République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie d'Oran Mohamed BOUDIAF
USTO-MB

Algorithmique et Structures de Données I (ASD1)

Dr. Hafida Bouziane
Faculté des Mathématiques et Informatique
Département d'Informatique
e-mail: hafida.bouziane@univ-usto.dz

"L'informatique est une science qui se nourrit de la logique, de l'algorithmique et de la programmation." - Philippe Dreyfus.

AVANT PROPOS

CEt ouvrage est une modeste contribution à l'enseignement de l'algorithmique, bagage de base pour l'art de la programmation. Comme on apprend par l'exemple et la pratique, ce document fournit des directives méthodologiques et des conseils résultant de plusieurs années d'enseignement de la matière, à l'intention des novices. Il est consacré plus particulièrement à l'initiation à l'algorithmique et à la programmation.

Ce document s'adresse aux étudiants du premier cycle universitaire suivant un enseignement d'informatique, plus particulièrement la première année Licence et Ingénieur toutes spécialités confondues. Cette première partie de la matière Algorithmique et Structures de Données comporte quelques généralités sur le fonctionnement de l'ordinateur et les langages de programmation. La notion d'algorithme est introduite avec détails avec une présentation des notions essentielles de programmation dans le but de faciliter la compréhension des concepts fondamentaux nécessaires à la mise en œuvre en C des algorithmes étudiés. La syntaxe des algorithmes ressemble à celle du langage de programmation PASCAL qui par son caractère simple et structuré reste un langage bien adapté aux débutants. Les étudiants pourront manipuler les types de données de base ainsi que les tableaux. Cette partie introduit également la notion de types personnalisés dont les structures, les énumérations et les unions en langage C.

Les programmes sont écrits en langage C qui est un excellent point de départ pour un programmeur. Les exemples et les exercices proposés sont illustrés par des figures présentant les résultats de leur mise en œuvre sur machine.

Organisation du document :

Le polycopié est réparti en sept chapitres dont les deux premiers présentent quelques notions préliminaires sur l'informatique et l'algorithmique, utiles pour maîtriser le contenu du document.

Le premier chapitre présente une introduction générale sur l'informatique et un bref historique sur l'informatique et l'algorithmique.

Le second chapitre présente quelques notions préliminaires relatives au traitement automatique de

l'information et aux composants d'un ordinateur dans le but de familiariser l'étudiant avec les concepts, le matériel et les ressources informatiques.

Le troisième chapitre est consacré à l'écriture des algorithmes à l'aide des différents langages de description d'algorithmes et des programmes en langage de rogrammation C.

Le quatrième chapitre est consacré aux structures conditionnelles simples, composées et le choix multiple.

Le cinquième chapitre introduit la notion de boucle et présente les différentes structures de boucles et les branchements.

Le sixième chapitre est consacré à la structure de données tableau et le type chaine de caractères.

Le septième chapitre et dernier introduit la notion de type personnalisé et met l'accent sur le type de données composées ou structures.

Conventions syntaxiques :

Pour présenter les notions essentielles et les directives méthodologiques, certaines règles ont été adoptées, que nous résumerons dans ce qui suit :

- Les remarques importantes du cours sont mises en relief en utilisant des icônes indiquant leur nature.
- Les codes présentés dans cet ouvrage ont été implémentés à l'aide de l'environnement de déve–loppement Dev-C++.
- Ce qui est entre crochets « [ ] » est facultatif.
- Le symbole « | » veut dire ou bien.
- La présence de points de suspensions dans le code source, indique n'importe qu'elles autres instructions.
- Les constantes sont différenciées des variables par un mix entre majuscule et miniscule dans leurs identificateurs.

Remerciements :

Je remercie tous ceux qui ont contribué de près ou de loin à l'élaboration de ce document et mes collègues Mme Raaf-Tair Hafida et Mme Khensous Ghania qui ont accepté d'examiner le contenu et dont les avis ont été constructifs et encourageants pour une meilleure présentation de cette modeste contribution.

SOMMAIRE

CHAPITRE 1

INTRODUCTION ET HISTORIQUE

Sommaire

Figures

1.1 Introduction

Ce chapitre introductif présente un bref historique de l'informatique et des algorithmes. Il relate les différentes étapes d'évolution de cette discipline qui a changé le monde.

1.2 Bref historique de l'informatique

Le terme informatique vient de l'assemblage du mot **information** et le mot **automatique**, il veut dire traitement automatique de l' information, équivalent en anglais à "computer science" ou de "data-processing". C'est en Allemagne qu'est naît le mot informatik en 1957, avant d'être repris en France où il a été utilisé pour la première fois en **1962** par **Philippe Dreyfus** (1945-2018), ancien directeur du Centre national de calcul électronique de Bull pour son entreprise Société d'Informatique Appliquée (SIA). Depuis ses débuts, l'informatique a subit beaucoup de transformations du point de vue principes de base, langages de programmation et matériel. Ainsi, l'informatique a évolué à travers plusieurs générations distinctes, chacune marquée par des innovations technologiques majeures. On distingue dans la littérature cinq générations qui peuvent être complétées par une sixième génération si on les décrit comme suit :

Première génération (1945-1955) : des relais aux tubes à vide (lampes électroniques) Les premiers ordinateurs étaient à base de tubes à vide. Ces machines étaient énormes (ENIAC, UNIVAC I) avec une très forte consommation d'énergie.

Deuxième génération (1955-1965) : les transistors Le transistor a remplacé le tube à vide dans des ordinateurs plus petits, plus rapides et consommant moins d'énergie. Cette génération est marquée aussi par l'apparition des premiers langages de programmation de haut niveau (COBOL, FORTRAN).

Troisième génération (1966-1973) : les circuits intégrés L'invention du circuit intégré (puce électronique) a permis de miniaturiser davantage les composants des ordinateurs qui sont devenus plus puissants, plus accessibles et moins coûteux. Cette génération est marquée également par le développement des systèmes d'exploitation, la multiprogrammation et l'apparition des premiers chatbots (chatbots à règles).

Quatrième génération (1974-1979) : les micro-processeurs L'invention du microprocesseur en 1971 (Intel 4004) révolutionne l'informatique. Naissance de l'ordinateur personnel dans les années 1970-80 (Apple II, IBM PC).

Cinquième génération (1980-2014) : interfaces graphiques et réseaux Cette période voit l'explosion d'Internet, les logiciels grand public. Cette génération est aussi marquée par l'émergence de l'intelligence artificielle (IA), la cyber-sécurité, la réalité virtuelle,l'informatique quantique, les smartphones, le big data et le cloud computing. l'Internet des objets (IoT) a marqué sa présence et les systèmes deviennent de plus en plus intelligents et interconnectés. Cette Génération est aussi considèrée comme l'ère de l'IA générative" (depuis 2022-2023 environ). Les chatbots et les LLMs (Large Language Models - Grands Modèles de Langage) font partie intégrante de la cinquième génération actuelle, représentant une avancée majeure dans le domaine de l'IA. La première vague

des chatbots (années 1960-2000) comme ELIZA (1966) fonctionnaient avec des règles prédéfinies et de la reconnaissance de mots-clés étaient très limités. La deuxième vague (années 2000-2010) basée sur l'apprentissage automatique (machine learning) est devenue alors plus sophistiqués et capable d'apprendre de conversations et de s'améliorer progressivement. La troisième vague (à partir de 2017) à l'ère des LLMs, l'apparition de l'architecture Transformer marqua un tournant décisif. Les LLMs comme GPT (en 2018), BERT, Claude, et d'autres transforment radicalement les capacités conversationnelles. La révolution LLMs représente une rupture technologique car les chatbots peuvent comprendre et générer du langage naturel de manière sophistiquée, raisonner sur des problèmes complexes, écrire du code, analyser des données et accomplir une multitude de tâches cognitives. Enfin, l'interaction homme-machine se fait naturellement en langage courant.

Nous sommes actuellement dans une phase de transition dont le future sera probablement marqué par l'informatique quantique mature, les interfaces cerveau-machine et l'informatique neuromorphique.

Quelques dates et noms qui ont marqué l'histoire de l'informatique :

Les premières traces de l'informatique remontent au boulier chinois, illustré dans la Figure 1.1. Il a été utilisé comme instrument de calcul, quelques dates ont été inscrites dans l'histoire de l'informatique comme :

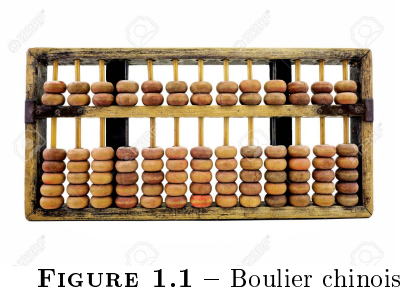


Figure 1.1 – Boulier chinois.

- **1614** : le mathématicien écossais **John Napier** (1550-1617) présente sa théorie des logarithmes, table de **Neper** qui transforme des multiplications compliquées en de simples additions et les divisions en soustractions.
- **1620** : invention de la règle à calcul à base de tables de Neper.
- **1624** : **Wilhem Schickard** (1592-1635) imagine une machine à calculer mécanique constituée des roues chiffrées pouvant effectuer les quatre opérations fondamentales.
- **1642** : le mathématicien et philosophe français **Blaise Pascal** (1623-1662) construit une machine mécanique capable d'additionner et de soustraire des nombres de huit chiffres, la Pascaline (première version) dédiée au calcul d'impôts. Cette machine fût créée pour son père qui était percepteur d'impôts. Plutart, son nom fût attribué à un langage de programmation (le Pascal).
- **1671** : le philosophe allemand **Gottfried Wilhelm von Leibniz** (1646-1716) améliore la Pascaline en la rendant capable également de multiplier et de diviser qui fût baptisée la Replica. Aussi, **Newton** et **Leibniz** introduisent le calcul différentiel et intégral ainsi que le calcul binaire qui sera par la suite adopté par la plupart des ordinateurs contemporains.
- **1725** : les français **Basile Bouchon** et **Jean-Baptiste Falcon** réalisent une machine à tisser à base de planchettes de bois munies de trous.
- **1805** : le français Joseph-Marie Jacquard (1752-1834) remplace les planchettes de bois munies de trous de la machine de Bouchon et Falcon par des cartes perforées et réalise ainsi, un métier à tisser où les mouvements des aiguilles sont commandés par des cartes perforées, ce fût le début de l'ère de la programmation.

- **1832** : invention du langage Morse, pour coder les caractères de l'alphabet, code binaire à base de deux symboles (trait court et trait long).
- **1833** : le mathématicien britanique **Charles Babbage** (1791-1871) imagine deux machines à calculer dont la seconde analytique combinant les possibilités des cartes perforées et la calculatrice avec la collaboration de la mathématicienne **Augusta Ada King** (1815-1852), comtesse de Lovelace fille du célèbre Lord Byron, un des plus grands poètes anglais à l'époque, dite **Ada Lovelace** dont le nom à été donné à un des premiers langages de programmation orientés objet en 1980. Son invention n'a pas pu être concrétisée faute de moyens à l'époque.
- **1847** : le mathématicien et logicien anglais **Georges Boole** (1815-1864) publie son ouvrage sur la logique binaire « Mathematical Analysis of Logic », où il définit les opérateurs logiques (booléens) fondés sur deux valeurs 0/1 pour coder Vrai/Faux.
- **1876** : **Alexander Graham Bell** invente le téléphone.
- **1890** : recensement des Etats Unis, l'ingénieur américain **Hermann Hollerith** (1860-1929) accompli la mission en deux ans grâce à son invention, une machine à compter automatique utilisant les cartes perforées.
- **1896** : **Herman Hollerith** fonde la société Tabulation Machine Corporation qui deviendra plutart en 1924, International Business Machine (IBM), l'informatique venait alors de naître.
- **1904** : invention des lampes à vide, diodes puis triodes, fonctionnant comme interrupteurs ou amplificateurs.
- **1936** : **Alan Mathison Turing** (1912-1954) propose sa définition de la « machine de Turing », la première machine universelle et **Alonzo Church** (1903-1995) invente le « lambda-calcul », qui se révèlent avoir des capacités de calcul équivalentes.
- **1937** : conception de l'ordinateur ABC (Atanasoff Berry Computer) par **John Vincent Atanasoff** (1903-1995) et son étudiant **Clifford Berry** (1918-1963), utilisant du binaire 16 bits pour l'additionneur, de l'électronique avec des diodes et une mémoire à base de lampes et tambours.
- **1941** : **Konrad Zuse** (1910-1995) fait fonctionner le premier ordinateur du monde utilisant le mode binaire au lieu du décimal, le Z3 (ou Zuse 3) qui étais précédé par Z1 (1938) puis Z2 (1939), il est l'un des pères de l'informatique en ayant développé le premier calculateur électromécanique Z1 en 1938. Le Z3 était plus fiable que les précédentes versions et servait à produire des calculs pour une usine aéronautique allemande, il fut détruit en 1944 par les bombardements alliés.
- **1943** : conception de l'énorme calculateur ASCC Mark I (Automatic Sequence-Controlled Calculator Mark I) à Harvard par **Howard Aiken** et son équipe, avec la collaboration d'IBM dont le principe de fonctionnement était proche de celui de la machine analytique de **Charles Babbage**, il pesait 5 tonnes et mesurait 17 m de long et 2,5 m de haut.
- **1945** : **John Von Neumann** (1903-1957) présente son rapport (First Draft of a Report on the EDVAC1) sur l'architecture interne d'un ordinateur, « architecture de Von Neumann » utilisée dans les ordinateurs modernes.
- **1946** : conception du premier ordinateur électronique l'ENIAC (Electronic Numerator Integrator Analyzer and Computer) par **John W. Mauchly** (19071980) et **John Presper Eckert**

(1919-1995), il pesait 30 tonnes et comptait 18 000 lampes. Il était basé sur le système décimal et uniquement programmable manuellement avec des commutateurs et des câbles à enficher, il a été utilisé pour mettre au point la bombe H.

- **1947** : invention du transistor qui révolutionne l'informatique permettant de concevoir des ordinateurs de plus grande capacité de calcul occupant un volume réduit.
- **1948** : **Claude Shannon** publie sa théorie mathématique de l'information introduisant la notion de quantité d'information d'un objet et sa mesure en bits.
- **1949** : construction de l'EDVAC (Electronic Discrete Variable Automatic Computer), l'un des premiers ordinateurs électroniques conçu suivant l'architecture de **Von Neumann** stockant ses données sur disques magnétiques et opérant en mode binaire plutôt que décimal comme l'ENIAC.
- **1951** : construction du premier ordinateur non expérimental UNIVAC de la société Speery-Rand.
- **1954** : apparition du langage FORTRAN (FORmula TRANslator) utilisé pour le calcul scientifique. IBM (société fondée en 1911) lance sur le marché le modèle 650, premier calculateur muni d'une mémoire à tambour dédié au calcul scientifique, il occupait plusieurs mètres cubes et était doté d'une mémoire vive de 2 Koctets.
- 956 : le terme d'Intelligence Artificielle est inventé lors d'une conférence à Dartmouth, aux Etats-Unis.
- **1958** : le premier ordinateur intégrant des transistors (circuit integré dit puce) mis au point par par Texas Instruments, l'IBM 7044 (64 Koctets de mémoire), **John McCarthy** invente le LISP (List Processing), premier langage adapté à l'Intelligence Artificielle.
- **1959** : conception de COBOL (Common Business Oriented Language), langage de programmation spécialisé pour la gestion et le domaine bancaire.
- **1960** : conception de ALGOL (ALGOrithmic Language), langage évolué dédié au calcul scientifique.
- **1964** : création du code ASCII, normalisé plus tard par ISO pour simplifier l'échange de données entre systèmes.
- **1967** : IBM lance la disquette (dans sa version 8 pouces) pour stocker les microprogrammes des systèmes 370.
- **1969** : Début du réseau Arpanet à 4 noeuds, renommé plus tard Internet réseau des réseaux qui s'appuie sur les protocoles TCP (Transmission Control Protocol) et IP (Internet Protocol), qui définissent les règles d'échange des données ainsi que la manière de récupérer les erreurs.
- **1971** : apparition de Unix, premier système d'exploitation multi-utilisateurs et réalisation du premier micro-processeur (milliers de transitors sur un seul circuit électronique), le 4004 d'INTEL dont les performances étaient équivalentes à celle de l'ENIAC (1946), qui occupait toute une pièce. Cette étape ouvrira la voie aux ordinateurs de la troisième génération et il étendra l'accès aux technologies numériques à de nouveaux usagers en permettant la naissance des ordinateurs personnels et des consoles de jeux.
- **1972** : conception du langage C, particulièrement adapté à la programmation et à l'utilisation de systèmes d'exploitation.

- **1973** : création de prototypes des premiers micro-ordinateurs et ommercialisation du Micral, le premier micro-ordinateur commandé par des interrupteurs n'ayant ni clavier ni écran. Invention du disque souple (floppy disc) et invention à Marseille du langage PROLOG (PROgrammation LOGique), par **Alain Colmerauer** (1941-2017).

- **1975** : la société MITS (Massachusetts Institute of Technology) lance l'Altair 8800 le premier véritable micro-ordinateur conçu à à Albuquerque en Californie basé sur le microprocesseur Intel 8080A, l'ancêtre de tous les systèmes actuels. **Bill Gates** (1955-) à 20 ans fonde l'entreprise Microsoft avec **Paul Allen** (1953-2018) et inventent un langage de programmation pour l'Altair 8800. Apparition du système d'exploitation CP/M.

- 1976 : **Steve Jobs** (1955-2011), **Steve Wozniak** (1950-) et Ronald Wayne (1934-) créent Apple Coputer et commercialisent le micro-ordinateur Apple I à châssis en bois, l'ingénieur américain **Seymour Cray** (1925-1996) invente le Cray 1, premier super-calculateur et conception du langage Smalltalk, qui introduit la programmation « orientée objet .

- 1977 : pour son deuxième essai, Apple sort le premier ordinateur personnel (PC) Apple II vendu à grande échelle qui incitera **Steve Jobs** à inventer le message man, one micro, inspiré du slogan démocratique man, one vote. Le jeu vidéo décolle aussi, grâce à la console à cartouches et l'Atari 2600 se vendra à plus de 40 millions d'exemplaires avec les jeux « E.T., « PacMan , « Space Invaders ,$\cdots$ etc.

- 1979 : **Alan Ashton**, professeur d'informatique dans l'Utah, invente le premier logiciel de traitement de texte « WordPerfect qui sera supplanté par « Microsoft Word en 1993.

- 1980 : réseau français Minitel disponible sur tout le territoire en 1982 et aux Etats-Unis, lancement de CNN, première chaîne télévisée d'informations en continu.

- 1981 : Le géant IBM (Intelligent Business Machines) lance le premier PC, équipé du logiciel MS-DOS (Microsoft Disk Operating System)qui est le système qui traduit le programme dans le langage de l'ordinateur lequel évoluera plutard en MS-Windows en intégrant une interface graphique. et l'Américain **Adam Osborne** présente le premier ordinateur portable (11 kilos), à ce moment là des groupes de hackers apparaissent aux Etats-Unis et en Allemagne.

- 1982 : L'arrivée du premier laptop à écran « refermable Grid Compass 1101, utilisé en 1985 à bord de la navette Discovery. Le magazine «Time» fait de l'ordinateur son « Homme de l'année , tandis que Philips et Sony lancent le CD (Compact Disk), premier support numérique pour la musique.

- 1983 : les micro-ordinateurs 16 bits inondent le marché et IBM impose la norme « compatible IBM mais Apple a maintenu son propre système d'exploitation MacOS. Conception du langage ADA (en hommage à Lady Ada Lovelace), extension du langage PASCAL pour répondre à une demande de la NASA. Utilisé depuis 1972 pour qualifier la jonction d'Arpanet (reliant des universités) et d'autres réseaux, le terme Internet devient officiel. Les virus informatiques apparaissent et se diffuser partout dans le monde en quatre ans.

- 1984 : MS-DOS s'impose comme norme pour les 16 bits et l'entreprise Sinclair propose son micro-ordinateur 32 bits. Conception du langage C++, version orientée objet du langage C.

- 1985 : Apple lance le Macintosh, le premier micro-ordinateur équipé en standard d'une souris et d'un écran graphique((menus, icônes...), Hewlett-Packard sort la première imprimante laser et

Microsoft commercialise la première version de Windows. Les ordinateurs apprennent à penser à travers les systèmes experts, l'intelligence artificielle vient de naître.

- 1985 : apparition du CD-ROM de Philips et Sony.
- 1987 : IBM annonce une nouvelle gamme de micro-ordinateurs les PS2 ainsi qu'un nouveau système d'exploitation l'OS2. Apparition des GPS destinés au secteur civil, en particulier aux compagnies aériennes et maritimes qui deviendra accessible au grand public en 1995.
- 1988 : apparition des premiers home cinémas utilisant la technologie du laser disc et la console vidéo Mega Drive puis le jeu « Sonic .
- 1989 : Cinq micro-ordinateurs sur le marché, les IBM PC-XT et PC-AT (et compatibles), IBM PS2, la famille Macintosh (Macplus, Mac SE et Mac II), l'Atari ST et l'Amiga. D'autre part, le physicien et informaticien britannique **Tim Berners-Lee** (1955-) chercheur du Cern, invente un système hypertexte reliant des pages présentes sur Internet : le « World Wide Web , dont l'abbréviation « www des adresses Internet. création du protocole HTTP et Nintendo sort la console de poche Game Boy.
- 1990 : apparition de Windows 3, première interface graphique pour PC et du premier appareil photo numérique (noir et blanc à usage professionnel) sur le marché qui permet de charger les images sur ordinateur.
- 1991 : naissance de Linux et le premier site Internet est créé au Cern, le laboratoire européen de recherches nucléaires basé à Genève et des téléphones mobiles utilisant la technologie numérique apparaissent.
- 1992-1993 : le premier navigateur (ancêtre de Netscape, Mozilla, Firefox...etc) Mosaïc est créé au CERN de Genève, permettant de visualiser des pages Web (et donc ancêtre de Netscape, Mozilla, Firefox...). Son auteur **Marc Andreessen** fonde la société Netscape en 1994 et lance le premier navigateur grand public, sortie de Windows NT et premier microprocesseur Pentium de Intel. Á cette époque aussi, Sony lance le CD enregistrable, le MiniDisc qui sera exploité réellement à la sortie du baladeur MP3.
- 1994 : le site de vente en ligne Amazon est fondé aux états unis par Bezos (1964-) qui s'élagira en 2017 dans de nombreux pays. Naissance de Yahoo et des premiers sites d'e-commerce, Sony lance la PlayStation et Casio présente le premier appareil photo numérique grand public.
- 1995 : apparition du DVD-ROM et **Steve Jobs** rachète le studio Pixar qui produit « Toy Story , le premier long-métrage d'animation en images de synthèse qui eût un énorme succès. Panasonic et Sony introduisent les premiers Caméscopes digitaux et l'entrepreneur **Jeff Bezos** crée Amazon et **Pierre Omidyar**, eBay. Windows 95 généralise l'interface graphique sur les PCs.
- 1996 : sortie des lecteurs et films DVD au Japon et en 2002, les ventes de DVD dépasseront celles des cassettes vidéos.
- 1997 : l'ordinateur Deep Blue d'IBM bat le champion du monde des échecs **Gary Kasparov**, Philips lance la télévision à écran plasma et sortie de Windows 97.
- 1998 : le moteur de recherche Google est créé par **Sergey Brin** et **Larry Page**, deux étudiants de Stanford. En Corée du Sud, Saehan Information Systems conçoit le premier baladeur MP3, capable de stocker l'équivalent de quatre CD. Á cette date aussi est sortie de Windows 98.

- 1999 : les télévisions à écran LCD innonde le marché et le Wi-Fi permet de se connecter sur la Toile sans prise électrique. Aux Etats-Unis, lancement du site Napster, qui permet de télécharger gratuitement des morceaux de musique.

- 2000 : l' année noire : l'explosion de la bulle Internet qui menace le système informatique mondial au tournant du millénaire. Les clés USB et les cartes mémoires commencent à remplacer les disquettes sur les nouveaux ordinateurs personnels. Problèmes du « SPAM » ou pourriel, attaques via des virus de toutes sortes et mise en œuvre du protocole sécurisé HTTPS.

- 2001 : Apple, dont Steve Jobs est redevenu président directeur général en 1997, sort son application iTunes, qui permet d'acheter légalement de la musique en ligne et simplifie le transfert des chansons depuis l'ordinateur vers un baladeur maison, le iPod. L'encyclopédie numérique Wikipédia est créée par **Jimmy Wales** (1966-), libre, multilingue et consultable gratuitement sur internet. Sortie de Windows XP qui durera plus que les versions précédentes de Windows.

- 2002 : Six sociétés (dont Intel, NEC, Philips, HP) lancent les clés USB 2.0 pour le grand public. La firme canadienne RIM sort le premier BlackBerry, optimisé pour l'envoi et la réception d'e-mails.

- 2003 : apparition de MySpace, un site permettant l'échange de messages, photos et chansons. C'est l'ère des réseaux sociaux qui commence.

- 2004 : Mark Zuckerberg (1984-), un étudiant de Harvard, lance le réseau social Facebook, dont le développement est fulgurant. En Estonie, naissance du logiciel Skype, qui permet de passer des appels téléphoniques à travers Internet.

- 2005 : Alors que le nouveau site hébergeur de vidéos YouTube cartonne, Google lance son service de cartes et plans en Europe.

- 2006 : Trois nouveautés : le Sony Reader, premier e-book grand public ; la console Wii de Nintendo, dont la manette sans fil peut être remplacée par un objet (épée, raquette) tenu par le joueur ; et les lecteurs et DVD Blu-ray à haute définition. Création de Twitter.

- 2007 : L'année est marquée par l'apparition de Windows Vista qui prend en charge plus de mémoire vive jusqu'a 4 Go pour la version 32 bits et 128 Go pour la version 64 bits et la sortie de l'iPhone d'Apple, dont l'ergonomie et le design redéfinissent la catégorie des smartphones, et par le lancement du mini-ordinateur Eee d'Asus, doté du logiciel gratuit Linux.

- 2008 : en France, lancement de la télévision numérique terrestre à haute définition qui stimule la vente de téléviseurs à écran plat.

- 2009 : apparition de Windows 7, une version améliorée de Vista.

- 2010 : Apple inaugure l'ère de la tablette avec l'iPad1 avec un écran tactile présenté par Steve Jobs, président-directeur général d'Apple à l'époque, qui fût la plus vendue dans le monde. De son côté, Samsung et d'autres constructeurs dévoilent leurs télévisions 3D. Lancement d' Instagram par l'Américain **Kevin Systrom** (1983-) et le Brésilien **Michel Mike Krieger** (1986-), qui intègrera par la suite Facebook.

- 2010 : Sony, abandonne la fabrication de disquettes $5\frac{1}{2}$ pouces, le dernier format de disquettes encore existant, après les 8 pouces et les $5\frac{1}{4}$ pouces. Apparition du Big Data et l'apprentissage profond.

