sont :

  - pred(élément), // **retourne la valeur précédente**
  - succ(élément), // **retourne la valeur suivante**
  - ord(élément) // **retourne l'ordre d'un élément (dans la déclaration), entre 0 et N-1).**

Exemple :

pred(mardi) = lundi ;
succ(mardi) = mercredi ;
ord(mardi) = 1 ;

III. Enregistrements (Structures)

Même si la structure de tableau nous permet de décrire une structure formée de plusieurs éléments de même type, il nous ne permet pas de regrouper des informations liées au même élément et qui n'ont pas forcément le même type. Or il existe d'autres données qui sont composées d'éléments de types différents comme par exemple:

- Les dates (année, mois, jour)
- Les nombres complexes (partie réelle, partie imaginaire)
- Les fiches estudiantines ou personnelles(nom, prénom, date_naissance, adresse)

La nature différente de ces éléments a poussé les programmeurs à utiliser des structures appelées "Les enregistrements" qui sont mieux adaptées pour représenter ce type de données.

Définition 6.1 : Enregistrement

Un enregistrement (ou structure) est une structure de donnée formée d'un ou de plusieurs éléments (appelés champs) pouvant être de types différents. Á la différence du tableau, ces éléments ne sont pas ne sont pas indexés.

A. Déclaration d'un enregistrement

La définition du type enregistrement précise pour chaque élément un identificateur de champ, dont la portée est limitée à l'enregistrement, et le type de ce champ.

La déclaration des enregistrements se fait dans la partie de déclaration des types comme suit :

Syntaxe :

Algorithme	langage C

nom_de_type= enregistrement:
champ1 : type_champ_1 ;
champ2 : type_champ_2 ;
. . . . . .
champN : type_champ_N;
fin enregistrement
	struct Nom_Structure
	{
	type_champ_1 champ1 ;
	type_champ_2 champ2 ;
	.....
	type_champ_N champN ;
	};

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

Exemple :

Algorithme	langage C

TYPE
Etudiant=**enregistrement**
nom : chaîne de caractères [30] ;
prenom : chaîne de caractères [25] ;
age : entier ;
moyenne : réel ;
fin enregistrement.
 	
VAR
e1 , e2 : Etudiant ;
	struct Etudiant {
	char nom [30];
	char prenom [25];
	int age;
	float moyenne ;
	};
	void main (){
	struct Etudiant e1 , e2;
	}

Question : En langage C, Faut-il obligatoirement écrire le mot-clé **struct** lors de la déclaration des variables de type enregistrement ?

Réponse : L'instruction appelée ***typedef*** permet de créer de nouveaux noms de types (autrement dit, elle sert à créer un alias pour l'enregistrement).

Exemple :

```
typedef float reel ;
typedef struct Etudiant Etudiant ;
struct Etudiant {
    char nom [30];
    char prenom [25];
    int age;
    reel moyenne ;
};
void main (){
     reel r;
     Etudiant e1 , e2;
}
```

- ✓ ***typedef*** : permet de créer un alias pour l'enregistrement;
- ✓ ***struct Etudiant :*** c'est le nom de l'enregistrement pour lequel on veut créer un alias (c'est-à-dire un « équivalent ») ;
- ✓ ***Etudiant :*** c'est le nom de l'équivalent.

B. Manipulation d'un enregistrement

Un enregistrement doit être manipulé champ par champ. On accède à ses champs en indiquant le nom de l'enregistrement suivi d'un point et du nom du champ.

Prenons l'exemple précédent de l'étudiant, les champs d'un enregistrement peuvent être manipulés par l'affectation, Ou bien par lecture et écriture :

Exemple affectation :

```
e1.nom ← 'khelifa ' ;
e1.prenom ← 'Rayen' ;
e1.age ← 17 ;
e1.moyenne ← 18.75 ;
e2 ← e1 ;
```

Exemple lecture et écriture :