- 2011 : apparition de l'application mobile gratuite de de la société Snap Inc., permettant le partage de photos et de vidéos, disponible sur plateformes mobiles iOS et Android. Elle a été conçue et développée par des étudiants de l'université Stanford en Californie : **Evan Spiegel** (1990-), **Bobby Murphy** (1988-) et **Reggie Brown** (1876–1961).
- 2012 : sortie de Windows 8. item 2015 : sortie de Windows 10. Création de OpenAI, l'entreprise spécialisée dans le raisonnement artificiel OpenAI par Elon Musk (qui quittera la société en 2018) et Sam Altman. OpenAI est à l'origine des premiers logiciels d'IA grand public DALL-E et ChatGPT.
- 2016 : TikTok est l'application la plus téléchargée de l'année 2020.
- 2017 : En novembre, IBM déclare avoir développé un ordinateur quantique capable de gérer 50 bits quantiques (qubits), un nouveau record pour l'industrie.
- 2018 : Google a marqué les esprits en présentant Bristlecone, un processeur quantique avec 72 qubits. Début du déploiement de la technologie 5G aux États-Unis. La fin du support de Windows 7, après avoir duré 10 ans, il sera toujours utilisable, mais il présentera de plus en plus de failles de sécurité sauf pour les entreprises ayant souscrit un abonnement payant, auprès de Microsoft.
- 2019 :
- 2020 : Forte croissance de l'utilisation de l'Informatique en nuage (Cloud computing) et de la visioconférence à l'occasion de la crise sanitaire.
- 2021 :Sortie de Windows 11. IBM annonce avoir gravé la première puce en technologie 2 nm. Investissements importants dans le domaine de l'Informatique quantique.
- 2022 : Sortie de Windows Server 2022, système d'exploitation Microsoft pour serveurs. Il fournit une sécurité multicouche, des fonctionalités pour serveur hybride avec une gestion de machine virtuelle améliorée.
- 2023 : L'outil de conversation automatisé basé sur le langage naturel qui utilise l'IA ChatGPT dépasse les 100 millions d'utilisateurs dans le monde.
- 2024 : Panne informatique mondiale le 19 juillet 2024, impactant les systèmes Windows Server utilisant l'EDR CrowdStrike. La panne causée par une mise à jour logicielle défectueuse a fortement touché les compagnies aériennes, les banques et les hôpitaux.
- 2025 : Microsoft continue avec Windows 11 avec des mises à jour régulières, améliorant Windows 11 avec des fonctionnalités nouvelles. Fin du support étendu de Windows 10 et la sortie prévue de Windows 12 semble décalée, probablement pour une intégration plus poussée de l'intelligence artificielle.

1.3 Bref historique sur l'algorithmique

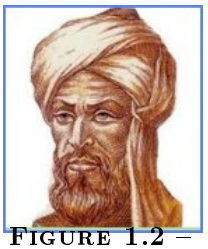


Figure 1.2 – Al Khwarizmi.

En informatique, on est passé des algorithmes aux machines à calculer puis les machines pensantes. L'algorithmique consiste à concevoir et optimiser des méthodes de calcul pour la résolution de problèmes. Le terme algorithme est du au mathématicien, géographe et astronome Perse (Irak actuel) **Abu Jaffar Mohammed Ibn Moussa Al Khwarizmi** (780-850) qui vécut à Bagdad dans son célèbre ouvrage le traité d'algèbre "Hisab al-jabr w'al-muqabala" et auquel on doit aussi les chiffres arabes et les algorithmes pour la résolution d'équations linéaires et quadratiques.
Les premières traces de l'algorithmique remontent à l'époque des Babyloniens (Iran actuel) avec des algorithmes rudimentaires pour le commerce et les impôts dont le code d'Hammourabi, roi de Babylone (1750 av. JC). Puis, en Grèce avec la logique et l'art de la démonstration d'Aristote (330 av. JC), l'algorithme d' Euclide (300 av. JC) qui permet de calculer le plus grand diviseur commun (PGCD) de deux nombres et celui d'Archimède pour le calcul de la valeur de π. L'algorithme a trouvé sa forme achevée grâce au mathématicien anglais **Alan Turing** (1912-1954) auteur du traité The Art of Computer Programming où il décrit de très nombreux algorithmes et posa des fondements mathématiques rigoureux pour l'analyse des algorithmes (Cordier, 2015).

Il existe plusieurs définitions d'un algorithme, celle qu'on retient est que c'est un ensemble de règles permettant la résolution d'un problème donné. Ces règles se présentent sous forme d'une suite d'opérations élémentaires obéissant à un enchaînement déterminé.

CHAPITRE 2

QUELQUES NOTIONS PRÉLIMINAIRES

Sommaire

Figures

Ce chapitre introductif présente quelques notions générales relatives à l'architecture et la structure d'un ordinateur. Le lecteur pourra avoir une idée sur les composants principaux d'un ordinateur avant d'aborder les notions d'algorithmique et de programmation.

2.1 Structure simplifiée d'un ordinateur

Un ordinateur est une machine capable de lire et restituer de l'information (entrées/sorties), traiter l'information (processeur) et stocker l'information (mémoire) qui y est représentée sous forme de bits (binary digits) ou positions binaires (0 ou 1). Le bit est donc l'information minimale représentable qui est traduite par la présence ou l'absence d'une tension ou d'un courant électrique sur un fil conducteur. Pour assurer ces fonctions, l'architecture de l'ordinateur est fondée sur deux couches, la couche matérielle (hardware) qui constitue l'ensemble des composants physiques et la couche logicielle qui permet aux différentes composantes matérielles de fonctionner comme il est illustré dans la Figure 2.1.

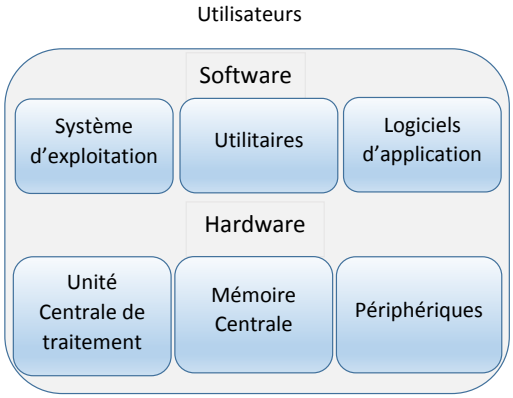


Figure 2.1 – Structure en couche d'un ordinateur.

2.1.1 Couche Matérielle

La couche matérielle consiste en trois circuits principaux selon le modèle de John von Neumann  (Burks et al., 1963) que nous pouvons décrire de manière simpliste comme suit :

L'Unité Centrale de traitement dite CPU (Central Processing Unit ou processeur) : qui exécute les programmes. Elle est constituée de deux unités, l'Unité de Commande (Cmde) ou de Contrôle (CTRL) et l'unité arithmétique et logique (U.A.L). La première décortique et organise les opérations à réaliser par l'ordinateur qui peuvent être de calcul ou comparaison ou bien de lecture/écriture dites d'entrée/sortie (E/S). La seconde réalise les opérations arithmétiques (addition, soustraction, multiplication et division) et logiques (tests et comparaisons).

La Mémoire Centrale : où résident les informations qui sont les données d'entrée et de sortie (résultats) et les programmes. On distingue deux types de mémoires, la mémoire vive dite RAM (Random Access Memory) qui est une mémoire à accès aléatoire volatile puisque les informations y sont stockées provisoirement, c'est à dire, le temps que la machine soit sous tension mais une fois hors tension toute l'information résidente en mémoire disparaît. La seconde mémoire est la mémoire morte dite ROM (Read Only Memory), c'est une mémoire à lecture seule non volatile puisque les informations qui y sont stockée sont programmées lors de la fabrication de la machine et ne peuvent ni être modifiées ni effacées, seulement lues et elles y résident de manière permanente même lors de la mise hors tension du système. Parmi ces informations, le BIOS (Basic Input/Output System) qui permet de charger le système d'exploitation de la machine et le POST (Power-On Self Test) qui est exécuté automatiquement à l'amorçage du système, permettant de faire un test du système. La volatilité de la RAM a induit le recours aux mémoires auxiliaires (secondaires) dites mémoires de masse qui offrent à l'utilisateur la possibilité de stocker aussi longtemps qu'il le désire de grandes quantités d'informations et à faible coût.

Les périphériques : qui permettent les échanges avec l'extérieur et sont soit d'entrée, soit de sortie ou les deux au même temps. Ils permettent la communication avec l'homme (clavier/écran ou console, souris, microphone, lecteur de DVD, imprimante, scanner, haut-parleurs, caméra Web, modem, projecteur électronique, carte réseau) et le stockage secondaire (disque dur, disques compactes ou CD, clés USB). Un organe spécialisé souvent dit unité d'échange placé entre la mémoire et les périphériques pouvant être munie parfois d'un processeur interne et une mémoire tampon assure l'interface homme-machine via des canaux spécialisés appelés bus qui sont des ensembles de fils conducteurs parallèles permettant de véhiculer des bits d'information (adresses ou données) ou des signaux. On distingue trois types de bus, le bus de données (bidirectionnel) qui véhicule les données du processeur vers la mémoire centrale ou les périphériques et inversement, le bus d'adresse (unidirectionnel) qui véhicule les adresses mémoire vers les différents organes de l'ordinateur et enfin, le bus de commande ou contrôle (bidirectionnel) qui transporte les ordres et les signaux de contrôle et de synchronisation de l'unité de commande vers les autres organes de la machine, comme il est illustré dans la Figure 2.2.

La carte mère : les différents composants électroniques de l'ordinateur sont assemblés en une carte principale dite carte mère (mother board) qui est un circuit imprimé rectangulaire muni d'une pile d'alimentation et de connecteurs (slots) pour greffer des composants et périphériques supplémentaires comme il apparait dans la Figure 2.3. La ROM sur laquelle est enregistré le BIOS, programme gérant la configuration "de base" du matériel et se chargeant de faire le lien avec votre système d'exploitation (Windows, Linux...). Ces réglages sont conservés en mémoire même en l'absence de courant grâce au CMOS, alimenté par la pile de la carte mère.

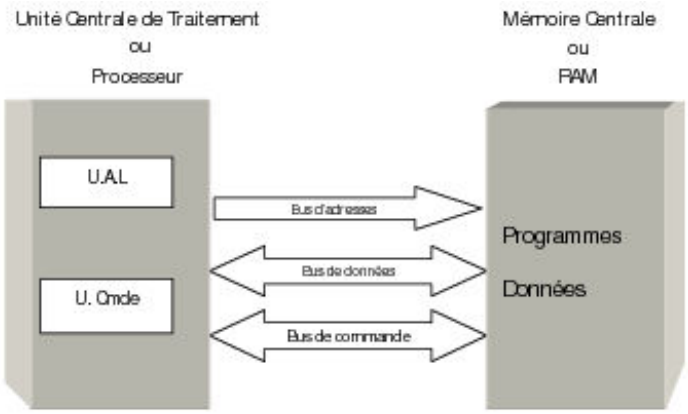


FIGURE 2.2 – Les différents types de bus véhiculant des bits d'information (adresses ou données) ou des signaux de commande.

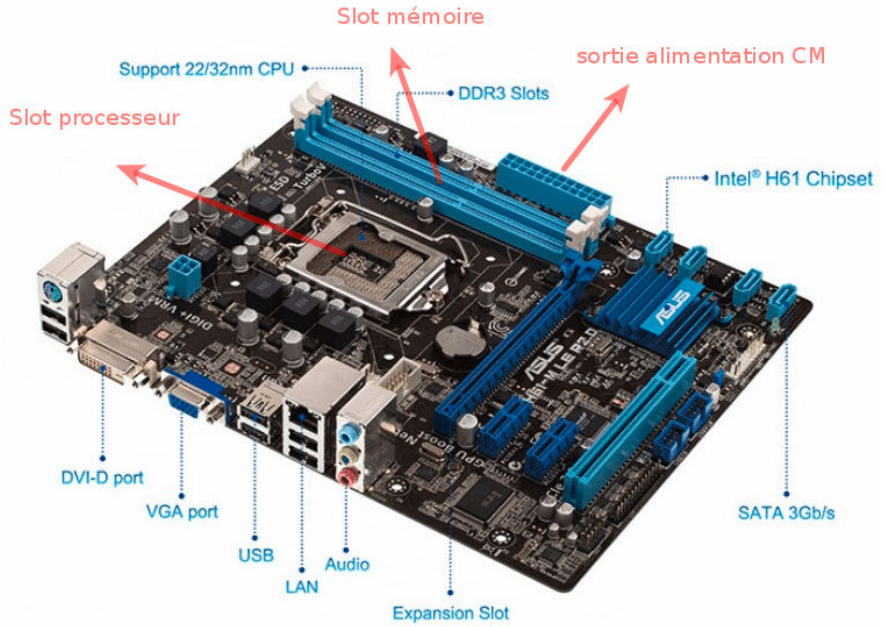


FIGURE 2.3 – Carte mère[1], où sont branchés toutes les composantes d'un ordinateur (processeur, RAM, ROM, disque dur, cartes).

2.1.1.1 La mémoire centrale

La mémoire centrale (MC) dite aussi principale ou vive est une mémoire à accès aléatoire (Random Access Memory) dont le contenu est volatile ce qui veut dire que l'information nécessaire au traitement effectué par le processeur y est stockée temporairement. En cas de coupure de courant, la mémoire est initialisé à zéro (tout s'envole). Elle contient l'information en cours d'emploi comme le noyau du système d'exploitation, les programmes en cours d'exécution et les fichiers de données en cours de traitement. Elle est structurée en bits (**b**inary dig**its**), pouvant prendre deux états stables 0 et 1 (codage binaire) par analogie à un interrupteur électrique (allumé/éteint). Ces bits sont regroupés par huit, chaque groupe est un octet (byte). Il existe deux type de mémoire RAM. La mémoire vive dynamique (DRAM) qui est la RAM classique et la mémoire vive statique (SRAM)à accès rapide, utilisée notamment dans les processeurs comme mémoire cache dans le but d'augmenter les performances et réduire le temps d'accès aux données. La MC apparait sous forme de barettes dans la carte mère et peut être augmentée

1. https://www.superprof.fr/blog/l-interieur-d-un-personal-computer/

grâce à l'ajout de barrettes de mémoire mais elle est assimilée à un ensemble de cellules (cases) pouvant contenir chacune une information dites alors mots mémoire. Chaque mot mémoire est repéré par un nombre entier dit adresse dont la grandeur dépend de la taille de la mémoire et est caractérisé par un contenu (instruction ou donnée). La capacité d'une mémoire s'exprime en fonction du nombre de mots mémoire ainsi que du nombre de bits par mot. Un mot mémoire peut être de 16, 32, 64 bits ou plus. Ces différences entre calculateurs a une incidence directe sur les valeurs numériques que l'on peut représenter en mémoire. Pour exprimer les tailles des MC, on utilise souvent les mesures suivantes :

- 1 octet = 2^3 bits = 8 bits. Un mot d'un octet prend les valeurs de 0 à 255.
- 1 Ko (Kilo octet) = 1024 octets = 2^{10} octets.
- 1 Mo (Mega octet) = 1024 Ko = 2^{10} Ko.
- 1 Go (Giga octet) = 1024 Mo = 2^{10} Mo.
- 1 To (Téra octet) = 1024^4 octets = 2^{40} octets.

Ainsi, une MC de 256 mots permet de représenter un mot sur $log_2(256) = log_2(2^8) = 8\ bits$ donc si on a 32 bits d'adressage, on peut représenter 2^{32} mots différents. Le temps nécessaire à la lecture/ecriture d'un mot mémoire est dit temps d'accès, il varie de la nano (10^{-9}) à la micro (10^{-6}) seconde. Analogue au mot mémoire, le registre qui se trouve quant à lui dans l'unité centrale de traitement permet un accès rapide et peut stocker n bits dans une machine à n bits.

L'acquisition et la restitution de l'information sont effectuées grâce à deux registres associés à la MC, le registre d'adresse mémoire contenant l'adresse du mot à lire ou écrire et le registre d'information mémoire qui reçoit l'information lue à partir de la mémoire ou destinée à y être écrite.

2.1.1.2 Unité Centrale de traitement

L'Unité Centrale de traitement est composée en plus de L'unité arithmétique et logique (U.A.L) et de l'unité de commande (Cmde ou CTRL) d'au minimum :

- un registre de données RD : destiné à contenir les données transitant entre l'unité centrale de traitement et l'extérieur.
- un accumulateur Acc : destiné à contenir les opérandes et les résultats des opérations effectuées par l'UAL.
- un registre d'état : indiquant l'état du déroulement de l'opération en cours, composé d'un ensemble de bits indicateurs (drapeaux ou flags).

L'UAL contient tous les circuits électroniques qui réalisent les opérations arithmétiques (+,-,*,/), logiques (négation, et , ou, ou exclusif) et le décalage. l'unité de commande est le chef d'orchestre, elle est chargée de commander et de gérer tous les organes de l'ordinateur (contrôler les échanges, gérer l'enchaînement des différentes instructions, etc...) Elle est composée :

- d'un registre instruction RI : contient l'instruction en cours d'exécution qui y reste jusqu'à son accomplissement, sa taille est la même que celle d'un mot mémoire.
- d'un compteur ordinal CO : permet de calculer l'adresse de la prochaine instruction à exécuter.

- d'un registre adresse RA : contient l'adresse de la prochaine instruction à exécuter.
- d'un décodeur d'instructions : analyse l'instruction à exécuter figurant dans le registre instruction et met en œuvre l'UAL ou l'unité d'échange ou bien la mémoire centrale.
- d'une horloge (clock) : exprime en Hertz (Hz) ou plutôt en MegaHertz (MHz) ou GigaHertz (GHz) la cadence à laquelle travaille l'ordinateur en lançant des "tops" à intervalles de temps réguliers où chaque "top" correspond à l'exécution d'une micro-commande. Elle permet de dater les opérations et de synchroniser l'ensemble des composants de l'unité centrale de traitement.

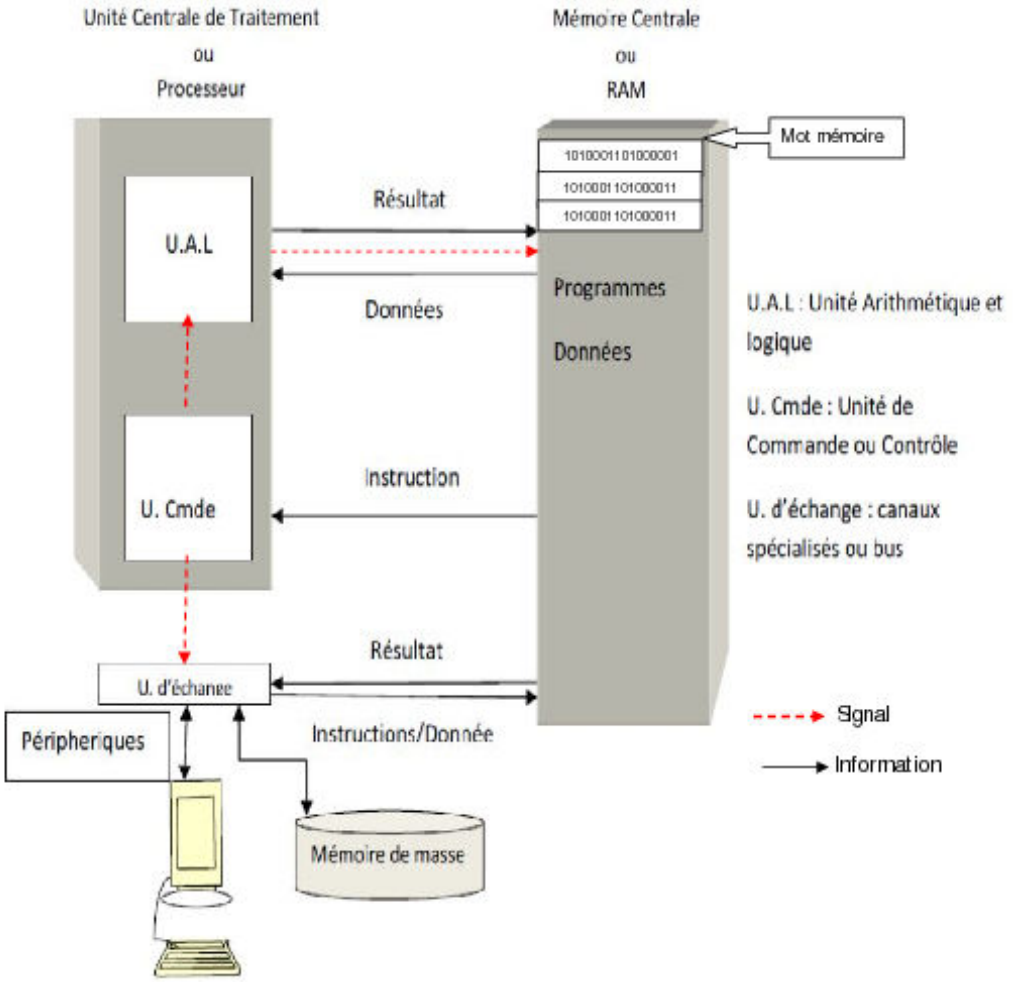


Figure 2.4 – Schéma fonctionnel d'un ordinateur.

En général, l'exécution d'un programme nécessite tout d'abord de charger le programme et les données en MC. Ensuite, les instructions du programme sont amenées une par une (séquentiellement) à l'unité de Cmde. L'instruction à exécuter dont l'adresse est pointée par le CO est d'abord chargée dans le RI, le contenu du CO est alors incrémenté sauf en cas de branchement à une instruction en amont ou en avale. Ensuite, l' unité de Cmde décode le contenu du RI (code opérande et champs adresse des données) et localise en MC les données de l'instruction. Une fois l'instruction analysée et les données repérées, elle déclenche le traitement en envoyant des signaux soit à l'UAL dans le cas d'une opération de calcul ou de comparaison, soit à l'unité d'échange dans le cas d'une opératioon d'entrée/sortie. Le traitement peut aussi necéssiter de faire appel à la MC pour y stocker le résultat comme il est illustré dans la Figure 2.4.

2.1.1.3 Unité d'échange ou d'entrée/sortie

L'unité d'échange permet le transfert d'information entre l'unité centrale de traitement et les unités périphériques. trois dispositifs sont mis en oeuvre, le DMA (Direct Memory Access), les bus et le canal d'E/S. Le DMA est composé généralement d'un registre d'adresse, d'un registre de comptage, d'un registre de commande, d'une mémoire tampon servant à stocker les informations et d'un processeur.

Les informations transférées sont rangées dans la MC séquentiellement. Pour initialiser un tel transfert, les instructions d'E/S doivent fournir à l'unité d'échange via le DMA l'adresse de rangement de la première information, l'identifiant de l'unité périphérique concernée, le sens du transfert et le nombre d'informations à transférer. Ensuite, l'unité d'échange se charge du transfert en ajoutant 1 à l'adresse de rangement et en retranchant 1 au nombre d'informations à transférer et une fois toute l'information épuisée, le DMA signalera à l'unité centrale de traitement la fin du transfert.

2.1.1.4 Les périphériques

Les périphériques sont des matériels spécialisés qui permettent de réaliser l'une des fonctions suivantes :

- La communication avec l'homme qu'on peut structurer en deux catégories :

  Les périphériques réseau qui servent à communiquer avec d'autres ordinateurs comme les modems et les circuits électroniques intégrés à la carte mère dits cartes (son, vidéo, graphique).

  Les périphériques d'E/S qui servent à dialoguer avec l'utilisateur, on distingue les périphériques d'entrée (clavier, souris, scanner, microphone, lecteur de codes à barres, capteurs,... etc.), les périphériques de sortie (écran, imprimante, haut-par–leurs, table traçante,...etc.) ou les deux en même temps (modem, caméra Web, lecteurs de CD et DVD).

- Le stockage de l'information : le coût élevé de la MC et sa volatilité à induit le recours aux mémoires secondaires qu'on résume en ce qui suit :

  Le disque dur (hard disk) C'est l'unité de stockage principale de l'ordinateur, il a l'apparence d'un boîtier rectangulaire, comme il est illustré dans la Figure 2.5. Il est relié à la carte mère grâce à une nappe (câble plat). C'est là où est sto–cké le système d'exploitation, les programmes et les fichiers de données. C'est un ensemble amovible de plateaux circulaires métalliques rigides enduits d'un re–vêtement magnétique montés sur un même axe animé d'une vitesse de rotation constante. La lecture/écriture se fait par un ensemble de têtes de lecture/écritures (peignes) pouvant se déplacer par translation. L'écriture se fait suivant des cercles concentriques appelées pistes où les bits sont représentés en série. Ses principales caractéristiques sont la vitesse de transfert, la vitesse de rotation des plateaux et la capacité de stockage. Aujourd'hui, les disques durs peuvent emmagasiner des centaines de Giga-octets de données.

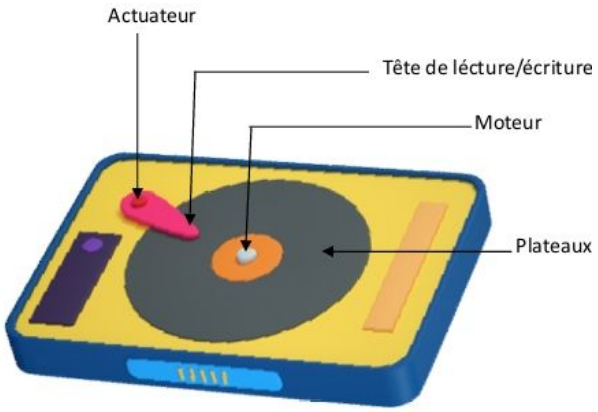


Figure 2.5 – Structure physique d'un disque dur [2].

Les supports optiques Les supports optiques consistent en disques compacts inscriptibles et réinscriptibles et disques vidéos numériques (Digital Video Disks ou DVD), nécessitent un graveur de disques compacts qui est en général vissé au boîtier de l'ordinateur et connecté à la carte mère, ses performances sont relatives aux vitesses de lecture et d'écriture, ainsi qu'aux formats qu'il supporte (CD, DVD, Blu-ray,$\cdots$).

La clé USB (Universal Serial Bus) C'est un accessoire amovible composé de mémoire flash et caractérisé par sa vitesse de transfert et sa capacité de stockage. Elles sont très populaires actuellement due à leur grande capacité de stockage.

Le stockage de l'information en dehors de la mémoire centrale nécessite la manipulation de fichiers qui représentent les supports où sera stockée l'information. Un fichier peut contenir des programmes, des données ou tout autre type d'information. C'est le système d'exploitation qui fournit l'ensemble des opérations de gestion des fichiers comme les appels système pour les créer, les détruire, les écrire ou les modifier.

2.1.2 Couche logicielle

La couche logicielle regroupe tous les programmes qu'utilise l'ordinateur, le système d'exploitation qui pilote les ressources matérielles et l'installations de divers programmes, Les logiciels d'application qui sont des programmes dédiés à des applications bien spécifiées et les utilitaires.

2.1.2.1 Le Système d'exploitation

Un ordinateur sans son logiciel ne sert pas à grand chose. Au fil des années les logiciels ont évolué dans plusieurs directions. L'indispensable système d'exploitation qui nous rappelle l'universel MS-DOS (MicroSoft Operating System) pour les micro-ordinateurs personnels (PC ou Personal Computers) est aujourd'hui en plusieurs modèles qui sont tout autant performants les uns que les autres, quant aux logiciels d'application et les utilitaires, le nombre ne cèsse de croître avec les besoins croissants des utilisateurs. Le système d'exploitation OS (Operating System) est un ensemble de programmes qui assurent la gestion de l'ordinateur et la liaison de ses composants matériels et les applications des

2. https://www.crucial.fr/articles/pc-builders/what-is-a-hard-drive

utilisateurs. Il est structuré en deux couches, le noyau (kernel) qui est l'ensemble logiciel indivisible minimal, systématiquement chargé au démarrage et les logiciels de base. Le noyau permet la gestion du processueur, de la mémoire, des travaux, l'allocation des ressources partagées, des E/S, des échanges réseaux, ...etc. Les logiciels de base fournissent des services pour un environnement de programmation de base comme les interfaces graphiques, les éditeurs de textes, compilateurs, assembleurs, interpréteurs de commandes, éditeurs de liens, programmes de gestion des communications et réseaux, ...etc. Le système d'exploitation peut utiliser deux sortes d'interface avec l'utilisateur :

- interface entre l'ordinateur et les applications logicielles (API).
- interface entre l'ordinateur et l'utilisateur (IHM).

Le système d'exploitation comprend 4 parties essentielles :

1. La gestion des programmes.
2. Les entrées/sorties.
3. La gestion de la mémoire.
4. Le système de fichiers.

2.1.2.2 Les utilitaires

Ce sont des services de base aux utilisateurs, comme l'interfaces graphique, l'interpréteur de commandes, et les gestionnaires de divers fonctions (impression, recherche de courrier, ...etc).

2.1.2.3 Les logiciels d'application

Ce sont des logiciels destinés aux utilisateurs comme des applications spécifiques à divers domaines pratiques comme la Bureautique (traitements de texte et tableurs), la comptabilité (facturation, paye, stocks,$\cdots$,etc), la navigation Internet, les jeux vidéo, la gestion de base de données (Oracle, Access, MySQL,...etc), la conception assistée par ordinateur (CAO), $\cdots$, etc.

Les systèmes d'exploitation les plus répandus sont Windows développé par Microsoft depuis le milieu des années 80 qui a englobé le MS-DOS (pour les PC) et dont plusieurs versions ont succédé (Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8 et Windows 10). Mac OS développé par Apple pour ses ordinateurs personnels dont Macintosh qui a été l'un des premiers à posséder une interface graphique (avec des menus, des fenêtres,... etc.), la version la plus récente est Mac OS X. GNU/Linux (pour les PC et les serveurs) basé sur le noyau Linux créé en 1991 qui existe maintenant en plusieurs distributions (Ubuntu, Debian, Fedora, OpenSuse,...etc). Unix (pour les serveurs). Le stockage des fichiers sur les mémoires externes par NTFS pour Windows, HFS Plus pour Mac OS et ext3 et ext4 ou autre pour Unix et Linux. Parmi les OS pour smartphones et tablettes tactiles, on peut citer Android (basé sur le noyau Linux et racheté par Google), iOS (développé par Apple pour son smartphone iPhone et sa tablette tactile iPad) et Windows 10 mobile (développé par Microsoft pour sa marque Nokia) qui reste relativement assez peu utilisé.

CHAPITRE 3

ALGORITHME SÉQUENTIEL SIMPLE

Sommaire

Figures

Tableaux

3.1 Notion de langage et langage algorithmique

L'algorithme est un concept fondamental en informatique. Apparu au IX^{me} grâce au mathématicien et astronome perse Abu Jaafar Mohammed Ibn Moussa El Khawarzmi . Intuitivement, un algorithme est toute procédure de calcul non ambiguë prenant en entrée une ou plusieurs valeurs et produisant en sortie une ou plusieurs valeurs résultat. Il représente un ensemble de règles a appliquer dans un ordre déterminé afin de résoudre un problème bien spécifié, transformant les données d'entrée en un résultat en sortie.
Exemples d'algorithmes :

- Recette de cuisine.
- Recherche d'une valeur dans une liste de valeurs.
- Recherche d'un mot dans un dictionnaire.
- Recherche du plus court chemin dans un graphe.
- Extraction des valeurs propres d'une matrice.
- Résolurion d'un système d'équations linéaires.
- Tri d'une liste de valeurs.

L'algorithme précise donc la relation entre les données d'entrée et le résultat cherché. Trois questions sont primordiales lors de l'écriture d'un algorithme, consistant en :

- Quelles sont les données d'entrée ?
- Qu'elle est la nature du résultat cherché ?
- Quelle est la méthode de résolution ou quelles sont les formules à appliquer pour obtenir le résultat à partir des données d'entrée ?

Le langage de description d'algorithmes (LDA) peut être textuel rédigé soit en langage naturel soit sous forme de pseudo-code ou bien graphique (organigramme).

Texte libre : la description des différentes étapes est faite en utilisant le langage naturel comme les recettes de cuisine et les modes d'emploi (guide, notice ou manuel d'utilisation). Cette façon de décrire un algorithme est dite informelle. Nous donnerons dans ce qui suit trois exemples où la partie environnement représente les objets (données) sur lesquelles opèrent les actions.
☞ Exemple1 : le problème est comment préparer de bonnes crêpes ? La recette est l'algorithme à suivre (exprimé en langage naturel). Les objets du problème sont : les ingrédients, le matériel de cuisine et le résultat (les crêpes), représentés dans la Table 3.1. La résolution du problème est illustrée dans la Figure 3.1.

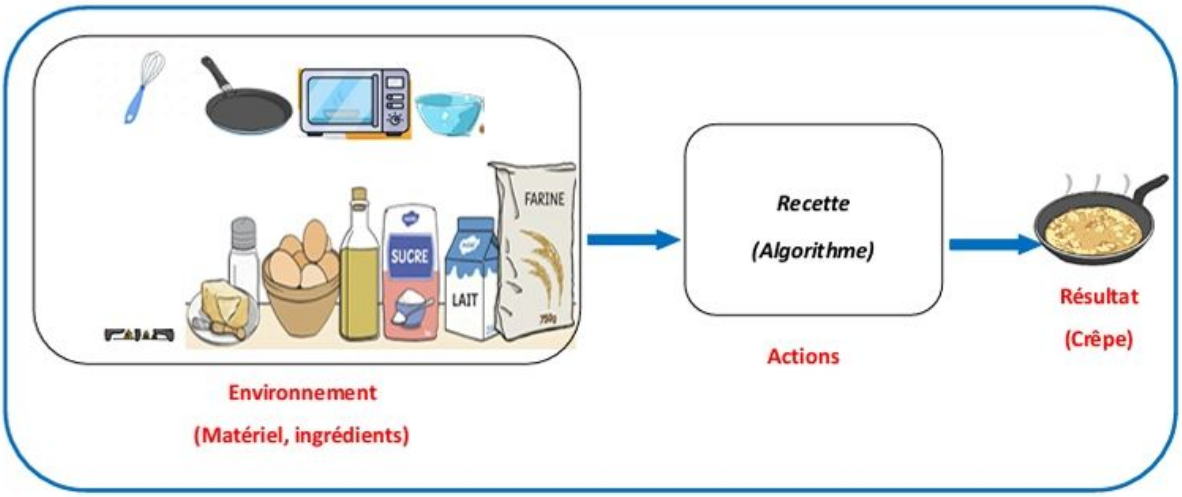


Figure 3.1 – Comment préparer des crêpes ?

Environnement (les objets)	Actions
œufs, farine, sucre, lait, beurre, poêle, louche, bol, fouet, feu, four à micro-ondes, crêpes	1/ Faire fondre le beurre dans un bol au four à micro-ondes 2/Mélangez la farine tamisée 3/Ajouter le sucre 4/Ajoutez les œufs 5/Ajouter le lait 6/Battre le tout avec un fouet 7/Faites chauffer sur le feu une noix de beurre dans la poêle 8/Versez y une louche de pâte 9/Faites cuire vos crêpes de chaque côté

Table 3.1 – Problème : comment faire des crêpes ?.

☞ Exemple2 : Problème de passage de rivière du chou, la chèvre et le loup.
Un berger, en compagnie d'un loup, de sa chèvre et d'un chou sur une rive (rive1), veut traverser une rivière pour aller à rive2 comme le montre la Figure 3.2. Il dispose d'une petite barque à 2 places (seul lui doit diriger la barque). De plus, si la chèvre et le chou sont ensemble sur une rive quand le berger s'éloigne, la chèvre mange le chou et si le loup et la chèvre sont ensemble quand le berger s'éloigne, le loup mange la chèvre. Aider-le à faire traverser le loup, sa chèvre et le chou sur l'autre rive. Préciser les objets du problème et les étapes à réalise dans l'ordre.

Figure 3.2 – Problème de passage de rivière du chou, la chèvre et le loup.

Les objets du problème sont : Le berger, le loup, le chou, la chèvre, la barque, la rive1 et la rive2.
Les objets et les actions sont représentés dans la Table 3.1.

Environnement (les objets)	Actions
Berger, loup, le chou, chèvre, barque, rive1, rive2	1/Déplacer la chèvre de rive1 vers rive2 2/Revenir à rive1 3/Déplacer le chou vers rive2 4/Déplacer la chèvre de rive2 vers rive1 5/Déplacer le loup de rive1 vers rive2 6/Revenir à rive1 7/Déplacer la chèvre de rive1 vers rive2

TABLE 3.2 – Problème de passage de rivière du chou, la chèvre et le loup.

☞ Exemple3 : Problème des tours d'Hanoi.
Le jeu de réflexion imaginé par le mathématicien français **Edouard Lucas** (1842-1891)  (Lucas, 1892) consiste en trois tours (départ, arrivée ou destination et intermédiaire ou temporaire) et de 64 disques, tous de rayons différents comme le montre la Figure 3.3.
Dans cet exemple pour simplifier l'énoncé, on considère uniquement 3 disques (un grand, un moyen et un petit disque). Au départ les 3 disques sont au niveau de la tour de départ, rangés par taille, le plus grand tout en bas. Le but est de déplacer ces disques pour les amener sur la troisième tour en suivant les règles suivantes :

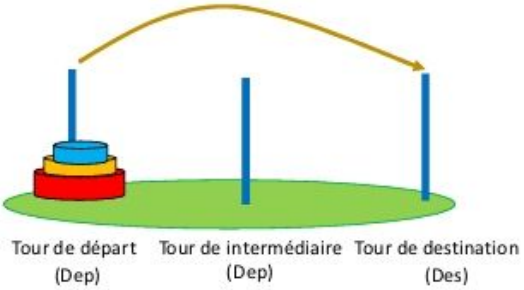


FIGURE 3.3 – Problème des tours d'Hanoi.

- on ne peut déplacer plus d'un disque à la fois ;
- à chaque étape et sur chaque tour, un disque ne peut être placé qu'au-dessus d'un disque de rayon plus grand ou le vide.

Il faut donc déplacer les trois disques de la tour de départ vers la tour de destination en un nombre minimum d'étapes. Préciser les objets du problème et les étapes à réalise dans l'ordre.
Les objets du problème sont : La tour de depart (Dep), la tour de destination (Des), La tour intermédiaire (Temp), le grand disque (Gdisque), le disque moyen (Mdisque) et le petit disque (Pdisque). Les objets et les actions sont représentés dans la Table 3.1.

Environnement (les objets)	Actions
Dep, Des, Temp, Gdisque,Mdisque, Pdisque	1/Déplacer Pdisque de Dep vers Des 2/Déplacer Mdisque de Dep vers Temp 3/Déplacer Pdisque de Des vers Temp 4/Déplacer Gdisque de Dep vers Des 5/Déplacer Pdisque de Temp vers Dep 6/Déplacer Mdisque de Temp vers Des 7/Déplacer Pdisque de Dep vers Des

TABLE 3.3 – Problème des tours d'Hanoi.

La Figure 3.4 détaille les étapes à effectuer.

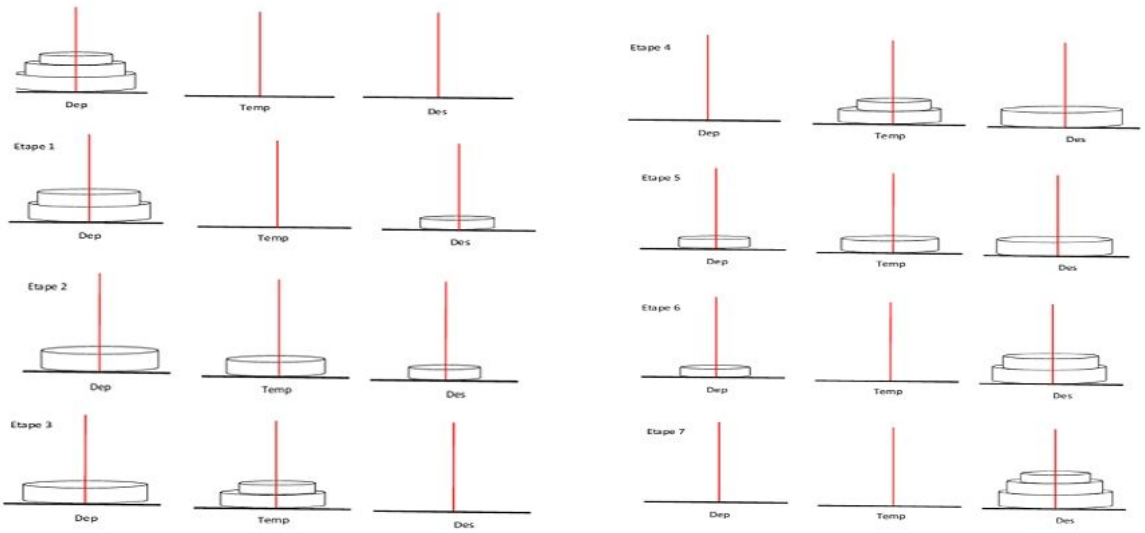


FIGURE 3.4 – Déroulement des étapes pour $n = 3$ disques.

Pseudo-code : La description des opérations est faite de façon formelle en utilisant un langage informatique simplifié ressemblant à du code en langage de programmation où l'écriture des données et des opérations à réaliser suit une syntaxe bien spécifiée (certaines conventions) basée sur les instructions disponibles dans la plupart des langages de programmation. Cette représentation est largement utilisée jusqu'à aujourd'hui. Dans tout ce qui va suivre, la structure du pseudo-code utilisé sera celle du langage de programmation Pascal inventé dans les années 1970 à Zürich par Niklaus Wirth adapté à l'enseignement de la programmation dans les écoles. Un algorithme sera organisé en trois parties distinctes. La première partie représente l'en-tête de l'algorithme, elle est constitué du mot Algorithme ou Algo suivi du nom de l'algorithme. La deuxième partie placée après l'en-tête de l'algorithme est réservée aux déclarations de variables, des constantes et des types nouveaux (autres que les types simple ou standards), dits types personnalisés. La troisième partie dite corps de l'algorithme, contient les instructions à exécuter. La structure générale d'un algorithme est donc la suivante :

```
Algorithme  <identificateur> ; (* En-tête de l'algorithme *)
[Déclarations]  (* Partie déclarative  de l'algorithme *)
Début  (* Début du corps (instruction exécutables) de l'algorithme *)
<instructions>;
Fin.  (* Fin du corps de l'algorithme *)
```

Le texte placé entre (* et *) est un commentaire selon la syntaxe du langage Pascal. La syntaxe et la structure des trois parties de l'algorithme sont plus amplement décrite dans ce qui suit :

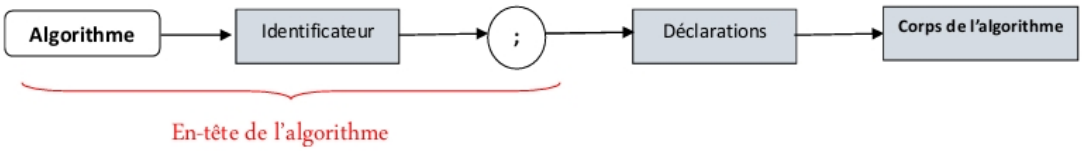


FIGURE 3.5 – Structure simplifiée d'un algorithme.

- **L'identificateur :** c'est un nom composé de caractères alphanumériques dont le premier doit être obligatoirement alphabètique. Le caractère blanc (espace), les mots clés du LDA,

les caractères accentués ainsi que les caractères de ponctuation à l'exception du cractère '_' ne sont pas autorisés comme illustré dans la Figure3.6.

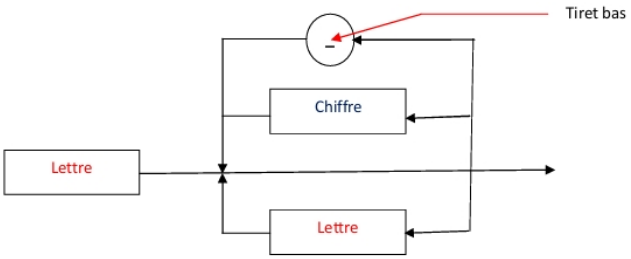


FIGURE 3.6 – Règle d'écriture d'un identificateur.

☞ Exemple :

```
a1, val_1, chaine, caract12, prix_total     Identificateurs corrects
1a, a b, x?1, Fin, z+2, prix total , couleur-ciel  Identificateurs incorrects
```

- **Partie déclarative :** cette partie contient les déclarations des constantes, variables, nouveaux types ainsi que les sous-programmes en cas de programmation modulaire comme illustré dans la Figure 3.7.

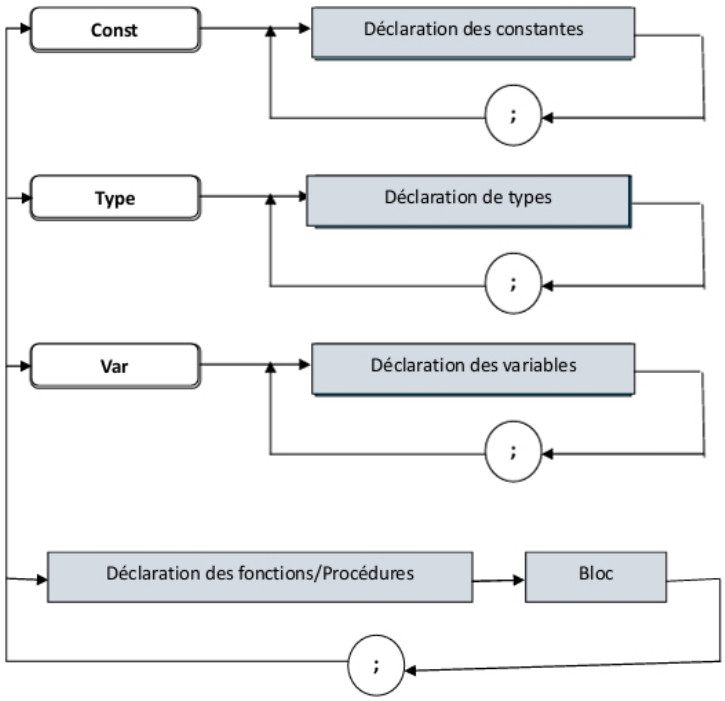


FIGURE 3.7 – Partie déclarative d'un algorithme.

- **Déclaration des constantes :** une constante est déclarée dans la section **Const** par un identificateur et une valeur. La valeur est fixe et ne peut être modifiée dans le corps de l'algorithme. Le type de la constante est implicite, si la valeur est un nombre contenant un point alors la constante est un réel autrement c'est un entier et la présence des quotes indique que c'est un caractère ou une chaine de caractères dans le cas d'une succession de caractères.

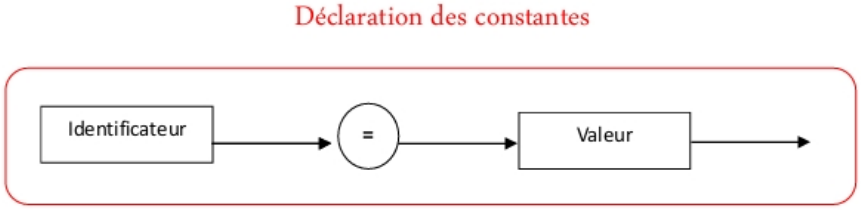


FIGURE 3.8 – Déclaration des constantes.

☞ Exemple :

```
Const Pi = 3.1415;
      k = 20;
      epsilon = 0.0001 ;
      existe = vrai ;
      rep = 'yes';
      blanc = ' ';
```

- **Déclaration des variables :** toute variable de l'algorithme (valeur modifiable) doit être déclarée dans la section **Var** par un identificateur et un type. Des variables de même type sont séparées par des virgules. Les types standards ou prédéfinis du langage sont le type entier, réel, caractère et logique (booléen).

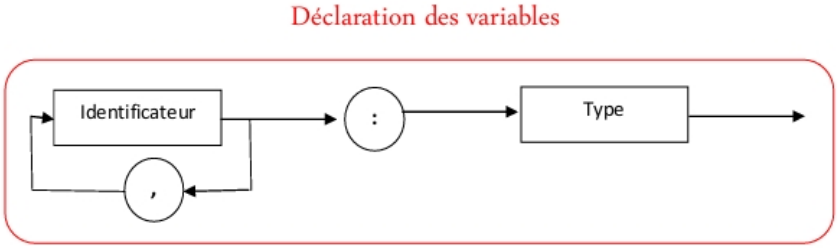


FIGURE 3.9 – Déclaration des variables.

☞ Exemple :

```
Var
    i,j,k : entier;
   x,y,z : réel;
   c : caractère;
   trouve : booléen;
```

- **Déclaration des types nouveaux dits personnalisés :** les types autres que les types standards doivent être définis dans la section **Type** comme le type tableau, énumération, intervalle, structure (enregistrement ou record en Pascal), union,... etc. Ces types seront présentés dans le chapitre.

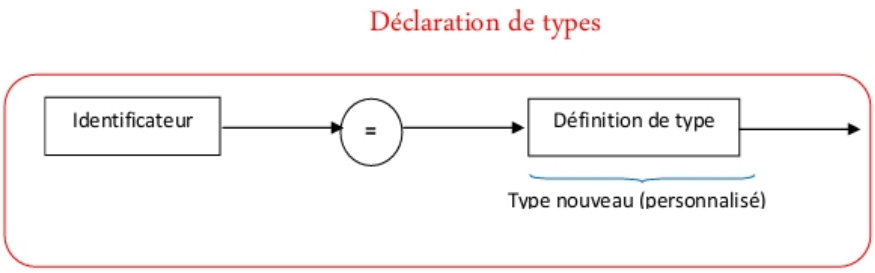


FIGURE 3.10 – Déclaration des types nouveaux.

- **Partie corps de l'algorithme :** cette partie contient les instructions exécutables. On distingue les instructions séquentielles (instructions d'Entrée/Sortie et l'affectation) et les structures de contrôle qui regroupent les instructions conditionnelles (l'alternative simple, l'alternative composée et le choix multiple) et les intructions itératives (boucles) comme illustré dans la Figure3.12.

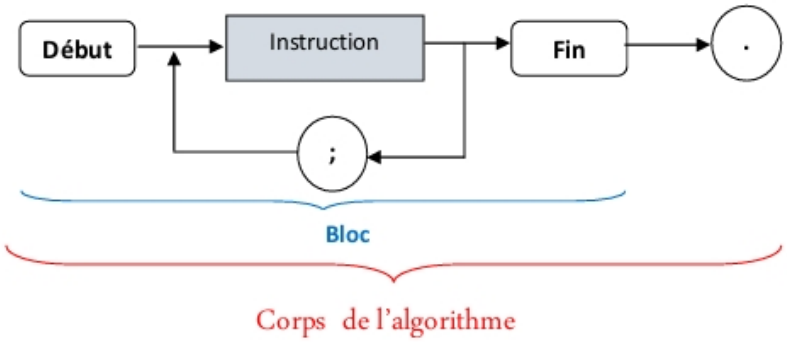


Figure 3.11 – Partie corps de l'algorithme ou instructions exécutables.

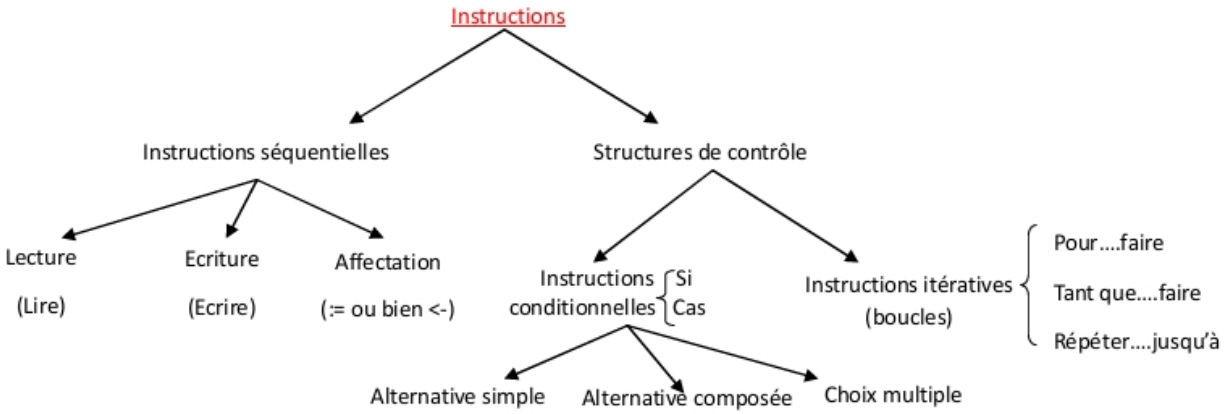


Figure 3.12 – Type d'instructions.

☞ Exemple1 : Soit à calculer la somme de deux nombres entiers a et b.
Entrée : deux nombres a et b de type entier.
Sortie : la valeur résultat s de type entier.

```
Algorithme Somme;
Var a,b,s : entier ;
Début
Ecrire('Entrer a et b : ') ;
Lire(a,b) ;
s <- a+b;
Ecrire('La somme = ', s) ;
Fin.
```

☞ Exemple2 : Soit à résoudre l'équation du premier degré ax+b avec a et b des nomres réels.
Entrée : les coefficients a et b de type réel.
Sortie : la solution x de l'équation de type réel si elle existe.

```
Algorithme Somme;
Var a,b : réel ;
Début
Ecrire('Entrer a et b : ') ;
Lire(a,b) ;
Si(a=0) alors
  Ecrire('Division par zéro');
  Sinon
  Ecrire('x=', -b/a);(*Une instruction d'écriture peut contenir une expression de calcul *)
Fin.
```

L'algorithme peut être re-écrit autrement en utilisant la notion de **bloc d'instructions** là où la syntaxe éxige une seule instruction, cas des deux langage Pascal et le C.

```
Algorithme Eq_1d;
Var a,b : réel ;
Début
Ecrire('Entrer a et b : ') ;
Lire(a,b) ;
Si(a=0) alors (* Une seule instruction autorisée après alors *)
        Ecrire('Division par zéro');
        Sinon (* Une seule instruction autorisée après sinon *)
            Début       (* Début du bloc *)
         x <- -b/a;
            Ecrire('x=', x);
            Fin;        (* Fin du bloc *)
Fin.
```

La séquence d'instructions à exécuter après **Sinon** est englobée dans un bloc pour n'en faire qu'une seule instruction (macro-instruction) qui sera alors exécutée dans le cas sinon. Dans le cas où le bloc Début-Fin est omis, ce sera uniquement l'instruction $x \leftarrow -b/a$ qui sera exécutée dans le cas sinon. L'instruction Ecrire('x=',x) sera exécutée en séquence après l'instruction Si et l'algorithme devient identique à l'algorithme ci-dessous.

```
Algorithme Eq_1d;
Var a,b : réel ;
Début
Ecrire('Entrer a et b : ') ;
Lire(a,b) ;
Si(a=0) alors (* Une seule instruction autorisée après alors *)
        Ecrire('Division par zéro');
        Sinon (* Une seule instruction autorisée après sinon *)
            x <- -b/a;
Ecrire('x=', x);
Fin.
```

Cet algorithme n'est pas correct puisque dans le cas où la valeur de a est nulle, la valeur de x à afficher sera indéterminée, ce qui représente un cas d'erreur sémantique (logique).

☞ Exemple3 : soit à calculer la surface d'un cercle de rayon r donné.

Entrée : le rayon r de type réel.

Sortie : la surface surf de type réel.

Ici, le calcul de la surface d'un cercle requiert la valeur de π qui sera déclarée comme constante.

```
Algorithme Aire_Cercle;
Const Pi = 3.1415;
Var r, surf : réel ;
Début
Ecrire('Entrer le rayon : ') ;
Lire(r) ;
surf <- Pi*r*r;
Ecrire('Surface du cercle =', surf);
Fin.
```

Bien entendu, le calcul pourra se faire dans l'instruction de sortie et l'algorithme peut être re-écrit comme suit :

```
Algorithme Aire_Cercle;
Const Pi = 3.1415;
Var r : réel ;
Début
Ecrire('Entrer le rayon : ') ;
Lire(r) ;
Ecrire('Surface du cercle =', Pi*r*r);
Fin.
```

Exercice 1 *Considérons la conversation suivante qui a lieu à une date d :*
"Quel age avez vous Monsieur ?" "J'ai deux fois votre age Monsieur et l'année ou le mur de Berlin a été, j'avais votre age." On se demande donc quel age ont les deux bonhommes.
Entrée : la date d quand a lieu la conversation et la date de la destruction du mur de Berlin (1990) qui est une constante.
Sortie : l'age x du vieux et l'age y du jeune ($x > y$).
La date d est dite instance du problème, le résultat en dépend ainsi que la date de destruction du mur de Berlin qui est une constante.
Deux equations fournissent donc le résultat cherché qui sont les suivantes :

$$\begin{cases} y = d - 1990 \\ x = 2 \cdot y \end{cases} \tag{3.1}$$

```
 Algorithme Dialogue;
Const Date=1990;
Var x,y : réel;
Début
Ecrire('Entrer la date de la conversation : ');
Lire(d) ;
y <- (d-Date) ;
x <- 2*y
Ecrire('Age du vieux bonhomme= ',x,'Age du jeune=',y)  ;
```

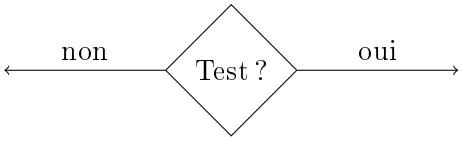
```
Fin.
```

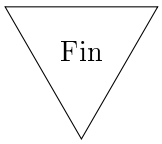
Soit donc si la conversation a eu lieu en 2019, le jeune homme avait 29 ans et le vieux 58 ans.