Algorithme	langage C
Lire (e1. nom) Lire (e1. prenom) Ecrire (e1.nom) Lire (e1. age) Lire (e1. moyenne)	void main () { Etudiant e1 , e2; puts (" Donnez votre nom : "); scanf ("%s", &e1.nom); puts (" Donnez votre prenom : "); scanf ("%s", &e1. prenom); puts (" Donnez votre age : "); scanf ("%d", &e1.age); puts (" Donnez votre moyenne : "); scanf ("%f", &e1. moyenne); e2=e1; }

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

Initialiser une structure

L'initialisation d'un enregistrement ressemble un peu à celle d'un tableau. En effet, on peut initialiser un enregistrement au moment de sa déclaration :

Exemple :

```
void main (){
Etudiant e = {" khelifa ", " Rayen ", 17, 18.75} ;
}
```

C. Les structures imbriquées

Un champ dans un enregistrement peut lui-même être un enregistrement.

Exemple :

```
typedef struct Date Date ;
struct Date {
  int jour ;
  int mois ;
  int annee ;
};
typedef struct Etudiant Etudiant ;
struct Etudiant {
  char nom [30];
  int age;
  float moyenne ;
  Date date_naissance ;
};
```

IV. Conclusion

Nous avons vu à travers ce chapitre comment utiliser les types personnalisés. Le type énuméré ou bien les enregistrements, sont des types de variables personnalisés qu'on peut créer et utiliser dans des programmes. C'est au programmeur de les définir, contrairement aux types de base. Ces types personnalisés permettent de combler l'insuffisance des types de données de base offerts par les langages de programmation et par conséquent représenter des données complexes et composées pour un développement avancé.

Série d'exercices 6

Exercice 6.1 : Contacts

Ecrire un programme qui permet de remplir et d'afficher un tableau de dimension N (fixée par l'utilisateur) qui contient les contacts de vos amies (leurs noms avec leurs numéros de téléphone).

Exercice 6.2 : Panneaux de bois

Une menuiserie industrielle gère un stock de panneaux de bois. Chaque panneau possède une largeur, une longueur et une épaisseur en millimètres, ainsi que le type de bois qui peut être pin (code 0), chêne (code 1) ou hêtre (code 2).

1) Définir une structure Panneau contenant toutes les informations relatives à un panneau de bois.

2) Écrire des fonctions de saisie et d'affichage d'un panneau de bois.

3) Écrire une fonction qui calcule le volume en mètres cube d'un panneau.

Exercice 6.3 : Composants électroniques

Un grossiste en composants électroniques vend quatre types de produits :

- Des cartes mères (code 1) ;
- Des processeurs (code 2) ;
- Des barettes mémoire (code 3) ;
- Des cartes graphiques (code 4).

Chaque produit possède une référence (qui est un nombre entier), un prix en euros et des quantités disponibles.

1) Définir une structure Produit qui code un produit.

2) Écrire une fonction de saisie et d'affichage des données d'un produit.

3) Écrire une fonction qui permet à un utilisateur de saisir une commande d'un produit. L'utilisateur saisit les quantités commandées et les données du produit. L'ordinateur affiche toutes les données de la commande, y compris le prix.

Références Bibliographiques

- Thomas H. Cormen, Algorithmes Notions de base *Collection : Sciences Sup, Dunod,* 2013.

- Rémy Malgouyres, Rita Zrour et Fabien Feschet. Initiation à l'algorithmique et à la programmation en C : cours avec 129 exercices corrigés. 2ième Edition. Dunod, Paris, 2011. ISBN : 978-2-10-055703-5.

- Jacques TISSEAU. Initiation à l'algorithmique. Cours d'Informatique S1. Presse Univ. Ecole Nationale d'Ingénieurs de Brest, 2009.

- Maurizio Gabbrielli et Simone Martini, Programming Languages: Principles and Paradigms , Springer, 2010. ISBN 9781848829138.

- Mahmoud BRAHIMI. Algorithmique et Structures de données I : Cours et exercices corrigés en langage C . Support de cours S1. Université M'sila,2018

- L, Baba Ahmed et S, Hocine. Algorithmique et structure de données statistiques. OPU, 2016. ISBN 978.9961.0.0630.6

- Ammar Boucherit. Algorithmique et Structures de données I. Support de cours S1. Université d'El Oued, 2020.