Graphique ou organigramme : un diagramme ou schéma imagé représentant les étapes à effectuer où chaque figure décrit l'opération réalisée, il n'existe pas une norme pour les figures mais les plus communes sont celles ci :

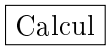
- Début de l'algorithme : 

- Opération d'Entrée/Sortie (E/S) : 

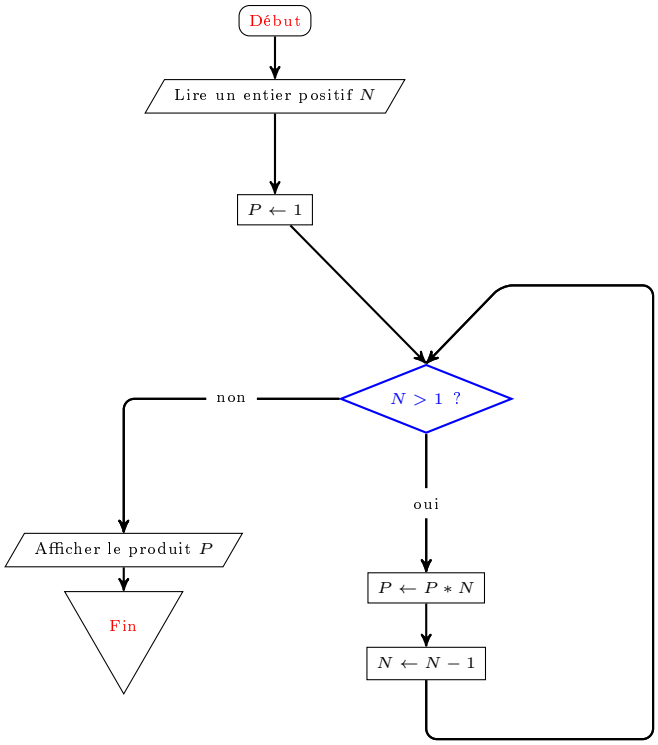
- Test ou condition (Si) : 

- Fin de l'algorithme, il peut y avoir plusieurs : 

ou bien 

- Toute autre opération (calcul) : 

☞ Exemple : Calcul du produit des N premier nombres entiers, N nombre entier lu en entrée.



Cette représentation est utilisée de moins en moins aujourd'hui, cela est probablement du au fait qu'elle est souvent difficile à maîtriser pour des problèmes complexes.

Exercice 2 *On veut calculer la moyenne annuelle d'un élève à partir des 3 moyennes semestrielles.*
a/ Préciser les données et les résultats qui constituent les objets du problème.

b/ Donner l'organigramme permettant de calculer et d'afficher cette moyenne et si l'élève est admis (sa moyenne est supérieure ou égale à 10) ou ajourné.

Solution de l'exercice :

a/ Les objets du problème :

Entrée : moy1, moy2 et moy3 de type réel.

Sortie : moy (moyenne générale) de type réel.

b/Organigramme permettant de calculer et afficher la moyenne :

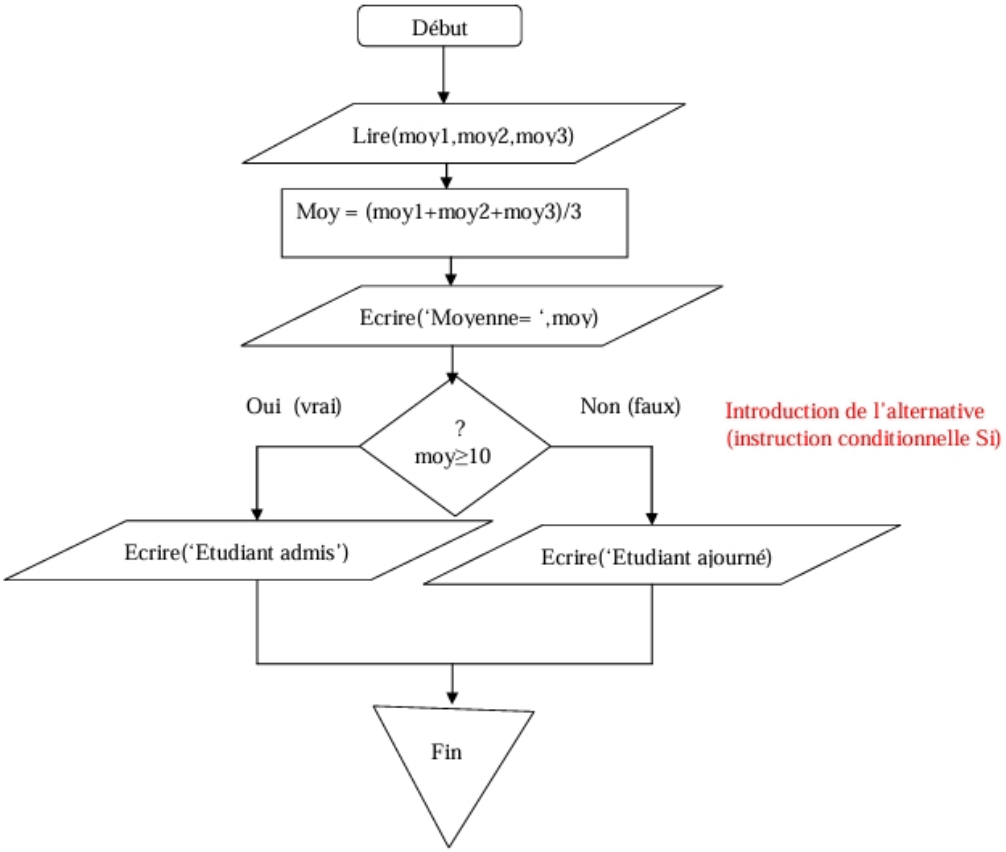


Figure 3.13 – Organigramme calculant la moyenne de trois nombres.

Exercice 3 *On veut calculer les solutions d'une équation du premier degré ax+b=0.*

1.Préciser les données et le résultat.

2.Donner l'organigramme correspondant.

Solution de l'exercice :

1. Les données et le résultat cherché :

Entrée : a et b de type réel.

Sortie : la solution x (si elle existe) de type réel.

2. L'organigramme correspondant :

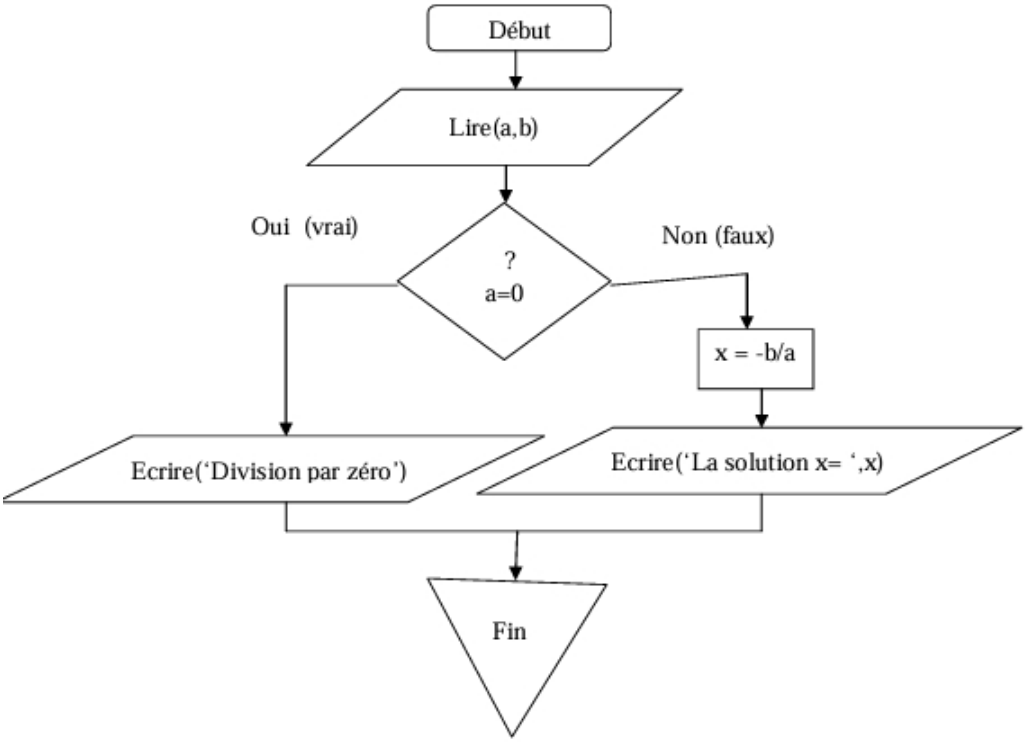


Figure 3.14 – Organigramme résolvant l'equation du 1^{er} degré.

Trois problèmes fondamentaux caractérisent un algorithme :

- La correction : un algorithme est correct, si pour toute instance du problème, il se termine et produit une sortie correcte.
- La complétude : l'algorithme produit un résultat pour tous les cas possibles.
- : L'éfficacité : l'algorithme doit être économique du point de vu consomation des ressources machine qui se résument en général en terme de temps (complexité temporelle) et espace mémoire (complexité spatiale).

3.2 Notion de programme

Un programme est la traduction de l'algorithme dans un langage formel que la machine peut comprendre et exécuter. Ce langage est dit langage de programmation et il est régi par une syntaxe tout autant que l'est le langage naturel de l'homme. Les instructions éxécutables et les déclaration des données doivent donc obéir à cette syntaxe. Une fois écrit, un algorithme est alors transformé en un programme source écrit dans un langage de programmation qui sera traduit et exécuté par un ordinateur comme le montre la Figure 3.15. Le langage de programmation quoi qu'il puisse parfois necessiter une maîtrise de l'architecture de la machine comme le langage d'assemblage ou assembleur (langage dit de bas niveau), il est assez éloigné du binaire et permet un programmation aisée et rapide. Le langage de programmation est par contre dit de haut niveau s'il fait complêtement abstration de l'architécture

de la machine et ne requiert pas de connaissances a priori du système de fonctionnement de la machine comme Pascal, Fortran, Java. Le langage C/C++ est quant à lui considéré comme langage de bas niveau mais pas autant que l'est l'assembleur bien heureusement.

Le premier véritable langage de programmation fut le FORTRAN (FORmula TRANslation) dédié au calcul scientifique, mis au point sur l'IBM 701, il nécessitait un programme intermédiaire approprié, qui devait traduire tout programme dans le langage codé de la machine. D'autres langages de programmation ont suivi comme le COBOL (COmmon Business-Oriented Language), l'ALGOL (ALGOrithmic Language) qui étaittrès théorique, LISP, PROLOG dédiés à l'intelligence artificielle ainsi que PASCAL. Ces langages qui ont apporté quelques progrès en matière de programmation ont contribué à la conception de nouveau langages tels que le BASIC (Beginner's All-purpose Symbolic Instruction Code) pour les débutants et le Pascal pour faciliter l'enseignement de l'informatique dans les écoles. Aujourd'hui, il existe une panoplie de langages à usages divers.

Le traducteur est un programme qui prend le programme en code source comme donnée d'entrée et produit en sortie le code objet en langage machine, le seul que comprend la machine est le binaire. Le traducteur peut être soit un compilateur ou interpréteur.

- Le compilateur : traduit le programme source en entier c'est à dire la totalité des instructions, il faut qu'il soit complet pour être compilé (Pascal, C/C++, Fortran). Le code cible ou objet produit, peut être conservé dans un fichier (exécutable) et doit passer par la phase d'édition des liens pour être exécuté. La phase d'édition de liens consiste à relier entre eux les fichiers objets qui constituent le programme, puisque généralement le programme est séparé en plusieurs fichiers source et peut faire appel à des librairies de fonctions standard prédéfinies. L'édition de liens produit alors un fichier dit exécutable.
- L'interpréteur : traduit le programme source par instruction et l'exécute directement, pas besoin que le programme soit complet et dans ce cas aucun programme objet n'est généré (Basic, Lisp, Ruby, Python). Il permet de tester un programme en cours d'écriture mais il est naturellement plus long qu'un compilateur puisqu'il est interactif.

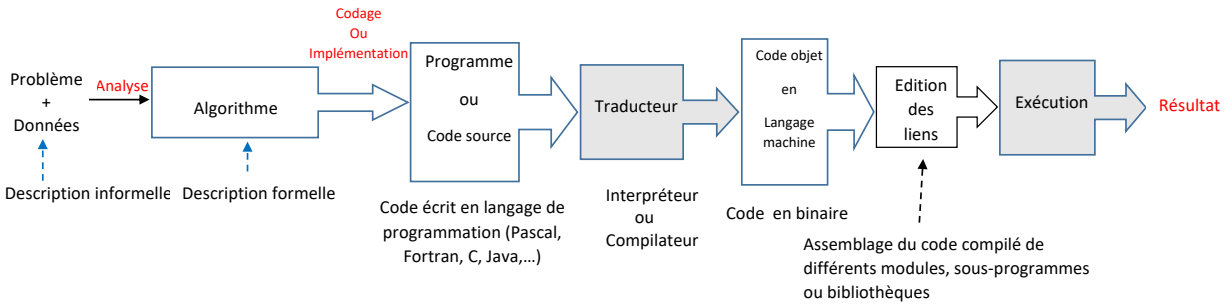


Figure 3.15 – Résolution d'un problème par un ordinateur.

Chacun de ces deux types de traducteurs a ses avantages et ses inconvénients et certains langages de programmation ont essayé d'en tirer profit en combinant les deux comme Python et Java. Les programmes écrits dans ces langages sont d'abord compilés et un code intermédiaire est généré, similaire à un langage machine appelé **bytecode** qui est ensuite interprété et exécuté directement. Il existe

actuellement des centaines de langages de programmation, certains sont plus populaires que d'autres et ceux qui sont toujours en haut de l'échelle sont C/C++ et Java quoi que certains nouveaux langages essayent de les détrôner comme R et Python.

3.2.1 C comme langage de programmation

Le Langage C qui est un langage compilé a été conçu en 1972 par Dennis Richie et Ken Thompson au Bell Labs de la compagnie Bell Telephone aux USA utilisé dés son apparition pour l'écriture du système d'exploitation UNIX qui était alors écrit en assembleur aussi bien que les autres systèmes d'exploitation. En 1978, Brian Kernighan et Dennis Richie publient la définition classique du C dans le livre "The C Programming language (Standard KR-C)" (Kernighan and Richie, 1988). Il est devenu populaire dans les années 80 et en 1983, L'ANSI (American National Standard Institute) décida de normaliser le C et en 1989 il y'eut le ANSI-C respectant la norme ISO (International Organization for Standardization) C89 (et ensuite C90), puis le langage C++ a vu le jour aux Bell Labs grâce à Bjarne Stroustrup comme une extension du ANSI-C avec à la fois tous les avantages de ANSI-C (puissance, modularité et portabilité) et la programmation orientée objet. En 1999, il y' eut la norme ISO C99 pour combler certaine lacunes du C et minimiser les incompatibilités entre le C et le C++.

Le compilateur C a la particularité de procéder en deux phases, dans la première phase il met en œuvre le préprocesseur qui effectue un pré-traitement du code source consistant en des transformations purement textuelles à partir de directives comme remplacer ou substituer les chaînes de caractères et les macros, inclure les différents sources comme les fichiers en-tête des librairies ou bibliothèques standard ou autres. En fait c'est une manière de re-écrire le code source en incorporant tous les sources dont il a besoin et dans la deuxième phase c'est la compilation proprement dite qui traduit le texte généré par le préprocesseur en langage machine dit code objet.

Mots-clés du langage : le langage C dispose d'un certain nombre de mots clés ou mots réservés, la liste est la suivante :

auto double int struct break else long switch case enum register typedef char extern return union const float short unsigned continue for signed void default goto sizeof volatile do if static while

Directives et macro en C : Les directives du préprocesseur ont un nom minuscule et commencent toutes par le symbole dièse « # ». Elles sont déclarées sur une ligne dédiée et doivent impérativement apparaître en début de ligne (une seule par ligne). Les noms standards de directives sont :

- define : définit les constantes et macros pour substitution par le préprocesseur. Toute occurence de l'identificateur indiqué dans la directive sera remplacée par la valeur précisée, la commande a la forme suivante :

define identificateur valeur

☞ Exemple :

```
# define PI 3.1415
# define Taille_Max 100
# define epsilon 0.1E-12
```

L'dentificateur d'une constante est parfois écrit en majuscule pour différencier entre une variable et une constante (voir subsection 3.3). L'identificateur d'une macro est par convention écrit en majuscule et peut éventuellement être spécifié avec des paramètres dans le cas d'une macro avec paramètres et la valeur peut être dans ce cas une expression à évaluer.

- include : permet d'inclure le texte des fichiers sources ayant l'extension « h » (header) comme les fichiers en-tête des librairies prédéfinies (<math.h>, <stdio.h>, <stdlib.h>, <string.h>,$\cdots$,etc) ou n'importe quels autres fichiers (d'extension ".c") qui composent le fichier source. Elle apparaît donc sous deux formes :

include <identificateur.extension>

Ou bien :

include "identificateur.extension"

La première forme est spécifique aux fichiers en-tête des librairies standard et la seconde est spécifiée pour les fichiers définis par le programmeur. Dans la première forme, le fichier en question est recherché par défaut dans le répertoire du système « /usr/include ». Dans la seconde forme, il est recherché dans le répertoire courant. Il est possible de spécifier d'autres répertoires de recherche en utilisant l'option »-I « de la commande de compilation **gcc**. ☞ Exemple :

```
# include <stdio.h> // Librairie standard associée aux opérations d'entrée/sortie
# include <string.h> //Librairie des opérations dédiées aux chaînes de caractères
# include "fonc.h" // Le fichier fonc.h doit être incorporé pour compléter ce code
```

- ifdef, ifndef, if, else, elif, endif : sont utilisées pour la compilation conditionnelle pour contôler ce qui sera compilé ou ce qui ne le sera pas selon une condition qui peut être soit une constante numérique désignée par un identificateur soit une expression à évaluer par le préprocesseur. Les deux formes selon que la condition est une expression ou une constante numérique sont :

#**if** $expression_1$
    $code_1$
#**elif** $expression_2$
    $code_2$
.....
#**elif** $expression_n$
    $code_n$
#**else**
    $code_0$
#**endif**

En général si une expression est non nulle, le morceau de code qui est entre #**if** et #**endif** est compilé puisque la clause #**if** sera ignorée et sera par contre ignoré lors de la compilation si l'expression est nulle. La clause **if** peut avoir une clause **else** et il peut y avoir des clauses **if** imbriquées, la séquence #**else...**#**if** est abrégée par #**elif**.

#**ifdef** identificateur
    $code_1$
#**else**
    $code_2$
#**endif**

Si la constante identifiée par identificateur est définie avant la rencontre de la directive #**ifdef**, alors le morceau de code code1 sera compilé et code2 sera ignoré. Autrement, c'est le morceau de code code2 qui sera compilé. La directive #**else** est facultative. Il est aussi possible d'utiliser le test sur la non définition d'une constante #**ifndef** (if not defined) comme suit :

#**ifndef** $identificateur_1$
    code1
#**else** $identificateur_2$
    code2
#**endif**

- undef : permet de retirer ou annuler la définition (substitution) d'une macro, provoque l'effet contraire de la directive « **define** ». Aprés cette directive, la macro n'existera plus et donc ne pourra pas être utilisée dans le programme.
  ☞ Exemple d'utilisation :

```
#define TailleMax 100
#undef TailleMax // annule la définition de la constante TailleMax
```

- pragma : permet de contrôler le compilateur en spécifiant des ordres que le compilateur doit respecter tout en conservant la portabilité du programme. Le format des ordres de la directive pragma n'est pas normalisé, il est spécifique à chaque compilateur. L'ordre spécifié peut être activé (ON), désactivé (OFF) ou remis à son état par défaut (DEFAULT).
- error : permet de stopper la compilation et émettre un message d'erreur personnalisé. Elle est utilisée comme suit :

#line error message

Cette directive doit apparaître un dans bloc if ou ifdef pour qu'elle fasse effet.

- line : permet de modifier le numéro de la ligne courante. Elle indique que la ligne suivante dans le fichier source sera la ligne indiquée par le numéro spécifié, elle est utilisée comme suit :

  #line numéro [fichier]

  Si la directive est suivie d'un nom de fichier entre guillemets (" "), celle ci indique que la ligne indiquée est dans le fichier source spécifié par le nom de fichier, exemple :

```
# line 42 "monfichier.c"
```

  Il est possible de changer les valeurs **__FILE__** et **__LINE__** en utilisant cette directive.

Une macro est une constante qui peut prendre un certain nombre d'arguments. La Table 3.2.1 montre les macros sans paramètres prédéfinies en C.

Macro	Signification
__LINE__	numéro de la ligne courante
__func__	nom de la fonction courante
__FILE__	nom du fichier en cours de traitement
__DATE__	date du jour sous la forme "Mmm jj aaaa"
__TIME__	heure sous le forme "hh :mm :ss"
__STDC__	vaut 1 si le compilateur est conforme à la norme ANSI
__STDC_VERSION__	vaut 199901L (type long) selon la norme C99

Table 3.4 – Les macros prédéfinies du langage C.

☞ Exemple d'utilisation :

```
#include <stdio.h>
int main ()
{ printf ("C'est la ligne %d du fichier %s\n", __LINE__, __FILE__);
}
```

Dans les macros avec paramètres, les paramètres sont placés entre parenthèses après le nom de la macro tout comme une fonction ordinaire, par exemple :

#define identificateur(paramètres) expression

Où les paramètres sont séparés par des « , » et expression est en fonction de la liste de paramètres spécifiés. Par exemple, avec les directives suivantes :

#**define** SIGNE(x) (x >= 0 ? 1 : -1)

le préprocesseur remplacera dans la suite du code toutes les occurrences de la forme **SIGNE(a)** où a est une valeur quelconque par **(a >= 0 ? 1 : -1)**. et pour :

#**define** DIFF_POS(x,y) ($x > y$? x-y : y-x)

le préprocesseur remplacera dans la suite du code toutes les occurrences de la forme **DIFF_POS(a,b)** par **(a > b ? a-b : b-a)**.

Les commandes correspondantes aux directives au préprocesseur ne doivent pas être terminées par le caractère « ; », c'est une convention du langage C pour faire la distinction entre une instruction du programme et une directive au préprocesseur.

Structure d'un programme C : tout programme C est constitué d'une en-tête, une partie déclarative où sont déclarés tous les objets dans le programme et une partie dite corps du programme comportant les instructions exécutables. Comme il a été mentionné auparavant, un programme source peut être réparti dans plusieurs fichiers avec l'extension « c » et l'extension « h », il a alors en général l'apparence suivante :

```
[Directives au préprocesseur]
[Déclarations des variables externes]
[Sous-programmes]
[int] main([char** argv[], int argc])
{
[Déclarations des variables internes]
instructions;
[return(statut);]
}
```

Le programme principal **main()** est appelé fonction principale et est unique, elle représente le point de départ de l'exécution du programme. Cette fonction peut éventuellement appeler une ou plusieurs fonctions secondaires qui à leur tour peuvent appeler d'autres fonctions secondaires. L'en-tête de la fonction **main()** se présente généralement sous les formes suivantes :

```
int main()
int main(void)
void main(char** argv, int argc)
int main(char** argv, int argc)
```

La première forme qui est la plus courante, est la plus simple. Dans la seconde forme la fonction **main()** retourne une valeur entière précisant l'état de déroulement de l'exécution, une valeur positive non nulle indique des anomalies et une valeur nulle, indique une exécution correcte. il est

possible d'utiliser comme valeur de retour les deux constantes symboliques **EXIT_SUCCESS** (valant 0) et **EXIT_FAILURE** (valant 1) fournies par la librairie <**stdlib.h**>. L'instruction **return**(statut) dans la fonction main, où statut est un entier spécifiant l'état de terminaison du programme, peut être remplacée par un appel à la fonction **exit()** de la librairie standard <**stdlib.h**>. La fonction **exit()** a pour prototype :

```
void exit(int statut) ;
```

provoque une terminaison normale du programme en notifiant un succès ou un échec selon la valeur de l'entier statut.
Les paramètres « argc » et « argv » s'ils sont spécifiés permettent de récupérer les arguments de la ligne de commande qui a exécuté le programme. La variable « argc » représente le nombre d'arguments, y compris le nom du programme. La variable « argv » est un tableau de chaînes de caractères contenant la liste des arguments. Les deux dernières formes permettent de récupérer des valeurs sur la ligne de commande au moment de l'exécution du programme, comme par exemple la ligne de commande suivante :

```
> gcc pgcd.c -o pgcd
> ./pgcd 12 26
```

Dans cet exemple le programme pgcd a deux données sur la ligne de commande, les chaînes de caractères 12 et 26. Le tableau argv va contenir dans l'ordre la chaîne pgcd puis 12 et 26 qui occuperont les cases d'indice 0 jusqu'à $argc-1$ soit ici argv[0], argv[1] et argv[2] respectivement.

Les commentaires : améliorent la lisibilité du code et facilitent sa compréhension et sa maintenance. Considérés comme du texte ignoré par le compilateur, ils sont repérés par deux slashes « // » si le texte du commentaire tient sur une seule ligne ou sont délimités par les balises « /* » et « */ » si le texte comprend plusieurs lignes. Les commentaires ne doivent pas être imbriqués.

Les directives du préprocesseur : sont repérées par le caractère « # » en début de ligne comme par exemple :

```
# include<stdio.h>
# define PI 3.1415
```

La directive « include » permet d'inclure les librairies et les fichiers en-tête (header), le préprocesseur se charge d'inclure les codes prédéfinis pour en tenir compte à la compilation. La directive « define » sert à déclarer les constantes, le préprocesseur peut se charger de substituer toutes les occurrences des constantes par leurs valeurs numériques avant la phase de compilation puisque l'utilisation des constante ne nécessite pas de réservation en mémoire.

Instructions et bloc d'instructions : chaque instruction est suivie d'un point-virgule « ; ». Plusieurs instructions peuvent être regroupées en un seul bloc délimité par une accolade ouvrante « { » et une accolade fermante « } ». Le bloc est souvent dit macro-instruction ou instruction composée, il est utilisé là où la syntaxe exige une seule instruction à la fois plutôt que plusieurs.

Compilation et exécution d'un programme C pour compiler un programme C, la commande gcc (GNU Compiler Compiler) est utilisée avec d'éventuelles commandes comme suit :

```
>gcc [option] fichiers
```

Où les options de compilation signifient :

-ansi : forcer l'utilisation de la norme ANSI-C.
-l librairie : pour inclure une bibliothèque dont le nom est spécifié.
-E : lancement de la phase de pré-compilation uniquement c'est à dire déclencher juste le préprocesseur, le fichier résultat a l'extension « i ».
-g : compilation en mode debug.
-S : exécution jusqu'à l'étape de compilation.
-c : pour avoir le fichier objet avant l'édition de liens ayant l'extension « o ».
-O niveau : active l'optimiseur du code, le niveau d'optimisation peut être 1,2 ou 3.
-I chemin : indique le chemin du répertoire où se trouve les fichiers en-tête (header).
-std=standard : indique le nom du standard, C89 (C de base) ou C99.
-v : mode verbose affiche tout ce qui se fait.
-o file : permet de renommer le fichier résultat.
-Wall : afficher tous les avertissements (warnings).
-Werror : les warnings sont considérés comme des erreurs.
fichiers : liste des fichiers à compiler.

En sortie cela crée un fichier a.out sous Linux ou un fichier.exe sous Windows. L'exécution du fichier objet généré se fait par la commande :

```
>./a.out [données]
```

Ou bien :

```
>gcc exemple.c -o exemple
>./exemple [données]
```

Les données si elles sont spécifiées peuvent être des valeurs ou des nom de fichiers. La première forme utilise le nom du fichier de sortie par défaut généré par gcc, la seconde forme utilise le nom de fichier nommé par l'option « -o ».

☞ Exemple de programme en langage C :

```c
# include <stdio.h>
# include <math.h>
#define PI 3.14159
int main()
{
float rayon;
```

```
printf("Quel est le rayon du cercle ? ");
scanf("%f", &rayon);
printf("Sa circonference est %f.", 2*PI*rayon);
printf("\nSon aire est %f.\n", PI*pow(rayon,2));
}
```

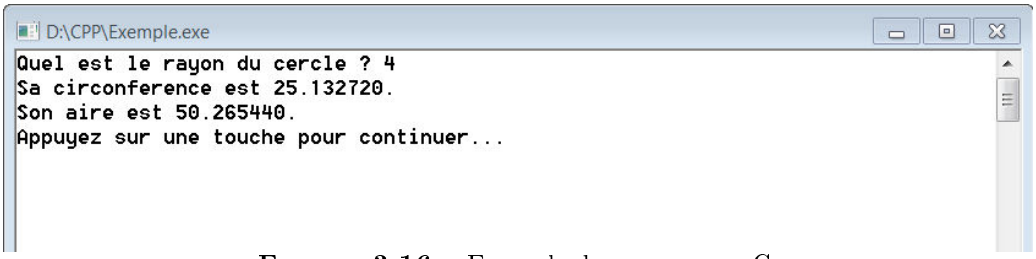


FIGURE 3.16 – Exemple de programme C.

3.3 Les données : variables et constantes

Les données peuvent être des variables ou constantes. Une constante est une valeur fixe qui reste inchangée au cours du déroulement du programme. L'intérêt d'assigner un nom à une constante est de donner une signification à une valeur et une meilleure lisibilité du code. La déclaration d'une constante ne nécessite pas de réservation en mémoire. Une variable est caractérisée par son type, un nom (identificateur) n'excédant pas 31 caractères ayant un emplacement en mémoire et une valeur qui contrairement à une constante peut évoluer au cours du déroulement du programme. Les identificateurs de nom des variables sont alphanumériques avec le premier caractère obligatoirement alphabétique, l'underscore (_) est autorisé mais l'espace et les caractères accentués sont exclus ainsi que les motc-clés du langage. Toute données utilisée dans le programme doit être déclarée avec son type dans la partie déclarative du programme.

☞ Exemple :

```
#define Taille_Max 1000  // Taille_Max constante dont la valeur est 1000
#define PI 3.1415        // PI constante dont la valeur est 3.14156
int i,j,k;      // Les variables i,j et k sont de type entier (int)
float x,y;     // Les variables x et y sont de type réel (float)
```

3.4 Types de données

3.4.1 Type entier

Le type entier regroupe un sous ensemble des entiers naturels et un sous ensemble des entiers relatifs comme le précise la Table 3.5. On reconnaît les nombres relatifs à leur signe, le modificateur « unsigned » permet d'obtenir un entier naturel tandis que « signed » ou son absence désigne un entier relatif. Les opérateurs applicables sur le type entier sont l'addition (+), la soustraction (-), la multiplication (*), la division (/) et le modulo ou reste de la division entière (%), l'affectation et les opérateurs de comparaison.

Type	Nombre d'octets	Intervalle
unsigned short int	2	0 à $2^{16} - 1 = 65535$
short int	2	$-2^{15} = -32768$) à $(2^{15} - 1 = 32767)$
int	2 ou 4	$(-2^{31} = -2147483648)$ à $(2^{31} - 1 = 2147483647)$
unsigned long int	4	0 à $(2^{32} - 1 = 4294967295)$
long int	4	$(-2^{31} = -2147483648)$ à $(2^{31} - 1 = 2147483647)$

Table 3.5 – Représentation du type entier sur machine.

La représentation en mémoire du type int dépend du compilateur C. Le compilateur Dev-C++ par exemple utilise 4 octets. Le mot-clé **sizeof** donne le nomre d'octets nécessaire pour stocker le type en mémoire, sa syntaxe est la suivante :

taille = sizeof(expression) ;

où expression est un type ou une donnée, exemple :

taille = sizeof(unsigned int) ;
taille = sizeof(y) ;

3.4.2 Type flottant (réel)

Le type flottant représente un sous ensemble des nombres réels. On distingue deux types de réels **float** et **double** mais certains compilateurs offrent le type réel long double.Ils sont représentés en virgule flottante (la position de la virgule n'est pas fixe). Ils s'écrivent en C sous la forme : mantisse + exposant (avec un point décimal) et sont représentés sur machine en binaire sous la forme :

<signe><exposant signé><mantisse> équivalent à <+/-><mantisse> $10^{<exposent>}$ où la mantisse est un décimal positif avec un seul chiffre devant la virgule et l'exposent entier relatif. La Table 3.4.2 donne la plage des valeurs réelles représentables sur machine.

Type	Nombre d'octets	Intervalle
float	4	3.4×10^{-38} à 3.4×10^{38}
double	8	1.79×10^{-308} à 1.79×10^{308}
long double	10	3.4×10^{-4932} à 3.4×10^{4932}

Table 3.6 – Représentation du type réel sur machine.

L'opérateur de cast ou transtypage « (type) » permet la conversion explicite de type d'une expression quelconque, exemple :

```
#define PI 3.1415 ;
int a,i=32,j=6 ;
float x,y=0.5
...
y = x + (float)i/j ;
a = (int)PI ;
```

La valeur de a sera 3 puisque la valeur de PI a été convertie en int pour être affectée à a mais la valeur PI vaut 3.1415 et reste inchangée dans le programme.

3.4.3 Type caractère

Le type caractère comporte les 26 lettres de l'alphabet latin en minuscules et majuscules, les chiffres de 0 à 9, les symboles de ponctuation et de comparaison, les caractères spéciaux,...etc. Le mot clef **char** est utilisé pour déclarer les données de ce type. Comme la mémoire centrale de l'ordinateur conserve l'information sous forme numérique (binaire), le type char est représenté comme un type entier codé sur un octet (8 bits ou byte). Ce qui permet de représenter 256 caractères différents et utiliser les valeurs de ce type dans des expressions entières. Ainsi chaque caractère possède un code numérique équivalent dit code ASCII (American Standard Code for Information Interchange). Comme le code ASCII a été mis au point pour les caractère en langue anglaise et à l'origine utilisait 7 bits de codage permettant donc de représenter 128 caractères différents tout au plus, les caractères accentués et les caractères spécifiques à une langue n'y figurant pas. Il a donc été étendu à 8 bits pour pouvoir coder plus de caractères d'où l'appellation code ASCII étendu. En langage C, les caractères imprimables sont écrits entre quotes « ' ' » et les caractères spéciaux utilisent l'antislash (backslash) « \ », comme par exemple :

Caractère	Signification	Caractère	Signification
$'A'$	lettre A	$'?'$	symbole ?
$'\backslash a'$	signal sonore	$'\backslash r'$	retour chariot
$'\backslash n'$	nouvelle ligne	$'\backslash t'$	tabulation horizontale
$'\backslash v'$	tabulation verticale	$'\backslash f'$	saut de page
$'\backslash\backslash'$	antislash	$'\backslash 0'$	fin de chaîne de caractères
$'\backslash"'$	guillemets	$'\backslash''$	apostrophe
$'\backslash?'$	point d'interruption	$'\backslash b'$	espace arrière

Table 3.7 – Quelques exemples de caractères.

3.4.4 Type Booléen ou logique

Á l'origine le type Booléen n'existait pas dans le langage C. Pour calculer et manipuler des données booléennes c'est le type entier qui est utilisé avec la convention suivante : la valeur 0 représente la constante booléenne faux ou false, toute autre valeur sera assimilée à la constante vrai ou true. Les opérateurs ou fonctions de comparaison utilisent cette convention. La définition du type Booléen a été introduite par la norme du langage C ISO 1999 (C99) avec la librairie <**stdbool.h**> disponible dans

le langage C++ permettant ainsi de manipuler les constantes **true** et **false**. Néanmoins, il est possible de définir un pseudo type booléen en utilisant la directive « define » comme suit :

```
#define bool unsigned int
#define true 1
#define false 0
```

3.4.5 Type vide (void/NULL)

Le type vide **void** est utilisé dans la déclaration du type du résultat de retour des sous-programmes c'est à dire les fonctions du C du type procédures (voir chapitre). Le type vide **NULL** correspond à une adresse vide utilisée pour les pointeurs.

Le langage C permet de définir de nouveaux types à partir des types de base au moyen de l'instruction « **typedef** », en écrivant :

typedef type1 type2 ;

Où :
type1 : est le type de base.
type2 : est le nouveau type définit à partir de type1. Ce nouveau type peut ensuite être utilisé dans une déclaration de variable.
Exemple :

typedef float reel ;

Ce qui veut dire que le mot reel indique un type float, on peut ainsi déclarer les deux variables comme suit :

reel x,y ;

3.5 Opérations de base

3.5.1 Opérateurs et expressions

Une expression est une combinaison d'opérateurs et d'opérandes qui sont soit des constantes soit des variables ou des expressions.

Le résultat de l'évaluation d'une expression dépend de l'ordre d'évaluation des opérateurs et des opérandes. La priorité des opérateur précise l'ordre dans lequel les opérations doivent se faire. Dans les cas où deux opérateurs ont la même priorité le calcul s'effectue de gauche à droite. Les parenthèses « () » sont souvent utilisées pour préciser l'ordre des opérations et augmenter la priorité des expressions. Dans

le cas d'expressions avec des parenthèses imbriquées, les parenthèses les plus internes sont évaluées en premier.

Les opérateurs arithmétiques : dans une expression, les opérateurs arithmétiques peuvent être unaires ou binaires. Ils sont évalués en respectant leur ordre de priorité. L'ordre de priorité qui est pratiquement commun à tous les langages de programmation est donné dans la Table 3.5.1.

Opérateur	Priorité
- (moins unaire)	1
* (multiplication), / (division), % (modulo)	2
+ (addition), - (soustraction)	3

Table 3.8 – Les opérateurs arithmétiques du langage C.

La division est entière si et seulement si ses 2 opérandes sont entiers. Les opérateurs unaires opèrent sur un seul opérande, on distingue le moins (-) unaire, qui fournit l'opposé de l'opérande et les opérateurs utilisés en préfixe ou en suffixe pour incrémenter ou décrémenter la valeur de l'opérande, les trois types sont illustrés dans l'exemple ci-dessous :

```
a = -2 ; a = -a ; // le moins (-) unaire
i = 2 ;
i++ ; // équivalente à i = i+1, i vaut 3
i--; //  équivalente à i = i-1, i vaut 2
i = 2 ; j = ++i ;   // j vaut 3 et i vaut 3
i = 2 ; j = i++ ;  // j vaut 2 et i vaut 3
i = 2 ; j = --i ;   // j vaut 1 et i vaut 1
i = 2 ; j = i-- ;  // j vaut 2 et i vaut 1
```

Les opérateurs relationnels : les expressions peuvent contenir des opérateurs de comparaison. Comme le type logique n'est pas prédéfini en C, le résultat de la comparaison est une valeur entière valant 0 (si le résultat est faux) ou 1 (si le résultat est vrai). La Table 3.5.1 montre les différents opérateurs de comparaisons implémentés en C.

Opérateur	Signification
<	inférieur à
>	supérieur à
<=	inférieur ou égal à
>=	supérieur ou égal à
==	égal à
!=	différent de

Table 3.9 – Les opérateurs relationnels du langage C.

Les opérateurs logiques : le langage dispose de trois opérateurs logiques permettant de connecter des expressions logiques, présentés dans la Table 3.5.1 selon leur priorité :

Opérateur	Signification
!	non
&&	et logique
||	ou logique

a	b	!a	a||b	a && b
vrai	faux	faux	vrai	faux
faux	vrai	vrai	vrai	faux
vrai	vrai	faux	vrai	vrai
faux	faux	vrai	faux	faux

Table 3.10 – Les opérateurs logiques du langage C.

En C, il existe aussi des opérateurs logiques qui agissent sur les entiers (short, int ou long), signés ou non au niveau du bit. Ces opérateurs sont reportés dans la Table 3.10. Le résultat d'une expression logique vaut 1 si elle est vraie et 0 sinon et toute valeur différente de 0 est considérée comme vraie et la valeur nulle comme fausse.

Opérateur	Signification
|	ou inclusif
&	et
∧	ou exclusif
~	complément à 1
<<	décalage à gauche
>>	décalage à droite

Table 3.11 – Les opérateurs logiques bit à bit du langage C.

☞ Exemple d'évaluation d'expressions :

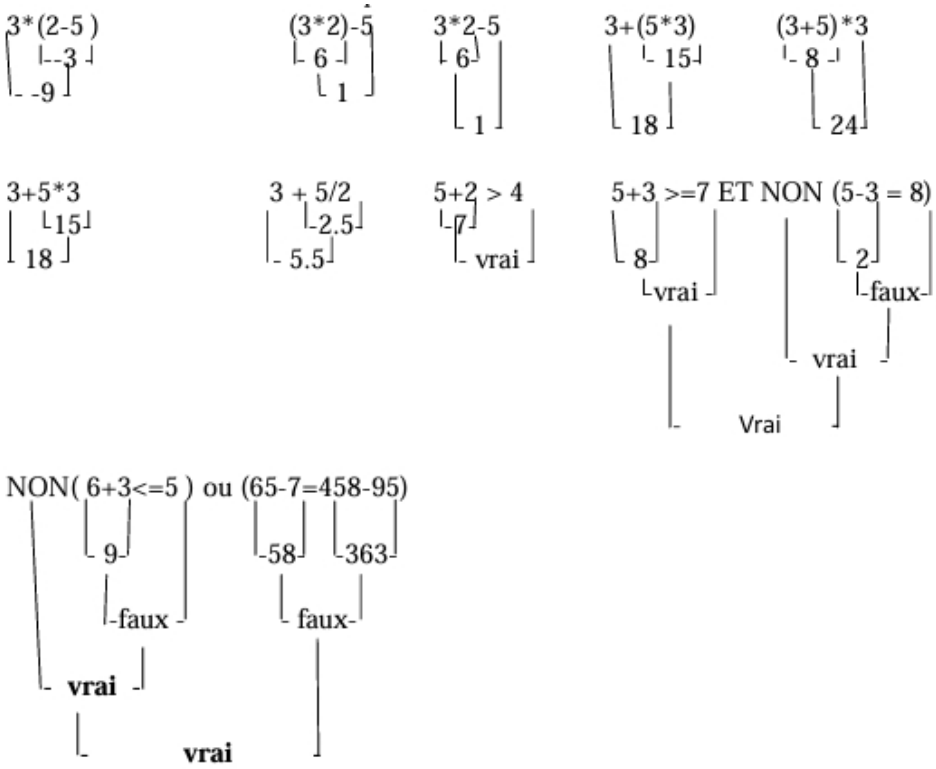


Figure 3.17 – Exemple d'évaluation d'expressions.

3.5.2 Fonctions prédéfinies

Dans une expression les appels aux fonctions prédéfinies sont plus prioritaires. Dans ces appels de fonction les paramètres sont toujours entre parenthèses. La librairie représentée par le fichier en-tête <math.h> déclare des fonctions mathématiques. Tous les paramètres et résultats de type réel sont du type double et les angles sont indiqués en radians. La table 3.5.2 donne quelques fonctions mathématiques.

Fonction	Signification
abs(int) ou fabs(double)	valeur absolue
sqrt(double)	racine carrée
pow(double,double)	puissance $pow(x, a) = x^a$
exp(double)	exponentielle $exp(x) = e^x$
log(double)	logarithme népérien
log10(double)	logarithme décimal
sin(double)	sinus
asin(double)	arc sinus
sinh(double)	sinus hyperbolique
cos(double)	cosenus
acos(double)	arc cosenus
cosh(double)	cosenus hyperbolique
atan(double)	arc tangente
tanh(double)	tangente hyperbolique
floor(double)	le plus grand entier proche de la valeur donnée en paramètre
ceil(double)	le plus petit entier proche de la valeur donnée en paramètre

Table 3.12 – Quelques fonctions mathématiques avec entre parenthèses le type du paramètre de la fonction.

L'instruction :

opérande$_1$= opérande$_1$ *oprateur* *oprande*$_2$

Peut être écrite :

opérande$_1$ *oprateur*= opérande$_2$

Avec les opérateurs +, -, *, /, %, », «, &, ˜, ∧.
Exemple :

```
x+=6 ;
y/=10 ;
i%=3 ;
```

Cette forme d'écriture permet une certaine optimisation du code. Elle est fréquemment utilisée dans la boucle for (cf. Chapitre 5, page 74).

3.6 Instructions de base

3.6.1 Affectation

L'affectation permet de modifier la valeur d'une variable ou initialiser une variable à une valeur. En pseudo-code son symbole est « <- » qui veut dire « reçoit » mais dans tous les langages de programmation son symbole est « = ».
Syntaxe :

variable = variable | expression | constante ;

Où **expression** est une opération à évaluer avec des opérandes et des opérateurs. L'évaluation de expression se fait en deux temps :

- Calculer la valeur de l'expression.
- Ranger la valeur dans la case mémoire identifiée par le nom de la variable.

Ce qui est en partie droite du symbole d'affectation doit être compatible avec le type de la variable dans la partie gauche. Le langage C fait une distinction entre le symbole "=" de l'affectation et l'égalité mathématique « == ». La valeur affectée doit être de même type que la variable. L'initialisation d'une variable peut se faire au moment de sa déclaration.

☞ Exemples :

x=12 ; y=2*x+1 ; p=2*PI*r ; d=(a+b)/c ; rep="Non" ;

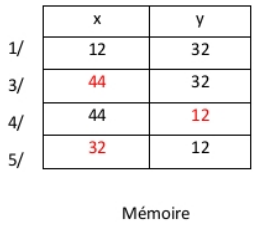
Nous pouvons à ce stade introduire la notion de trace d'un programme qui consiste à exécuter pas à pas les instructions en se focalisant sur ce qui se passe dans la mémoire centrale et sur l'écran d'affichage. Comme l'instruction d'affectation modifie le contenu d'une case mémoire, la trace nous permet d'observer les différents changements du contenu au fur et à mesure de l'exécution du code.

☞ Exemple de trace d'un algorithme : Soit à permuter le contenu de deux variables x=12 et y=32. Faire la trace de l'algorithme : a/ Sans faire intervenir une variable auxilliaire pour la permutation. b/ En utilisant une variable auxilliaire z.
Soit donc l'algorithme pour a/ :

```
Algorithme Permutation1;
Var x,y ;
Début
x <- 12; y <- 32;            (* 1 *)
Ecrire('x=',x,'y=',y);       (* 2 *)
x <- x+y ;                   (* 3 *)
```

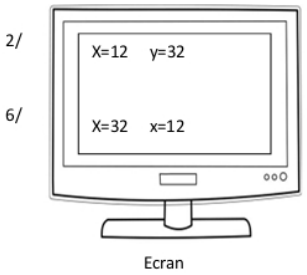
```
y <- x-y ;                    (* 4 *)
x <- x-y ;                    (* 5 *)
Ecrire('x=',x,'y=',y);        (* 6 *)
Fin.
```

La trace pour a/



	x	y
1/	12	32
3/	44	32
4/	44	12
5/	32	12

Mémoire

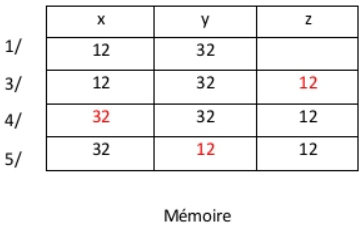


Ecran

Algorithme pour b/ :

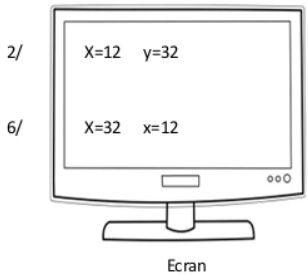
```
Algorithme Permutation2;
Var x,y,z ;
Début
x <- 12; y <- 32;             (* 1 *)
Ecrire('x=',x,'y=',y);        (* 2 *)
z <- x ;                      (* 3 *)
x <- y ;                      (* 4 *)
y <- z ;                      (* 5 *)
Ecrire('x=',x,'y=',y);        (* 6 *)
Fin.
```

La trace pour b/



	x	y	z
1/	12	32	
3/	12	32	12
4/	32	32	12
5/	32	12	12

Mémoire



Ecran

☞ Autres exemples de trace d'un algorithme :

a/Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Var A, B : Entier ;
Début
1/ A ← 5 ;
2/ B ← A*A-A*2 + 4 ;
3/ A← A + B ;
4/ B ← A – B ;
Fin
```

Trace

	Mémoire	
	A	B
1/	5	
2/	5	49
3/	54	49
4/	54	5

b/Que fait cet algorithme ? donner les valeurs finales de A,B et C.

```
Début
1/ A ← 2;
2/ A ← A+2;
3/ B ← A*2+A ;
4/ C ← 4 ;
5/ C ← B - C;
6/ C ← C + A - B ;
7/ A ← B – C * A;
8/ A ← (B – A) * C;
9/ B ← (A + C) * B;
Fin.
```

	Mémoire		
	A	B	C
1/	2		
2/	4		
3/	4	12	
4/	4	12	4
5/	4	12	8
6/	4	12	0
7/	12	12	0
8/	0	12	0
9/	0	0	0

3.6.2 Instructions d'entrée/sortie

Les instructions d'entrée/sortie sont des opérations de lecture/écriture, généralement à partir du clavier et vers l'écran ou autres périphériques spécifiques. Les plus communes en langage C sont les fonctions standard **printf** et **scanf** utilisant des flux standard de caractères qui sont le flux d'entrée **stdin** correspondant au clavier et le flux de sortie **stdout** pour l'écran. Décrivons plus en détail ces instructions :

Instruction d'entrée permet de lire la valeur d'une variable sur le périphérique.

Syntaxe :

Lire($paramtre_1$, $paramtre_2$,..., $paramtre_n$) ;

Où **paramétre** est un **identificateur de variable**. En C : L'instruction de lecture utilise les adresses des variables représentée par l'opérateur « & ». Cet opérateur est omis dans le cas où la variable à lire est une chaîne de caractères.
Il faut inclure la librairie <stdio.h>.

```
#include <stdio.h>
...
scanf("format", &paramtre1, &paramtre2,..., &paramtren) ;
```

La lecture se fait en deux temps :

- Récupérer la valeur du périphérique.
- Ranger cette valeur dans la case mémoire identifiée par le nom de la variable.

Pour lire un seul caractère saisi au clavier, il suffit d'utiliser la fonction **getchar()**, comme suit :

```
char c;
........
c = getchar(); /* Lit un caractère saisi au clavier et le stocke dans la variable c */
```

Ce qui est équivalent à :

```
char c;
........
scanf ("%c",&c) ;
```

Lecture en C++ : l'opérateur **cin** représente le flot d'entrée connectée à l'entrée standard qui est le clavier. Il correspond au fichier prédéfini **stdin** du langage C. #include <iostream> using namespace std ;
...
cin » $paramtre_1$» $paramtre_2$ » ... » $paramtre_n$;

Exemple en C++ :

```
#include <iostream>
using namespace std;
int main ()
{
float variable1, variable2;
cout << "entrez 2 valeurs : ";
cin >> variable1 >> variable2;
}
```

Instruction de sortie permet d'écrire la valeur d'une variable ou d'une expression sur un périphérique de sortie.

Syntaxe :

Ecrire($paramtre_1$, $paramtre_2$,..., $paramtre_n$) ;

En C :

printf("format", $paramtre_1$, $paramtre_2$,..., $paramtre_n$) ;
Où **paramètre** = variable | expression | constante
constante = nombre | message.

Dans le cas d'une expression, l'écriture se fait en deux temps :

- Évaluer l'expression.
- Écrire la valeur du résultat sur le périphérique.

Le format permet de spécifier la représentation de la sortie, il est spécifié avec la syntaxe **%lettre**. La Table 3.6.2 reporte les différents formats utilisés.

Symbole	Type
%d ou %i	entier relatif
%u	entier naturel (unsigned)
%o	entier exprimé en octal
%x	entier exprimé en hexadécimal
%c	caractère
%s	chaîne de caractères
%f ou %e	rationnel en notation décimale ou exponentielle

Table 3.13 – Type de format pour les E/S formatées. Les spécificateurs %ld, %li, %lu, %lo et %lx sont utilisés pour le type long. Les spécificateurs %le ou %lf sont utilisés pour le type double. Pour le type long double, les spécificateurs %Le ou %Lf sont utilisés.

Pour écrire un seul caractère, il suffit d'utiliser la fonction **putchar()**, comme suit :

```
#include <stdio.h>
int main()
{
char c;
........
c=getchar();
putchar(c); /* Affiche la valeur de la variable c qui est un caractère */
........
}
```

Ce qui est équivalent à :

```
#include <stdio.h>
int main()
{
char c;
........
c=getchar();
printf ("%c",c) ;
........
}
```

La bibliothèque standard C++ portant le nom « std » fournit entre autre les primitives d'entrée/sortie « **cout** » et « **cin** ». La directive « using namespace std » rend accessible tout ce qui est implémenté dans cette librairie. Écriture en C++ : l'opérateur **cout** représente le flot de sortie connectée à la sortie standard qui est l'écran. Il correspond au fichier prédéfini **stdout** du langage C. #include <iostream>
using namespace std ;
{
...
cout « $expression_1$ « $expression_2$ « ⋯ « $expression_n$;
}

Exemple en C++ :

```
#include <iostream>
using namespace std;
int main ()
{
float variable1=12, variable2=32;
cout << "Les valeurs sont : ";
cout << variable1 << variable2;
cout << variable1 << variable2 << endl; /* affichage avec retour à la ligne */
cout << variable1 << variable2 << "\n"; /* affichage avec retour à la ligne */
}
```

Le mot clé *endl* signifie que le programme doit aller à une nouvelle ligne après l'affichage.

3.6.3 La séquence

La séquence désigne une suite d'instructions exécutées l'une après l'autre dans l'ordre où elles apparaissent. Par défaut, les instruction d'un programme sont exécutées en séquence. La séquence donc correspond à une suite d'instructions simples qui s'exécutent dans l'ordre où elles apparaissent dans un programme sans rupture de séquence.
Syntaxe de la séquence :

$instruction_1$;
$instruction_2$;
...
$instruction_n$;

Dans ce qui suit, nous noterons par le mot **instructions** toute séquence d'instructions.

3.6.4 Construction d'un algorithme simple

En utilisant les notions d'algorithmique présentées à ce stade, il nous est possible de résoudre un problème simple tel que le calcul du périmètre ou de la surface d'une forme géométrique simple.
☞ Exemple : calcul de la surface d'un rectangle :
Algorithme en pseudo-code :

```
Algorithme Sequence;
Var longueur, largeur, surface : réel;
Début
Ecrire('Entrer la longueur et la largeur du rectangle : ');
Lire(longueur,largeur);
Surface <- longueur * largeur ;
Ecrire('Surface = ',surface);
Fin.
```

Algorithme sous forme d'organigramme :

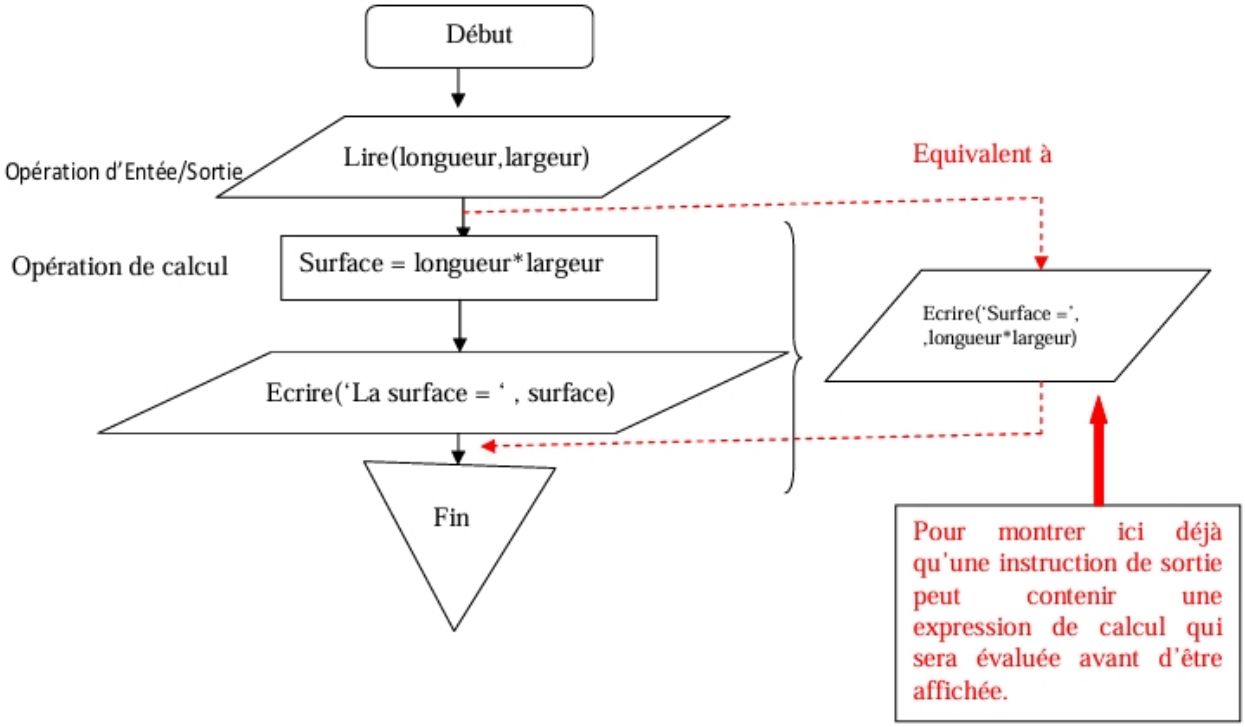


Figure 3.18 – Exemple d'algorithme séquentiel simple.

3.6.5 Représentation d'un algorithme par un programme

La représentation d'un algorithme par un programme qui est l'expression concrète d'un algorithme dans un langage de programmation tel que Python, Java ou C/C++. La conversion d'un algorithme en programme implique trois étapes essentielles :

- Déclarations : Définir les variables et constantes utilisées dans l'algorithme.
- Traitement : Écrire les instructions qui réalisent les opérations décrites par l'algorithme.
- Entrées/Sorties : Gérer la saisie des données par l'utilisateur et afficher les résultats.

3.6.6 Traduction en langage C

Reprenons l'algorithme séquentiel simple de la section 3.6.4 et traduisons le en langage C.

```
#include<stdio.h>
int main() {
float longueur, largeur, surface;
printf("Entrer la longueur et la largeur du rectangle : ");
scanf("%f%f", &longueur, &largeur);
surface = longueur * largeur;
printf("Surface = %f\n", surface);
}
```

Reprenons aussi l'algorithme de calcul de la surface d'un cercle qui fait intervenir la valeur de la constante π.

```
Algorithme Aire_Cercle;
Const Pi = 3.1415;
Var r, surf : réel ;
Début
Ecrire('Entrer le rayon : ') ;
Lire(r) ;
surf <- Pi*r*r;
Ecrire('Surface du cercle =', surf);
Fin.
```

Sa traduction en C sera comme suit :

```
#include <stdio.h>
#define Pi 3.1415
int main() {
floar r,surf ;
printf("Entrer le rayon : ");
scanf("%f", &r);
surf = Pi*r*r;
printf("Surface du cercle = %f\n",surf);
}
```

En intègrant la librairie <math.h> des fonctions mathématiques pour utiliser la fonction pow(x,n) qui fournit la valeur x^n, le programme devient :

```
#include <stdio.h>
#include <math.h>
#define Pi 3.1415
int main() {
floar r,surf ;
printf("Entrer le rayon : ");
```

```
scanf("%f", &r);
surf = Pi*pow(r,2);
printf("Surface du cercle = %f\n",surf);
}
```

CHAPITRE 4

LES STRUCTURES CONDITIONNELLES

Sommaire

Figures

4.1 Introduction

Les instructions conditionnelles permettent de réaliser des tests afin de sélectionner la séquence d'instructions à exécuter suivant le résultat de la condition ou expression. Elles sont de deux types, l'alternative ou le choix simple et l'aiguillage ou choix multiple.

4.2 Structure conditionnelle simple

C'est suivant le résultat d'une condition qu'à lieu la sélection du bloc d'instructions à exécuter. Comme il a été mentionné auparavant, une valeur logique vraie est représentée par un entier positif non nul et par l'entier 0 si elle est fausse. Cette structure existe sous deux formes :
Syntaxe de la première forme :

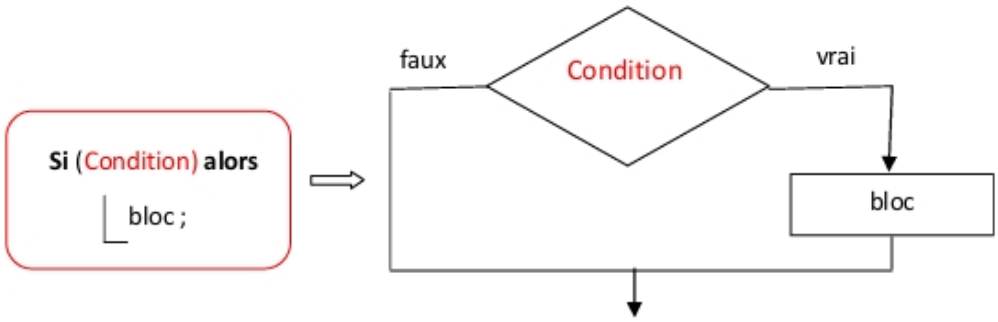


Figure 4.1 – Syntaxe de l'instruction conditionnelle simple (première forme).

En C :

```
if(Condition)
   bloc;
```

Syntaxe de la deuxième forme :

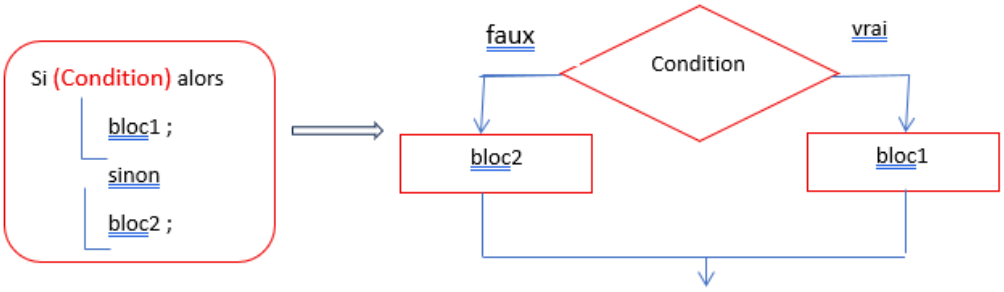


Figure 4.2 – Syntaxe de l'instruction conditionnelle simple (deuxième forme).

En C :

```
if(Condition)
   bloc1;
   else
   bloc2;
```

Si la condition est différente de 0 (vraie), alors ce sont les instructions du bloc1 qui seront exécutées. Si par contre la condition est égale à 0, ce seront les instructions de bloc2 qui seront exécutées (si **else** existe) autrement l'exécution continue en séquence. ☞ Exemple : Calcul d'une différence positive de deux nombres entiers a et b.

En utilisant la première forme :

```
Algorithme comparaison  ;
Var
 a,b,d : entier   ;
Début
Ecrire('Entrer a et b : ');
Lire(a,b);
d <- a-b;
Si(d<0) alors d<- -d;
Ecrire('Différence positive = ',d);
Fin.
```

En C :

```
#include<stdio.h>
int main(){
int a,b,d;
printf("Entrer a et b : ");
scanf(" %d%d", &a,&b)  ;
d=a-b;
if(d<0)
   d = -d;
   printf("Difference positive =%d\n",d);
}
```

En utilisant la deuxième forme :

```
Algorithme comparaison  ;
Var
 a,b,d : entier   ;
Début
Ecrire('Entrer a et b : ');
Lire(a,b);
Si(a  \geq b) alors d<- a-b ;
              sinon d<-b-a;
Ecrire('Différence positive = ',d);
Fin.
```

En C :

```
#include<stdio.h>
int main(){
int a,b,d;
printf("Entrer a et b : ");
scanf(" %d%d", &a,&b)  ;
if(a>=b)
   d = a-b ;
    else
   d = b-a;
  printf("Difference positive =%d\n",d);
}
```

En utilisant la troisième forme : En C :

```
#include<stdio.h>
int main(){
int a,b,d;
printf("Entrer a et b : ");
scanf(" %d%d", &a,&b)  ;
(a>b) ? d=a-b : d=b-a;
  printf("Difference positive =%d\n",d);
}
```

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

```
if(Condition1 && Condition2)
```

```
if(Condition1 || Condition2)
```

Syntaxe de la troisième forme :
En C :

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

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

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

La structure :
if(a>b)
        if(a>c)
                printf("Max= %d",a) ;
Peut être remplacée par :
if(a>b && a>c)
                printf("Max= %d",a) ;

Exercice 4 ***Calcul de remise***

Ecrire un algorithme et le programme C qui cherche le prix à payer pour un article à P DA . Une remise de 30% est accordée si la quantité achetée est supérieure à 100 *unités. P vaut* $50DA$.

```
Algorithme Calcul_Prix ;
Const
R = 0.3 ;
Var
P,Pt,Qte  : réel  ;
Début
Ecrire(Entrer le prix unitaire et la quantite :');
Lire(P,Qte);
Pt = P * Qte;
If(Qte>=100) alors
     Pt = Pt - Pt*R;
Ecrire('Prix total = ', Pt)  ;
Fin.
```

Programme correspondant :

```
#include<stdio.h>
#define R 0.3
int main(){
float Pt,P,Qte;
printf("Entrer le prix unitaire et la quantite : ");
scanf("%f%f", &P,&Qte)  ;
Pt = P * Qte ;
if(Qte>100)
   Pt = Pt- Pt*R;
printf("Prix total = %.2f",Pt);
}
```

4.3 Structure conditionnelle composée

La structure conditionnelle composée est formée par une suite d'instruction **if** imbriquées.
En C :

```
if(Condition1)
   bloc1;
   else
   if(Condition2)
       bloc2;
       else
       if(Condition3)
       bloc3;
       ..............
       else
       ..............
```

Les blocs d'instructions bloc1, bloc2,$\cdots$etc. peuvent à leur tour contenir des instructions **if**. Un exemple typique est celui de la résolution de l'équation du second degré.
Programme de résolution de l'equation du second degré $ax^2 + b.x + c = 0$.

```
#include<stdio.h>
#include<math.h>
int main(){
float a,b,c,d;
printf("\nEntrer les coefficients de l'equation ax2+bx+c=0\n");
scanf("%f%f%f", &a,&b,&c);
printf("\nResolution de l'equation (%.2f))x2+(%.2f)x+(%.2f)=0\n",a,b,c);
if(a==0)
   if(b==0)
      if(c==0)
       printf("Infinite de solutions\n");
       else
       printf("Impossible\n");
  else
  printf("Equation du premier degre : \n Une solution x = %.2f\n",-c/b);
else
{d = b*b-4*a*c;
if(d==0)
  printf("Solution double x1 = x2 = %.2f",-b/(2*a));
  else if(d>0)
   printf("Deux solutions relles : x1 = %.2f   x2=%.2f\n", (-b+sqrt(d))/(2*a),  (-b-sqrt(d))/(2*a));
   else
   printf("Deux solutions complexes : x1 = %.2f + (%.2f) i  \t x2 = %.2f + (%.2f) i \n", -b/(2*a),
    sqrt(-d)/(2*a),-b/(2*a), -sqrt(-d)/(2*a));
}
printf("\nMerci et Au revoir !\n");
}
```

Résultat de l'execution :

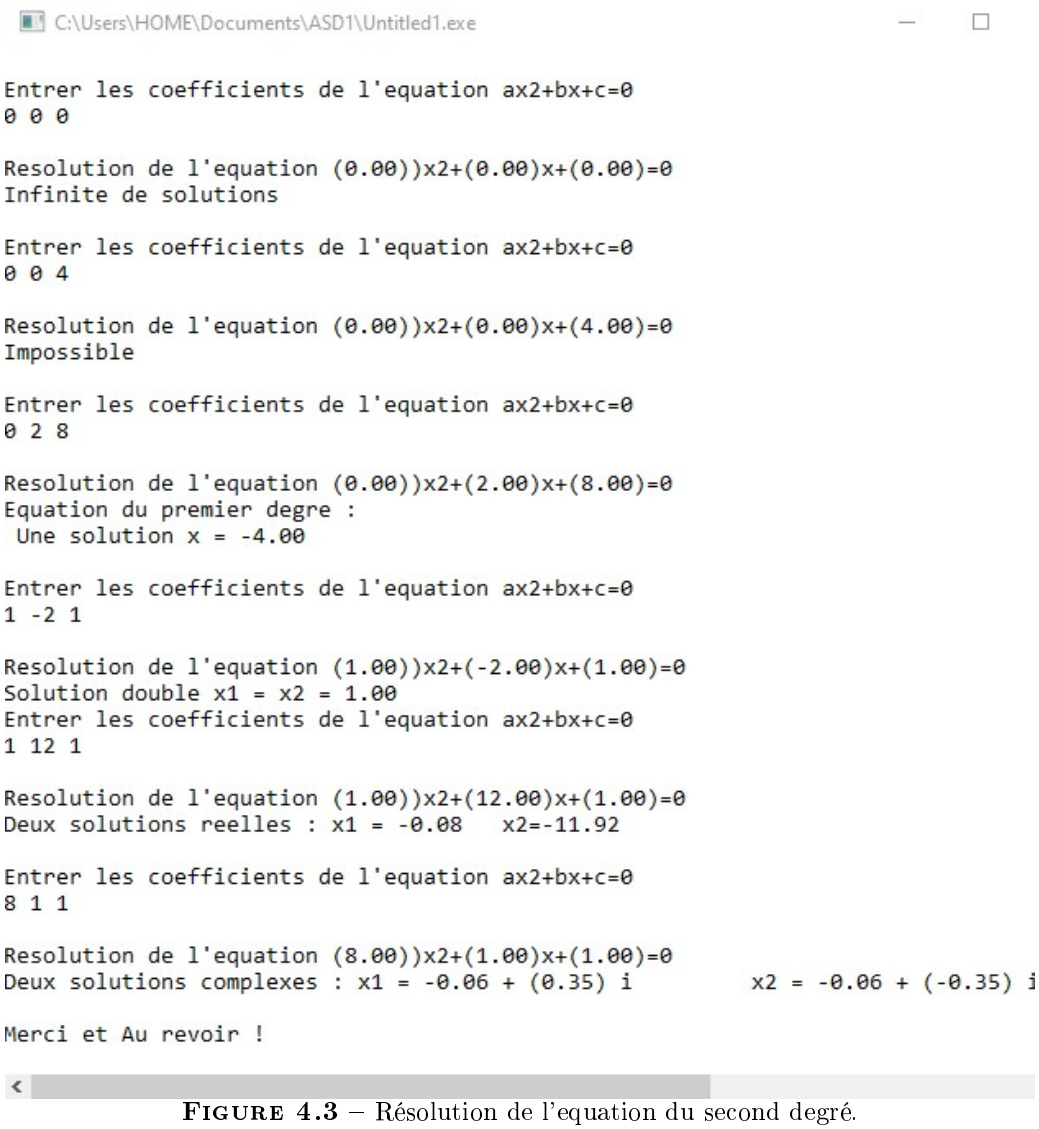


Figure 4.3 – Résolution de l'equation du second degré.

Exercice 5 *Comparer 3 nombres entiers a,b et c et afficher le plus grand.*

```
Algorithme comparaison  ;
Var
 a,b,c : entier   ;
Début
Ecrire('Entrer a, b et c : ');
Lire(a,b,c);
Si (a>b) alors
   Si (a>c) alors
              Ecrire('Max=',a);
              Sinon
              Ecrire('Max=',c);
        Sinon
        Si(b>c) alors
            Ecrire('Max=',b);
            Sinon
            Ecrire('Max=',c);
Fin.
```

En C :

```
#include<stdio.h>
int main(){
int a,b,c;
printf("Entrer a,b,c  : ");
scanf(" %d%d%d", &a,&b,&c)  ;
if(a>b)
    if(a>c)
          printf("Max= %d",a);
           else
           printf("Max=%d",c);
     else
     if(b>c)
      printf("Max=%d",b);
      else
      printf("Max=%d",c);
}
```

Pour généraliser le problème de comparaison à plusieurs valeurs puisque dans le cas où leur nombre excède 2 valeurs, la cascade de if imbriqués devient complexe à gérer, on procède en supposant la première valeur comme étant le minimum (respectivement maximum) et il sera remplacé au fur et à mesure par la valeur valeur qui lui est inférieure (respectivement supérieure).

La solution devient donc :

```
Algorithme comparaison  ;
Var
 a,b,c, Max : entier   ;
Début
Ecrire('Entrer a, b et c : ');
Lire(a,b,c);
Max <- a ;   (* On considère a priori que le maximum c'est la valeur a *)
Si (b>Max) alors Max <- b;
Si(c>Max) alors Max <- c;
Ecrire('Max=',Max);
Fin.
```

En C :

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

```
int a,b,c, Max;
printf("Entrer a,b,c  : ");
scanf(" %d%d%d", &a,&b,&c)  ;
Max = a;
if(b>Max)
    Max = b;
if(c>Max)
    Max = c;
printf("Max=%d",Max);
}
```

4.4 Structure conditionnelle de choix multiple ou aiguillage

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

```
switch(expression)
{
case valeur1 : instructions;
               break;
case valeur2 : instructions;
               break;
..............
case valeurn : instructions;
               break;
default : instructions;
}
```

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

- La séquence d'instructions suivant le mot clé **case** ne constitue pas un bloc d'instruction proprement dit, c'est l'instruction **break** qui empêche la poursuite du programme en séquence.
- Le mot-clé **break** est omis dans le cas où l'exécution doit se poursuivre à la suite d'instructions du **case** suivant. C'est souvent le cas quand certains **case** du **switch** ont le même traitement.
- L'instruction **default** correspond aux valeurs non spécifiées de l'expression du **switch**. Il est toujours recommandé de prévoir un traitement par défaut pour éviter que le programme se plante.

Exercice 6 *Simulation d'une calculatrice.*

```
#include <stdio.h>
int main()
{int a,b;
char c;
printf("Entrer l'operation");
scanf("%d%c%d", &a,&c,&b);
switch(c)
{
         case '+' : printf("Resultat = %d\n",a+b);
                    break;
         case '-' : printf("Resultat = %d\n",a-b);
                    break;
         case '*' : printf("Resultat = %d\n",a-b);
                    break;
         case '/' : if(b!=0)
                    printf("Resultat = %d\n",a/b);
                    else
                    printf("Division par 0\n");
                    break;
         default : printf("Mauvais symbole d'operation\n");
}}
```

Une même séquence d'instructions peut être rattachée à plusieurs cas comme dans l'exemple suivant qui distingue entre les symboles de ponctuation et les symboles d'opération arithmétique :

```
#include <stdio.h>
int main()
{
char c;
printf("Entrer le caractère");
c = getchar();
switch(c)
```

```
{
        case '+' :
        case '-' :
        case '*' :
        case '/' : printf("Symbole d'opération\n");
                   break;
        case ',' :
        case '.' :
        case ';' :
        case '?' :
        case '!' : printf("Symbole de ponctuation\n");
                   break;
        default : printf("Autre\n");
}}
```

4.5 Le branchement

Les instructions de branchement permettent d'interrompre le déroulement séquentiel du programme en faisant un saut (rupture de séquence) vers une instruction en aval ou en amont ou vers une fonction appelante (cas de return). Il y a cinq instructions de branchement en C : **break**, **continue**, **return**, **goto** et **exit** qui a un effet similaire à celui de return. L'instruction **break** a été introduite précédemment avec l'instruction **switch**, elle peut également être utilisée dans les boucles. Les instructions **break** et **continue** seront décrites dans le chapitre suivant.

4.5.1 Instruction return

L'instruction return est utilisée dans les sous-programmes (programmation modulaire) pour sortir d'une fonction invoquée et renvoyer le contrôle à la fonction appelante. Son utilisation dans la fonction principale (main), permet de rendre le contrôle au système d'exploitation. Elle permet de sortir immédiatement de la fonction en cours et retourner à la fonction appelante, avec éventuellement un renvoi d'une valeur résultat.
Syntaxe de l'instruction :

return expression ;

Si l'expression est omise dans la fonction main, cela sous-entend le retour de la valeur 0, par contre dans une fonction devant retourner un résultat (fonction non-void), la valeur retournée est indéfinie.

4.5.2 Instruction exit

La fonction exit, définie dans la librairie <**stdlib.h**>, permet de mettre fin à l'exécution d'un programme en spécifiant une valeur de retour.
Syntaxe de l'instruction :

exit (statut) ;

Le paramètre statut est un entier indiquant l'état de terminaison du programme. La valeur 0 (ou la constante **EXIT_SUCCESS**) signifie que le programme s'est terminé correctement, par contre une valeur non nulle ou la constante **EXIT_FAILURE** indique une terminaison anormale du programme. Les deux constantes **EXIT_SUCCESS** et **EXIT_FAILURE** sont définies également dans la librairie <**stdlib.h**>.

☞ Exemple : Résolution de l'equation du premier degré ax+b=0.

```
#include <stdio.h>
#include <stdlib.h> /* Librairie où sont définies la fonction exit() et les constantes EXIT_FAILURE
                       et EXIT_SUCCESS */
int main()
{ float a,b,x;
printf("Entrer les coefficients a et b : ");
scanf("%f%f", &a,&b);
if(a==0)
   {printf("Oups, division par 0 !\n"); /* En cas de division par 0, afficher le message
                                           et arrêter le grogramme */
    exit(EXIT_FAILURE); // Sortie avec échec en exécution
   }
x=-b/a;
printf("La solution x = %.2f\n",x);
exit(EXIT_SUCCESS); // Retourner la valeur 0 en cas de succès
}
```

4.5.3 Instruction goto

L'instruction **goto** réalise un saut inconditionnel à l'instruction étiquetée en amont ou en aval du code sans possibilité de retour au point de rupture. L'étiquette est un identificateur suivi du caractère « ; » souvent mis en retrait des instructions qu'elle désigne. Elle permet de sortir de plusieurs boucles imbriquées, possibilité que ne permet pas l'instruction break.

Syntaxe de l'instruction :

Etiquette :
    instructions ;
goto Etiquette ;

☞ Exemple : Résolution de l'equation du premier degré ax+b=0.

```
#include <stdio.h>
int main()
{ float a,b,x;
printf("Entrer les coefficients a et b : ");
scanf("%f%f", &a,&b);
if(a==0) goto Probleme ; // En cas de division par 0 se diriger vers l'étiquette Probleme
x=-b/a;
printf("La solution x = %.2f\n",x);
return 0; // Retourner la valeur 0 en cas de succès
Probleme : printf("Division par 0 \n");  /* Probleme est l'étiquette de branchement en cas de
```

```
                                        division par 0 */
return 1; // Retourner la valeur 1 en cas de division par 0
}
```

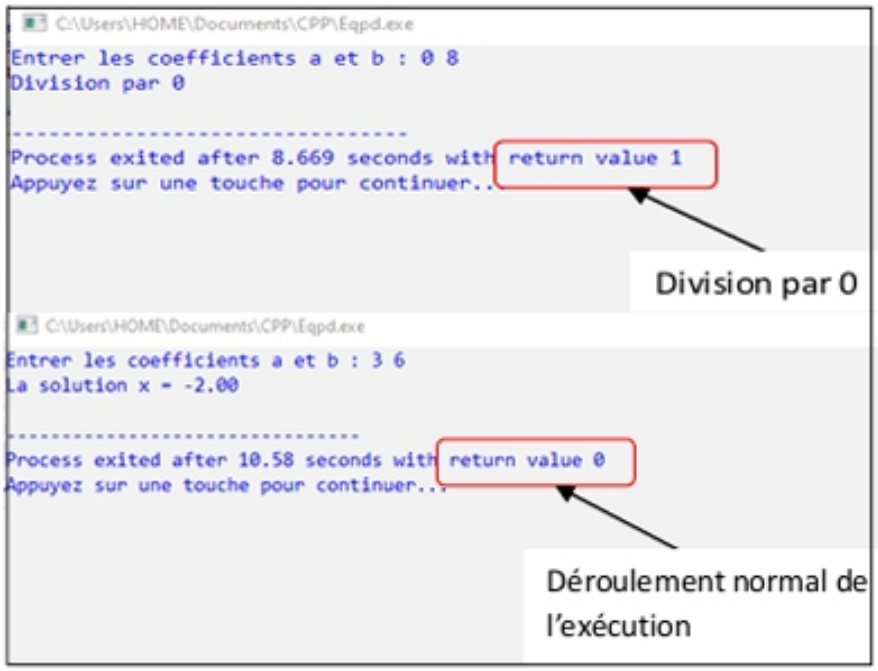


Figure 4.4 – Déroulement de l'instruction goto.

CHAPITRE 5

INSTRUCTIONS DE RÉPÉTITION OU BOUCLES

Sommaire

Figures

5.1 Introduction

Les structures de répétition communément appelées boucles permettent d'exécuter plusieurs fois la même suite d'instructions. La répétition est réalisée selon une condition de contrôle dépendant d'une expression ou variable compteur qui est incrémentée ou décrémentée jusqu'à arriver à une valeur finale. Le langage C fournit trois instructions pour réaliser des boucles.

5.2 Instruction Tant que (while)

Tant que la condition est fausse, le contenu de la boucle est exécuté. Comme la condition est testée en début de boucle, il est possible de ne jamais entrer dans le corps de la boucle.
Syntaxe de l'instruction Tant que :

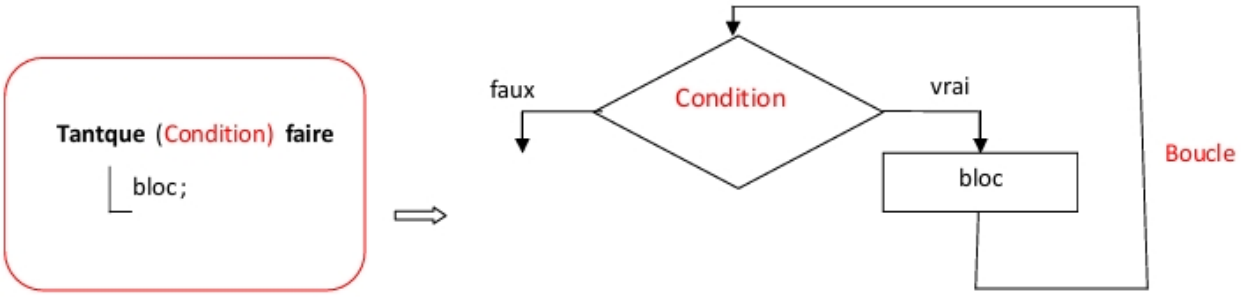


Figure 5.1 – Syntaxe de la boucle Tant que.

En C :

```
while(Condition)
        bloc;
```

☞ Exemple 1 : Calcul de la somme suivante :

$$s = \sum_{i=1}^{n} i \tag{5.1}$$

```
Algorithme somme_cumulee ;
Var i,n,s : entier ;
Début
Ecrire('Entrer la valeur de n : ') ;
Lire(n) ;
s <- 0; i <- 1;
Tant que i <= n faire
   Début
       s <- s+1;
      i <- i+1;
         Fin;
Ecrire('s=',s);
```

En C :

```
#include <stdio.h>
int main()
{ int n,i=1, s=0; // Initialisation de la variable i à 1 et la variable s à 0
printf("Entrer n : ");
scanf("%d",&n);
while(i<=n)
 {s=s+i ; // Incrémentation de la variable s d'un pas i
   i=i+1 ; // Ou bien i++ ; incrémentation de la variable i d'un pas 1
   }
printf("s= %d\n",s);
}
```

☞ Exemple 2 : Calculer la factorielle d'un nombre entier n. En utilisant la formule :

$$n! = 1 \times 2 \times 3 \times \cdots \times n \tag{5.2}$$

center

```
Algorithme factorielle ;
Var i,n,f : entier ;
Début
Ecrire('Entrer la valeur de n : ') ;
Lire(n) ;
f <- 1; i <- 1;
Tant que i <= n faire
   Début
       f <- f*i;
         i <- i+1;
         Fin;
Ecrire(n,'!=',f);
```

En C :

```
#include <stdio.h>
int main()
{ int n,i=1;
long f=1;  // La variable f est déclarée long int (puisque la valeur croît très vite)
printf("Entrer n : ");
scanf("%d",&n);
while(i<=n)
 {f=f*i ;
  i++ ;
   }
printf("%d! = %ld\n",n,f); // La variable f est déclarée long int et donc affichée avec le format "%ld
    ")
}
```

☞ Exemple 3 : Calculer la factorielle d'un nombre entier n. En utilisant la formule :

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1 \tag{5.3}$$

```
Algorithme factorielle ;
Var n,f : entier ;
Début
Ecrire('Entrer la valeur de n : ') ;
Lire(n) ;
Ecrire(n,'!='); (* Afficher la valeur initiale de n d'abord puisqu'après, elle sera modifiée
f <- 1;
Tant que n >= 2 faire
   Début
       f <- f*n;
      n <- n-1;
   Fin;
Ecrire(f);
```

En C :

```c
#include <stdio.h>
int main()
{ int n;
long f=1;  // La variable f est déclarée long int (puisque la valeur croît très vite)
printf("Entrer n : ");
scanf("%d",&n);
printf("%d! = ",n); //Afficher la valeur initiale de n d'abord puisqu'après, elle aura été modifiée
while(n>=2)
 {f=f*n ;
  n=n-1 ; // Ou bien n--; décrémentation de la valeur de n
   }
printf("%ld\n",f);
}
```

☞ Exemple 4 : Afficher le nombre de chiffres d'un nombre entier.

```c
#include <stdio.h>
int main() {
int nombre, compteur = 0, aux; // aux variable auxiliaire le traitement
printf("Entrez un nombre entier : ");
scanf("%d", &nombre);
// Cas où nombre est nul
if (nombre == 0)
        compteur = 1;
        else
     aux = nombre; // Préserver la valeur initiale de nombre pour l'affichage
   // Cas où nombre<0
  if (aux < 0)
            aux = -aux;
 // Compter les chiffres
    while (aux != 0) {
```

```
            aux = aux / 10;
            compteur++;
        }
printf("Le nombre %d contient %d chiffre(s)\n", nombre, compteur);
}
```

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

if(Condition1 && Condition2)
if(Condition1 || Condition2)

5.3 Instruction Faire...Tant que (do...while)

Á l'inverse de l'instruction Tant que faire (**While**), l'instruction Faire ··· Tant que (**do while**) réalise le test aprés l'exécution du bloc d'instructions ce qui fait que le bloc d'instructions sera exécuté au moins une fois.
Syntaxe de l'instruction Tant que :

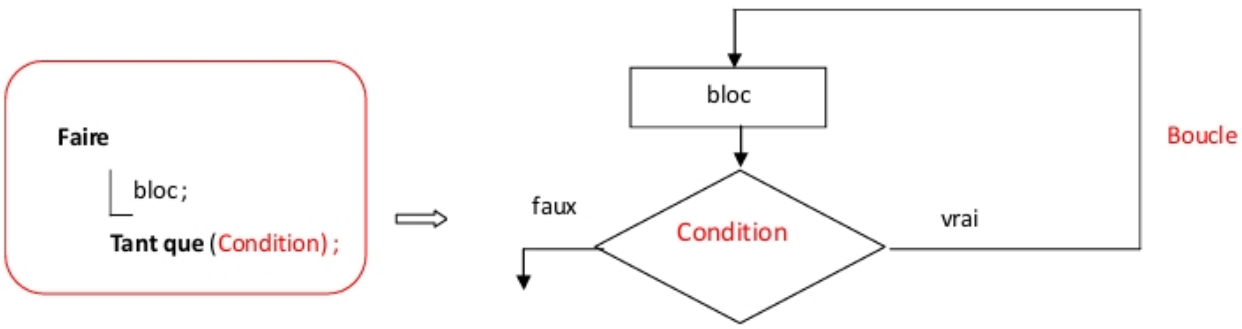


Figure 5.2 – Syntaxe de la boucle Faire...Tant que.

En C :

```
do
   bloc;
while(Condition);
```

Avec l'instruction faire ...Tantque ou do ..while, la condition de la boucle est vérifiée à la fin. La différence avec l'instruction Tant que ou while est que celle-ci s'exécute au moins une fois avant de vérifier la condition.

☞ Exemple 1 : Calculer la factorielle d'un nombre entier n. En utilisant la formule :

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1 \tag{5.4}$$

```
Algorithme factorielle ;
Var i,n,f : entier ;
Début
Ecrire('Entrer la valeur de n : ') ;
Lire(n) ;
Ecrire(n,'!=');
f <- 1;
Faire
          Début
      f <- f*n;
      n <- n-1;
      Fin;
Tant que (n > 1) ;
Ecrire(f);
Fin.
```

En C :

```
#include <stdio.h>
int main()
{ int n;
long f=1;
printf("Entrer n : ");
scanf("%d",&n);
printf("%d! = ",n);
do {
  f=f*n ;
  n-- ;
   } while(n>1) ;
printf("%ld\n",f);
}
```

☞ Exemple 2 : Écrire l'algorithme permettant de calculer le plus grand commun diviseur (PGCD) de deux nombres a et b positifs non nuls tels que $a > b$ en utilisant la méthode d'Euclide (division euclidienne).

Principe : Le PGCD de deux nombres a et b peut s'obtenir par la division de a par b , puis de b par le reste obtenu et ainsi de suite jusqu'à ce que l'on obtienne un reste nul, le dernier diviseur est alors le PGCD de a et b. En utilisant l'instruction Faire...Tantque (do...while) et en utilisant l'instruction Tantque (while) :

```
Algorithme pgcd;
Var a,b,r : entier;
Début
Ecrire('Entrer a et b avec b non nul:');
Lire(a,b);
Tantque (b <> 0) faire
         Début
         r=a mod b;   (* mod est le modulo qui est le reste de la division entière *)
         a <- b ;
         b <- r;
         Fin;
Ecrire('PGCD=', a);
Fin.
```

En C :

```
#include <stdio.h>
int main(){
int a,b,r;
printf ("Donner a et b avec b non nul: ");
scanf("%d%d",&a,&b);
while(b!=0)
   {r=a%b; //Le le symbole % représente le modulo qui est le reste de la division entière
   a=b;
   b=r;
}
printf(" Le PGCD est : %d\n",a); // Le PGCD se trouve dans la variable a
}
```

En utilisant l'instruction Faire ...Tantque (do ...while)

```
Algorithme pgcd;
Var a,b,r : entier;
Début
Ecrire('Entrer a et b avec b non nul:');
Lire(a,b);
Faire Début
      r <- a mod b;
    a <- b ;
   b <- r;
  Fin;
         Tantque(b <> 0) ;
Ecrire('PGCD=', a);
Fin.
```

En C :

```c
#include <stdio.h>
int main(){

int a,b,r;
printf ("Donner a et b avec b non nul: ");
scanf("%d%d",&a,&b);
do
{ r=a%b;
   a=b;
   b=r;
}
while(b!=0);
printf(" Le PGCD est : %d\n",a); // Le PGCD se trouve dans la variable a
}
}
```

Le calcul du PGCD par la méthode des différences succèssives est un bon exemple pour montrer que les deux boucles while et do...while n'ont pas le même comportement ou effet sur les variables sur lesquelles elles agissent.

☞ Exemple 3 : Écrire l'algorithme permettant de calculer le plus grand commun diviseur (PGCD) de deux nombres a et b positifs non nuls en utilisant la méthode des diffèrences successives.
Principe : Le PGCD de deux nombres a et b est le PGCD de la différence des deux nombres et le plus petit d'entre eux.
Si a=b alors PGCD(a,b) = a = b
Si (a>b) alors PGCD(a,b) = PGCD(a-b,b)
Si (b>a) alors PGCD(a,b) = PGCD(a,b-a)
En utilisant l'instruction Faire...Tantque (do...while) et en utilisant l'instruction Tantque (while), il vient :

```
Algorithme pgcd;
Var a,b: entier;
Début
Ecrire('Entrer a et b :');
Lire(a,b);
Tantque (a<>b) faire
    Si (a>b) alors a <- a-b;
             sinon b <- b-a;
Ecrire('PGCD=', a);
Fin.
```

En C :

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

```
int main(){
int a,b,r;
printf ("Donner a et b : ");
scanf("%d%d",&a,&b);
while(a!=b)
    if(a>b) a=a-b;
      else b=b-a;
printf(" Le PGCD est : %d\n",a);
}
```

En utilisant l'instruction Faire ...Tantque -do ...while)

```
Algorithme pgcd;
Var a,b: entier;
Début
Ecrire('Entrer a et b avec b non nul:');
Lire(a,b);
Faire
Si(a>b) alors a <- a-b;
    sinon Si(b>a) alors b <- b-a; (* On vérifie si b>a pour éviter de calculer la différence si a=b *)
         Tantque(a<>b) ;
Ecrire('PGCD=', a);
Fin.
```

En C :

```
#include <stdio.h>
int main(){
int a,b;
printf ("Donner a et b : ");
scanf("%d%d",&a,&b);
do
if(a>b) a=a-b;
 else if(b>a) b=b-a; // On vérifie si b>a pour éviter de calculer la différence si a=b
while(a!=b);
printf(" Le PGCD est : %d\n",a); // Le PGCD se trouve dans la variable a
}
}
```

5.4 Instruction Pour (for)

Syntaxe :

```
for(expression1;expression2;expression3)
   bloc;
```

Equivalente à :

```
expression1;
while(expression2)
{
bloc;
expression3;
}
```

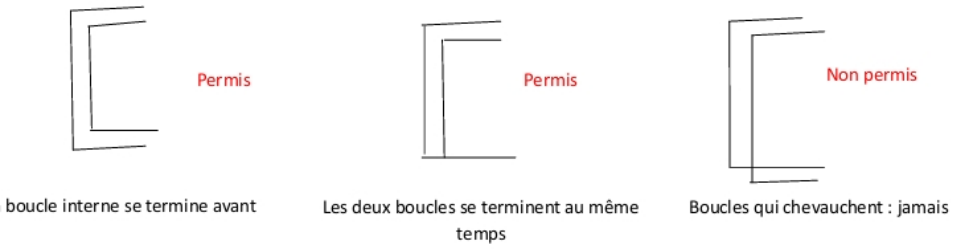
L'expression1 permet d'initialiser la variable compteur avant l'entrée dans la boucle, la condition expression2 réalise le test de continuation de boucle pour entamer la prochaine itération de la boucle, expression3 est évaluée à la fin de chaque itération pour faire progresser la boucle.
☞ Exemple : : calcul de la somme $s = x + \frac{x^2}{2} - \frac{x^3}{3} + \cdots + (-1)^n \frac{x^n}{n}$.
Entrée : x variable réelle et n entier naturel.
Sortie : la somme s réelle.

```
#include <stdio.h>
#include <math.h>
int main()
{
   int i,n,signe=1;  // variable signe varie entre 1 et -1
   float s=0,x;
   printf("Entrez la valeur x : ");
   scanf("%f",&x);
   printf("Entrez la valeur n : ");
   scanf("%i",&n);
   for(i=1;i<=n;i++)
   {
      s=s+signe*pow(x,i)/i;
      signe=-signe;
   }
   printf("s=%f\n",s);
}
```

5.5 Les boucles imbriquées

Les boucles sont dites imbriquées lorsqu' une boucle est placée à l'intérieur d'une autre. Cela permet de répéter une suite d'instructions pour chaque itération de la boucle externe (englobante). Les boucles imbriquées peuvent être réalisées avec les trois types de boucles vues précédemment (while, do...while et for).

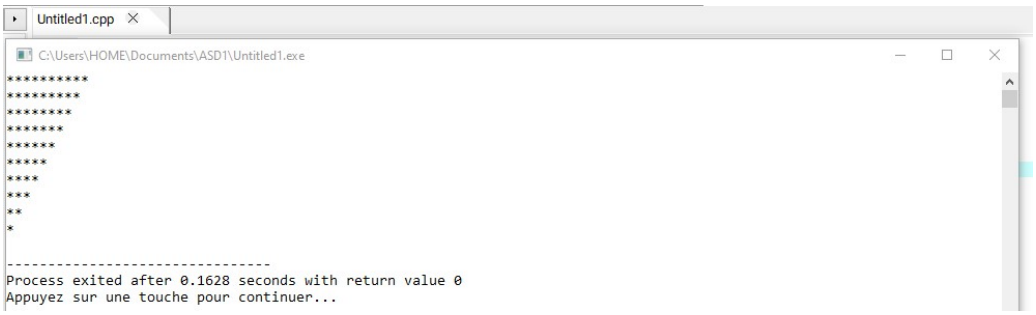
Une boucle peut contenir à son tour une ou plusieurs boucles mais attention la boucle interne doit toujours se terminer en premier.



☞ Exemple : Soit à afficher un triangle d'étoiles

```
#include<stdio.h>
int main()
{ int i,j;
for(i=1;i<=10;i++)
    {
   for(j=10;j>=i;j--)
      printf("*");
 printf("\n");
}}
```

Résultat de l'exécution :



Exercice 7 *Que font les deux programmes suivants ?*

Programme 1 :

```
#include<stdio.h>
int main(){
int   i=1, j;
while(i<=5)
    {    j=1;
          while( j<=i )
          { printf("*");
              j++; }
        printf("\n");
```

```
    i++;     }}
```

Programme 2 :

```
#include<stdio,h>
int main(){
int   i=1, j;
wle(i<=10)
    {    j=1;
          while( j<=10 )
          { printf("%d\t",i*j);
              j++; }
         printf("\n");
         i++;     }}
```

5.6 Instructions de rupture de séquence

Les instructions de rupture de séquence permettent de rompre le déroulement séquentiel d'une suite d'instructions. Ces instructions sont :

break : provoque l'interruption de la boucle (où elle se trouve) ou du branchement conditionnel la/le plus proche, en passant le contrôle à l'instruction la plus imbriquée qui suit. ☞ Exemple : Code demandant le mot magique offrant 5 tentatives.

```
#include<stdio.h>
int main()
{
    int i, magique = 14,n;
    for(i=1;i<=5;i++)
    {printf("Entrez le nombre magique (entre 10 et 20): ");
    scanf("%d",&n);
    if(5-i==0) printf("Pas de chance, vous avez perdu)!\n");
     else{
       if(n<magique) printf("Oups, essayez un nombre plus grand (attention, il vous reste %d
    tentatives)!\n", 5-i);
        if(n>magique) printf("Oups, essayez un nombre plus petit (attention, il vous reste %d
    tentatives)!\n", 5-i);
       if(n==magique)
      {printf("Bravo! c'est le nombre magique!\n");
       break;
      }
    }}
    return 0;
}
```

Résultat de l'exécution :

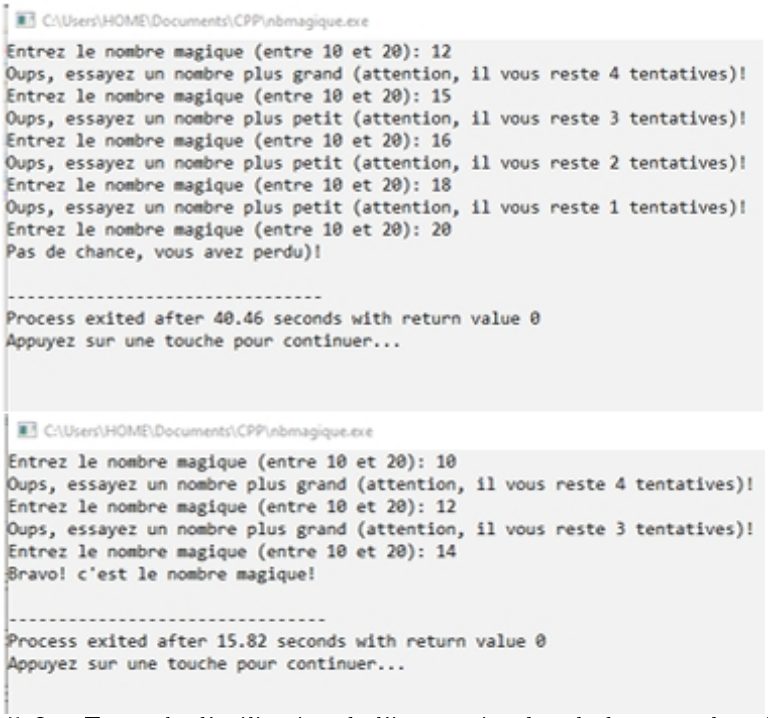


FIGURE 5.3 – Exemple d'utilisation de l'instruction break dans une boucle.

Exercice 8 *Donner le code qui vérifie si un nombre entier n lu en entrée est premier.*

```c
#include <stdio.h>
int main() {
    int n, i;
    int estPremier = 1; // La valeur 1 signifie vrai
    printf("Entrez un nombre: ");
    scanf("%d", &n);
    if (n <= 1)
        estPremier = 0;
     else
        for (i = 2; i < n; i++) {
            if (n % i == 0) {
                estPremier = 0; // L'entier n n'est pas premier
                break; // break arrête la boucle dès qu'un diviseur est trouvé
            }
    if (estPremier)
        printf("%d est un nombre premier.\n", n);
    } else
        printf("%d n'est pas un nombre premier.\n", n);
}
```

Attention

L'instruction break est locale à sa boucle. Avec par exemple, deux boucles imbriquées (une externe for et une interne for) break n'arrête que la boucle où elle se trouve (la boucle interne), tandis que la boucle externe continue son exécution normale comme le montre cet exemple de code C :

```c
#include <stdio.h>
int main() {
    int i, j;
    for (i = 1; i < 5; i++) {
        printf("Valeur de i = %d\n", i);
        for (j = 1; j < 3; j++) {
            printf("  Valeur de j = %d\n", j);
            if (j == 2)
                break;  // Arrête uniquement la boucle j
     }}
    printf("Boucle externe i terminee normalement.\n");
}
```

Exécution :

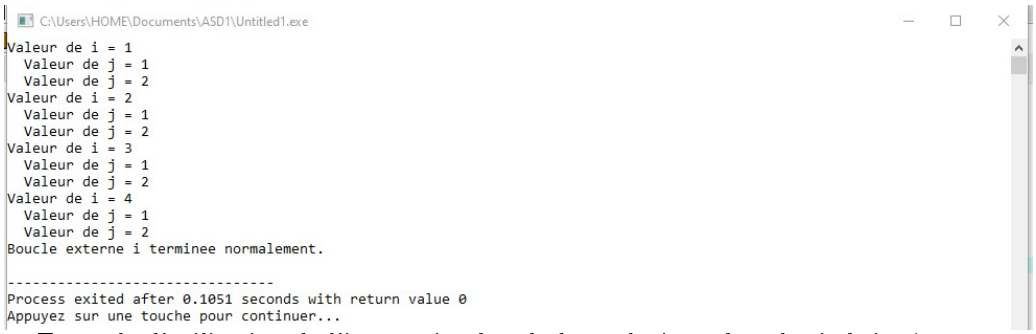


Figure 5.4 – Exemple d'utilisation de l'instruction break dans plusieurs boucles imbriquées : rupture de la boucle où elle se trouve

Pour rompre les deux boucles au même temps :

```c
#include <stdio.h>
int main() {
    int i, j;
    for (i = 1; i < 5; i++) {
        printf("Valeur de i = %d\n", i);
        for (j = 1; j < 3; j++) {
            printf("  Valeur de j = %d\n", j);
            if (j == 2)
                break;  // Arrête uniquement la boucle j
     }
      if (j == 2)
                break;  // Arrête aussi la boucle i
      }
    printf("Boucle externe rompue au meme temps que la boucle interne.\n");
}
```

Exécution :

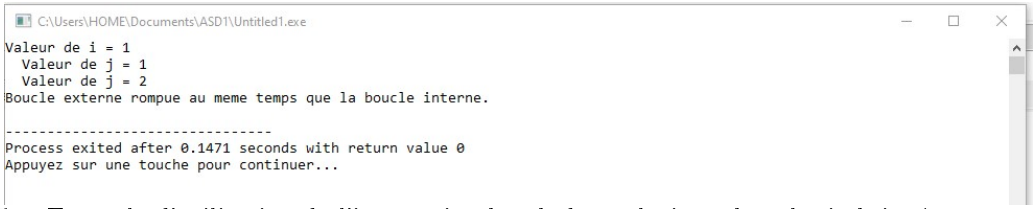


FIGURE 5.5 – Exemple d'utilisation de l'instruction break dans plusieurs boucles imbriquées : rupture des deux boucles

continue force le passage à l'itération suivante de la boucle la plus proche (interrompe l'exécution des instructions sans quitter la boucle).

☞ Exemple : lecture d'une suite de 10 valeurs entières et affichage du nombre total de valeurs nulles lues.

```c
#include<stdio.h>
int main()
{
int i,n,nbz=0;
for(i=1;i<=10;i++)
{
printf("Entrez un nombre entier : ");
scanf("%d",&n);
if (n != 0) continue ;
nbz++;
}
printf("Le nombre de valeurs nulles lues est : %d\n",nbz);
return 0;
}
```

Le résultat est :

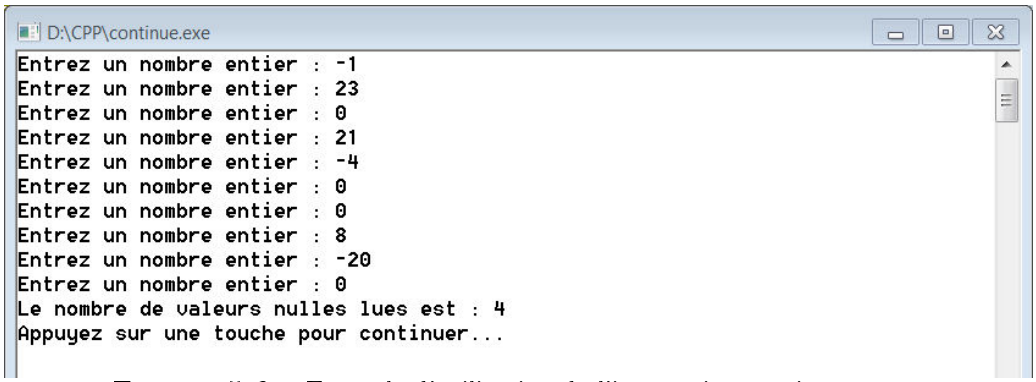


FIGURE 5.6 – Exemple d'utilisation de l'instruction continue.

CHAPITRE 6

LES TABLEAUX ET LES CHAÎNES DE CARACTÈRES

Sommaire

Figures

6.1 Introduction

6.2 Le type tableau

Les variables, telles que nous les avons présentées dans le chapitre 3.3, ne permettent de stocker qu'une seule donnée à la fois. Or, il arrive qu'on ait besoin d'utiliser plusieurs données et manipuler plusieurs variables indépendantes devient difficile à gérer. Le langage C propose la structure tableau pour stocker un ensemble de données de même type dans une variable commune composée dite variable indicée dont le nombre de cases est égal au nombre de variables simples qu'on veut manipuler, comme il est illustré dans la Figure 6.1 . Chaque variable simple est repérée par un indice indiquant sa position dans le tableau. les cases sont numérotées à partir de zéro, c'est à dire le plus petit indice est zéro.

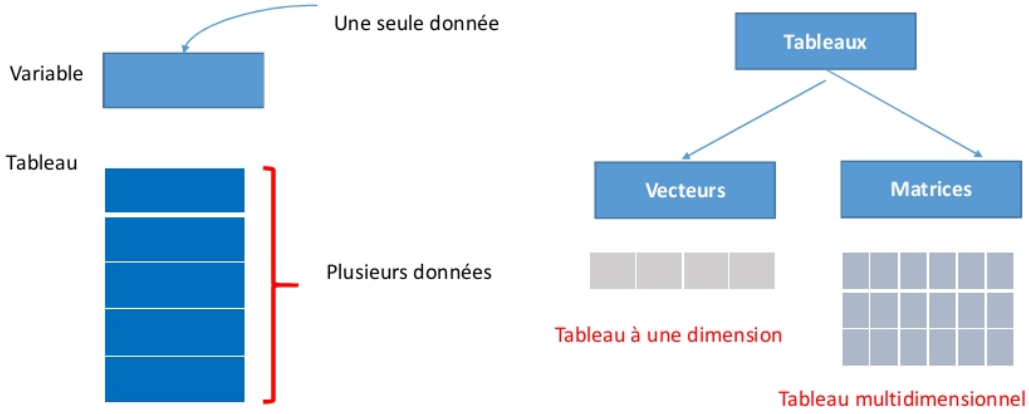


Figure 6.1 – Le Type Tableau

6.3 Les tableaux unidimensionnels

Un tableau à une dimensiion est caractérisé par :

- Un nom.
- Le type des éléments qu'il contient.
- Le nombre d'éléments ou taille.

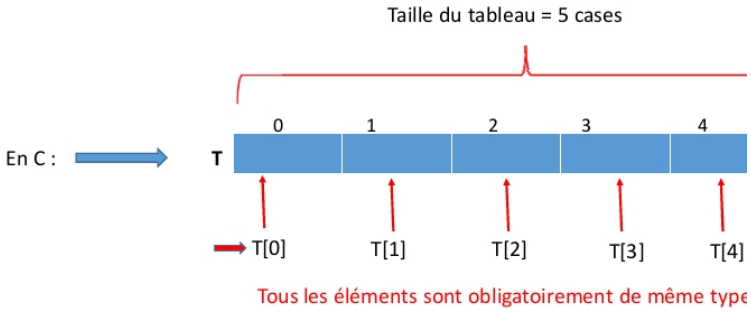


Figure 6.2 – Structure Tableau.

6.3.1 déclaration du tableau

Tous les éléments d'un tableau sont obligatoirement du même type. Pour les tableaux statiques, la taille est fixée une bonne fois pour toute au début du programme.

Syntaxe de déclaration en algorithmique :

Var identificateur[1..Taille] de Type ;

☞ Exemple :

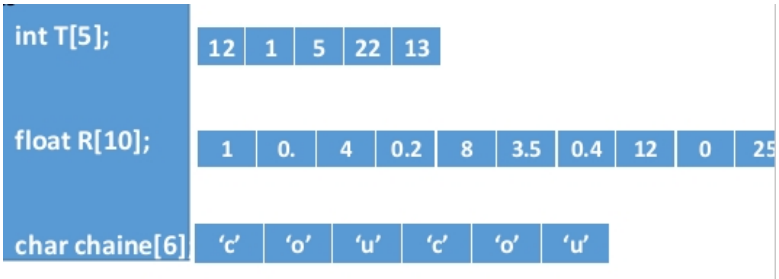
```
Algorithme Exemple;
Var
T : Tableau[1..5] d'entiers ;
R: Tableau[1..10] de réels;
chaine : Tableau[1..6] de caractères ;
```

Syntaxe de déclaration en C :

Type identificateur[Taille] ;

Où
Type : le type des éléments du tableau et donc celui du tableau.
identificateur : représente le nom du tableau qui est commun à tous les éléments du tableau.
Taille : nombre d'éléments du tableau.

```
int main(){
int T[5];
float R[10];
char chaine[6];
----------------
```



On peut initialiser un tableau dès sa déclaration en lui affectant une liste de valeurs séparées par des virgules et entourée par des accolades comme suit :

identificateur $=\{\text{valeur}_1, valeur_2, \cdots, \text{valeur}_n\}$

☞ Exemple :

```
int tab[6] = {1,2,15,10,0,3}; /* Les valeurs spécifiées seront placées dans les 6 case de tab */
char voyelles[6] = { 'a', 'e', 'i', 'o', 'u', 'y' }; /* Le tableau voyelles est rempli de la mê
    me manière */
```

```
int tab[6] = {1,3}; /* Initialisation partielle, tab[0]=1 et tab[1]=3 les autres cases du
    tableau sont indéfinies */
```

6.3.2 Accès à un élément du tableau

L'accès à élément d'un tableau se fait comme suit :

identificateur[indice]

Où
identificateur : représente le nom du tableau.
indice : est la position de l'élément dans le tableau.

Dans le langage C, l'indice d'un tableau commence à la valeur 0.

☞ Exemple :

```
tab[5] = 12 ;  \* affecte la valeur 12 au dernier élément du tableau tab */
voyelles[0] = 'i' ; \* affecte le caractère 'i' au premier élément du tableau voyelles*/
```

6.3.3 Lecture et écriture d'un tableau

Syntaxe de lecture et écriture en algorithmique :

Lire(identificateur[indice]) ;

Ecrire(identificateur[indice]) ;

Syntaxe en C :

scanf("format", &identificateur[indice]) ;

printf("format",identificateur[indice]) ;

☞ Exemple : soit à lire et à écrire les éléments d'un tableau à une dimension T de taille n, T(n).

```
Algorithme Exemple;
Var i : entier;
T : Tableau[1..10] d'entiers ;
Début
Ecrire(Entrer les éléments du tableau :);
Pour i=1 à 10 faire
                lire(T[i]);
Ecrire(Tableau lu est : ) ;
```

```
Pour i=1 à 10 faire
                Ecrire(T[i]);
Fin.
```

Programme en C :

```c
#include<stdio.h>
int main()
{ int i, T[10];
printf("Entrer les elements du tableau :");
for(i=0;i<10;i++)
      scanf("%d",&T[i]);
printf( "Le tableau lu est :");
for(i=0;i<10;i++)
      printf("%d\t",T[i]); // Affichage sur la même ligne avec tabulation entre les valeurs
}
```

Nous verrons par la suite qu'il est possible de créer dynamiquement des tableaux par soucis de gaspillage de la mémoire. La déclaration statique des tableaux engendre une perte de mémoire considérable puiqu'en pratique les cases réservées ne sont pas toutes éxploitées. Néanmoins en C++, il est possible de différer la déclaration d'un tableau après la lecture de la taille effective de celui-ci, comme dans l'exemple ci-dessous donné auparavant, permettant de rechercher une valeur x dans un tableau T dont la taille est déclarée après la lecture de la taille effective.

```c
#include <stdio.h>
int i,n;
int main()
{
     printf("Entrer la taille n du tableau : ");
     scanf("%d",&n);
     float T[n];    // déclaration différée du tableau T avec la taille éffective n lue en entré
    e
     printf("Entrer les  elements du tableau : ");
     for(i=0;i<n;i++)
     scanf("%f", &T[i]);
    }
```

Les traitements sur les tableaux sont divers, donnons quelques exemples.

Exercice 9 *Elimination des valeurs nulles dans un tableau T d'entiers de taille n.*

```c
#include <stdio.h>
#define taille 60
int T[taille],i,n,p=0; // T tableau d'entiers
int main()
{
     printf("Entrer la taille n du tableau : ");
     scanf("%d",&n);
     printf("Entrer les  elements du tableau : ");
     for(i=0;i<n;i++)
     scanf("%d", &T[i]);
     printf("\nTableau initial : ");
     for(i=0;i<n;i++)
     printf("%d\t", T[i]);
     // Elinination des valeurs nulles
     i=0;
     while(i<n)
            { if(T[i]!=0)
                {
                  T[p]=T[i];
                  p++;
                }
              i++;
            }
     printf("\nTableau final : ");
     for(i=0;i<p;i++)
     printf("%d\t", T[i]);
     printf("\n");
     }
```

Résultat de l'exécution :

```
C:\Users\HOME\Documents\ASD1\Untitled1.exe
Entrer la taille n du tableau : 8
Entrer les  elements du tableau : 10
0
0
8
0
2
0
0

Tableau initial : 10    0       0       8       0       2       0       0
Tableau final : 10      8       2

--------------------------------
Process exited after 19.99 seconds with return value 0
Appuyez sur une touche pour continuer...
```

Figure 6.3 – Élimination des valeurs nulles dans un tableau.

Exercice 10 *Recherche d'une valeur x dans un tableau de réels de taille n.*

```c
#include <stdio.h>
#define taille 30
float T[taille], x; // T tableau de réels et x la valeur cherchée
int i,n;
int main()
{
     printf("Entrer la taille n du tableau : ");
     scanf("%d",&n);
     printf("Entrer les  elements du tableau : ");
     for(i=0;i<n;i++)
     scanf("%f", &T[i]);
     printf("Entrer la valeur x à chercher : ");
     scanf("%f",&x);
     i=0;
     while(i<n && T[i]!=x)
                 i++;
      if(i<n)
            printf("la valeur %2.2f est trouvee à la position %d\n", x,i+1) ; // Ajouter 1 à la
    position trouvée pour l'affichage
            else
            printf("la valeur %2.2f ne se trouve pas dans T\n");
     }
```

Résultat de l'exécution :

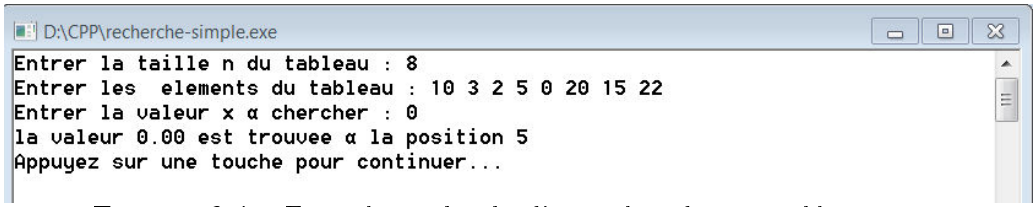


Figure 6.4 – Exemple : recherche d'une valeur dans un tableau.

Exercice 11 *: Algorithme et programme C permettant de lire un tableau de 10 valeurs et de rechercher le minimum et le maximum.*

```
Algo minmax;
Var i,min,max : entier;
T : Tableau[1..10] dentiers ;
Début
Ecrire(Entrer les éléments du tableau :);
Pour i=1 à 10 faire
                                  lire(T[i]);
min=T[1]; max = T[1];
Pour i=2 à 10 faire
       Début
      Si(T[i]<min) alors min = T[i];
      Si(T[i]>max) alors max=T[i];
     Fin;
   Ecrire(Le min est :,min, Le max est :,max);
Fin.
```

Le programme en C :

```
#include<stdio.h>
int main()
{ int i, T[10],min,max;
printf("Entrer les elements du tableau :");
for(i=0;i<10;i++)
      scanf("%d",&T[i]);
min=T[0];
max=T[0];
for(i=1;i<n;i++)
{
      if(T[i]<min)
            min = T[i];
      if(T[i]>max)
          max = T[i];
printf("Le min est : %d   Le max est %d\n",min,max);
}
```

Exercice 12 *Insertion d'une valeur val à une position p lue en entrée dans un tableau T de taille n d'entiers.*

```
#include <stdio.h>
#define Taille 100
int main() {
      int n, i, p, val;
            int T[Taille];
    printf("Entrer la taille du tableau : ");
    scanf("%d", &n);
      // Lecture du tableau
    printf("Entrer les %d elements du tableau:\n", n);
    for(i = 0; i < n; i++)
        scanf("%d", &T[i]);
  // Lecture de la valeur à insérer
    printf("Entrer la valeur a inserer: ");
    scanf("%d", &val);
      // Lecture de la position
    printf("Entrer la position d'insertion (0 <= position <= %d): ", n);
    scanf("%d", &p);

    if (p < 0 || p > n) {
        printf("Position invalide.\n");
        return 1;
    }
      // Affichage du tableau avant insertion
      printf("'Tableau initial : ");
      for(i = 0; i < n; i++)
        printf("%d\t", T[i]);
    // Décaler les éléments vers la droite pour libérer la position p-1
```

```
    for(i = n; i > p-1; i--) // p-1 puisque les indices en C commencent à 0
        T[i] = T[i - 1];
    // Insertion de la valeur val à la position p
    T[p-1] = val;
    n++; // Taille du tableau devenue n+1
      // Affichage du tableau après insertion
    printf("\nTableau apres insertion:\n");
    for(i = 0; i < n; i++)
        printf("%d\t", T[i]);
    printf("\n");
    return 0;
}
```

Résultat de l'exécution :

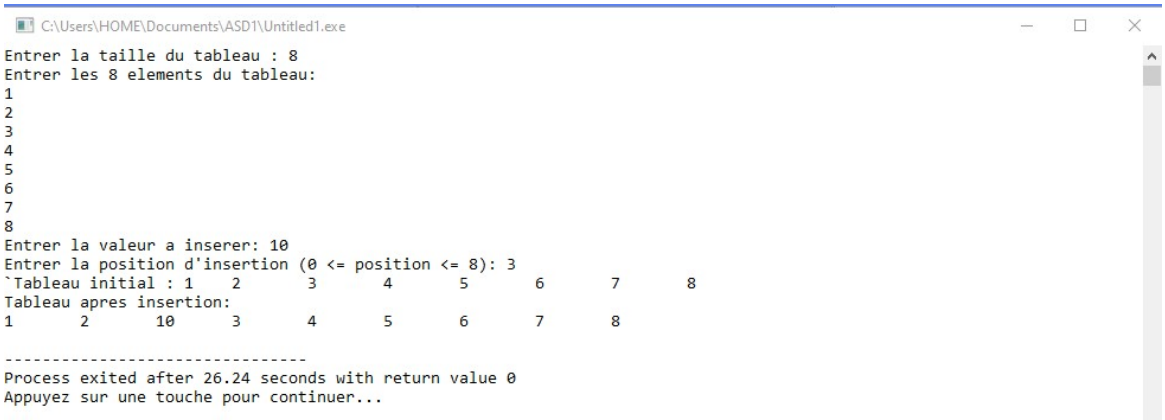


FIGURE 6.5 – Exemple : insertion d'une valeur dans un tableau.

Exercice 13 *suppression d'une valeur val dans un tableau T de taille n d'entiers.*

```
#include <stdio.h>
#define Taille 100
int main() {
    int T[Taille], n, i, j = 0, val;
    // Lecture de la taille du tableau
    printf("Entrez la taille du tableau: ");
    scanf("%d", &n);
    // Lecture du tableau
    printf("Entrez les elements du tableau: ");
    for (i = 0; i < n; i++)
        scanf("%d", &T[i]);

    // Lecture de la valeur à supprimer
    printf("Entrez la valeur a supprimer: ");
    scanf("%d", &val);
      // Affichage du tableau avant suppression
    printf("\nTableau avant suppression: ");
    for (i = 0; i < j; i++)
        printf("%d\t", T[i]);
    // Suppression de toutes les occurrences de val
    for (i = 0; i < n; i++)
```

```
        if (T[i] != val)
            T[j++] = T[i];
      // Affichage du tableau après suppression
    printf("\nTableau apres suppression: ");
    for (i = 0; i < j; i++)
        printf("%d\t", T[i]);
    printf("\nSa taille : %d\n", j);
    return 0;
}
```

Résultat de l'exécution :

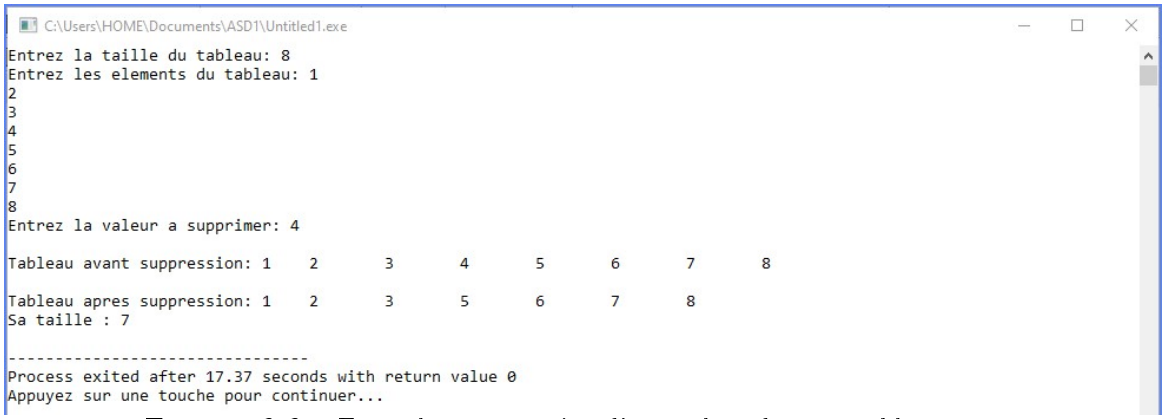


Figure 6.6 – Exemple : suppression d'une valeur dans un tableau.

6.4 Les tableaux multidimensionnels (Matrices)

On peut avoir des tableaux à plusieurs dimensions qui sont des tableaux de tableaux, dans ce cas on donne un sens à chaque dimension. Un tableau à deux dimensions est dit matrice par analogie aux matrices mathématiques, les deux dimensions représentent alors les lignes et les colonnes.

6.4.1 Déclaration du tableau

Syntaxe de déclaration en algorithmique :

Type identificateur[Taille1][Taille2] ;

Var identificateur : Tableau[1..Taille1, 1..Taille2] de Type ;

Où
Type : est le type des éléments du tableau.
identificateur : représente le nom du tableau.
Taille1 : nombre de lignes du tableau.
Taille2 : nombre de colonnes du tableau.
☞ Exemple :

```
Algorithme Exemple;
Var
T : Tableau[1..5,1..5] dentiers ;
R: Tableau[1..3,1..2] de réels;
```

Syntaxe de déclaration en C :

Type identificateur[Taille1][Taille2] ;

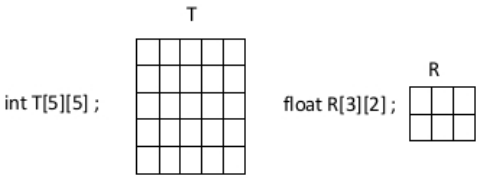


FIGURE 6.7 – Tableaux multidimensionnels.

```
int main(){
int T[5][5];
float R[3][2];
--------------
}
```

Il est possible d'initialiser le tableau dès sa déclaration en lui affectant une liste de valeurs comme suit :

identificateur $=\{\{\text{valeur}_1, valeur_2, ..., valeur_n\}, ..., \{valeur_1, valeur_2, ..., valeur_m\}\}$

☞ Exemple :

```
int tab1[2][3]={{1,2,3},{4,5,6}} ; // équivalente à int tab1[2][3]={1,2,3,4,5,6} ;
int tab2[4][6] = {{10,12,15,3},{1,2,15,10,0,3}}; /* Les valeurs spécifiées seront placées dans
    les 10 premières cases de tab rangées lignes par lignes */
```

6.4.2 Accès à un élément du tableau

L'accès à un élément du tableau se fait comme suit :

identificateur[indice$_1$][$indice_2$]

Où
identificateur représente le nom du tableau.
$indice_1$: est la position ligne de l'élément dans le tableau.
$indice_2$: est la position colonne de l'élément dans le tableau.
☞ Exemple :

```
tab[3][2] = 12 ;  \* affecte la valeur 12 à l'élément se trouvant à la quatrième ligne et troisi
    ème colonne de tab */
```

6.4.3 Lecture et écriture du tableau

Syntaxe de lecture et écriture en algorithmique :

Lire(identificateur[indice$_1$, *indice*$_2$]);

Ecrire(identificateur[indice$_1$, *indice*$_2$]);

Syntaxe en C :

scanf("format", &identificateur[indice$_1$][*indice*$_2$]);

printf("format",identificateur[indice$_1$][*indice*$_2$]);

☞ Exemple 1 : soit à lire et à écrire les éléments d'un tableau à deux dimensions (matrice) A à n lignes et m colonnes, A(n,m).

```
Algorithme Exemple;
Var i,j,n,m : entier;
A : Tableau[1..50,1..60] d'entiers ;
Début
Ecrire(Entrer le nbre de lignes et colonnes de A:);
Lire(n,m);
Ecrire(Entrer les éléments de A :);
Pour i=1 à n faire
       Pour j=1 à m faire
           Lire(A[i,j]);
Pour i=1 à n faire
      Pour j=1 à m faire
            Ecrire(A[i,j]);
Fin.
```

Programme en C :

```
#include<stdio.h>
int main()
{ int i,j, A[50][60],n,m;
// Lecture de la taille de A
printf("Le nbre de lignes et colonnes de A:" );
scanf( "%d%d", &n,&m);
// Lecture de A
printf("Entrer les elements de A :");
for(i=0;i<n;i++)
   for(j=0;j<m;j++)
      scanf("%d",&A[i][j]);
// Affichage de A
```

```
for(i=0;i<n;i++)
{ for(j=0;j<m;j++)
       printf("%d\t",A[i][j]);
printf("\n");
} }
```

☞ Exemple 2 : affichage de la table de multiplication de Pythagore (mathématicien, philosophe et astronome de la Grèce antique né en 569 av. J.-C. et mort vers 494 av. J.-C.). Notons que dans ce code la table est d'abord construite en mémoire puis affichée à l'écran.

```
#include <stdio.h>
int main()
{int Table[10][10],i,j;
// Construction de la table
for(i=0;i<10;i++)
    for(j=0;j<10;j++)
         Table[i][j] = (i+1)*(j+1);
// Affichage de la table
printf("Table de multiplication : \n");
for(i=0;i<10;i++)
   {for(j=0;j<10;j++)
         printf("%d\t",Table[i][j]);
      printf("\n");
}}
```

Résultat de l'exécution :

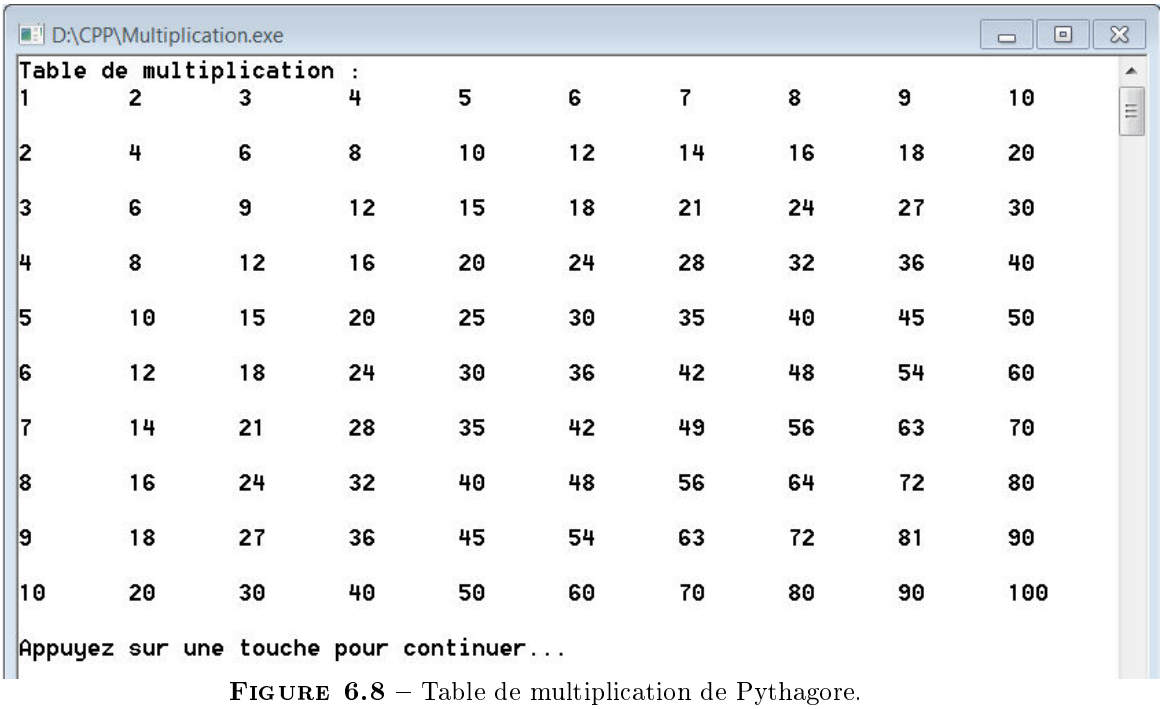


Figure 6.8 – Table de multiplication de Pythagore.

☞ Exemple 3 : Transposée d'une matrice A à n lignes et m colonnes.

$$(A^T)_{i,j} = A_{j,i} \tag{6.1}$$

Soit : A= $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ alors $A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$

a/ Matrice quelconque $n \neq m$ (généralisation du traitement pour tout types de matrices).

```c
#include <stdio.h>
#define Nblignes 50
#define Nbcolonnes 30
int main() {
 int A[Nblignes][Nbcolonnes], AT[Nbcolonnes][Nblignes], n, m,i,j;
// Lecture de la taille de A
    printf("Entrez n (lignes) et m (colonnes): ");
    scanf("%d %d", &n, &m);
    printf("Entrez les elements de A (%dx%d):\n", n, m);
    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            scanf("%d", &A[i][j]);
   // Affichage de la matrice A
    printf("Matrice A (%dx%d):\n", n, m);
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++)
            printf("%d\t", A[i][j]);
        printf("\n");
    }
// Construction de AT la matrice transposée
    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            AT[j][i] = A[i][j];
// Affichage de la transposée
    printf("Matrice transposee AT (%dx%d):\n", m, n);
    for(i = 0; i < m; i++) {
        for(j = 0; j < n; j++)
            printf("%d\t", AT[i][j]);
        printf("\n");
    }
    return 0;
}
```

Résultat de l'exécution :

```
C:\Users\HOME\Documents\ASD1\Untitled1.exe
Entrez n (lignes) et m (colonnes): 2
3
Entrez les elements de A (2x3):
1
2
3
4
5
6
Matrice A (2x3):
1       2       3
4       5       6
Matrice transposee AT (3x2):
1       4
2       5
3       6

--------------------------------
Process exited after 15.97 seconds with return value 0
Appuyez sur une touche pour continuer...
```

FIGURE 6.9 – Transposée d'une matrice quelconque.

b/ Matrice carrée d'odre n.

Soit A= $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ alors $A^T = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$

Principe : La transposée est obtenue en permutant les éléments deux à deux par rapport à la diagonale principale.

```c
#include <stdio.h>
#define Tmax 100
int main() {
int A[Tmax][Tmax],n,i,j, temp; // temp variable auxiliaire pour permuter deux variables
// Lecture de la taille
    printf("Entrez n (taille matrice carree): ");
    scanf("%d", &n);
// Lecture des éléments de A
    printf("Entrez les elements de A (%dx%d):\n", n, n);
    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            scanf("%d", &A[i][j]);
 // Affichage de la matrice A
    printf("Matrice A (%dx%d):\n", n, n);
    for(i = 0; i < n; i++) {
        for( j = 0; j < n; j++)
            printf("%d\t", A[i][j]);
        printf("\n");
    }
// Transposee: permutation par rapport a la diagonale principale
    for(i = 0; i < n; i++)
        for(j =0; j < i; j++) {  // Seulement au-dessous de la diagonale
            temp = A[i][j];
            A[i][j] = A[j][i];
            A[j][i] = temp;
        }
    printf("Matrice transposee :\n");
    for( i = 0; i < n; i++) {
        for(j = 0; j < n; j++)
            printf("%d\t", A[i][j]);
        printf("\n");
    }
    return 0;
}
```

Résultat de l'exécution :

FIGURE 6.10 – Transposée d'une matrice carrée.

☞ Exemple 4 : Lire deux matrices A et B d'entiers et afficher leur produit. La matrice A a n lignes et m colonnes A(n,m) et la matrice B a m lignes et p colonnes B(m,p).

Remarque *Le produit de deux matrices n'est réalisable que si le nombre de colonnes de la première est égal au nombre de lignes de la deuxième.*

$$A(n,m) \times B(m,p) = C(n,p) \tag{6.2}$$

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \times b_{kj} \quad avec \quad i = 1,...,n \quad et \quad j = 1,...,p \tag{6.3}$$

Soit : A(2,3)= $\begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 3 \end{pmatrix}$ et B(3,3)= $\begin{pmatrix} -1 & 2 & -2 \\ 2 & 1 & 1 \\ 2 & -2 & 1 \end{pmatrix}$ alors C(2,3) = $\begin{pmatrix} 5 & 2 & 1 \\ 6 & -1 & 0 \end{pmatrix}$

```
#include<stdio.h>
int main()
{int A[20][30], B[30][40], C[20][40],i,j,k,n,m,p;
printf("Entrer le nombre de lignes et le nombre de colonnes de la matrice A:");
scanf("%d%d",&n,&m);
// Lecture de A
printf("Entrer les elements de la matrice A:\n");
for(i=0;i<n;i++)
 for(j=0;j<m;j++)
      scanf("%d",&A[i][j]);
printf("Entrer le nombre de lignes et le nombre de colonnes de la matrice B:");
scanf("%d%d",&m,&p);
// Lecture de B
printf("Entrer les elements la matrice B:\n");
for(i=0;i<m;i++)
 for(j=0;j<p;j++)
      scanf("%d",&B[i][j]);
// Calcul du produit A X B
for(i=0;i<n;i++)
  for(j=0;j<p;j++)
    {C[i][j] = 0; // Initialisation de la somme cumulée à 0
     for(k=0;k<m;k++)
            C[i][j] = C[i][j]+A[i][k]*B[k][j];
    }
```

```
// Affichage de C
printf("la matrice C = A X B est:\n");
for(i=0;i<n;i++)
{for(j=0;j<p;j++)
      printf("%d\t",C[i][j]);
  printf("\n");
 }
return 0;
}
```

Résultat de l'exécution :

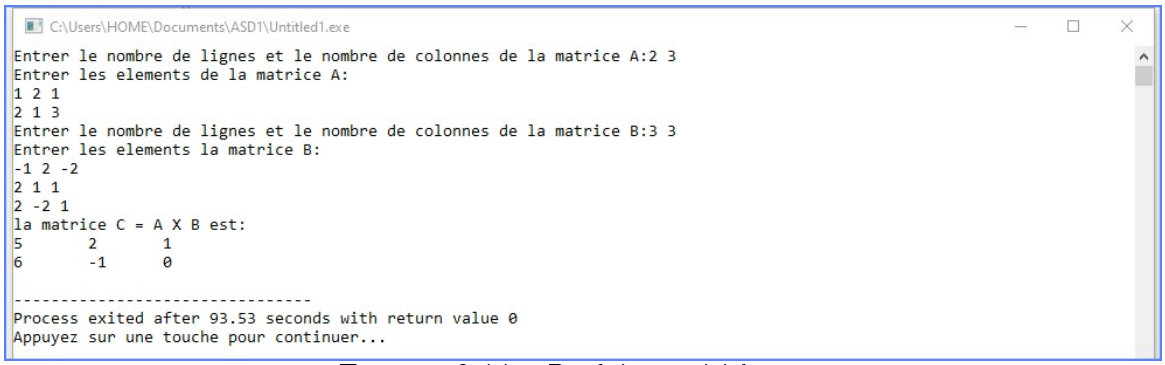


FIGURE 6.11 – Produit matriciel.

6.5 Les chaînes de caractères

En C il n'existe pas de type prédéfini pour les chaînes de caractères.
Les chaînes de caractères sont des séquences de caractères représentées par un tableau unidimensionnel de caractères.
Le nombre de caractères composant une chaîne de caractères est la longueur de cette chaîne.
La représentation en mémoire d'une chaîne de caractères est terminée par le symbole `\0` (non affichable). Pour cela, une chaîne de caractères de longueur n nécessite $(n+1)$ octets (bytes) puisque chaque caractère occupe un octet.
Une chaîne de caractères peut être déclarée de deux manières :

- En utilisant un tableau de caractères dont la taille est fixée à l'avance (allocation statique) :
  Syntaxe de déclaration :

  char identificateur[longueur] ;

  Où
  identificateur : représente le nom de la chaîne de caractères. Il donne l'adresse en mémoire du premier caractère de la chaîne.
  longueur : est le nombre de caractères maximal de la chaîne, le caractère délimitant la chaîne en mémoire `\0` n'est pas compté.
  ☞ Exemple :

```
char nom[20], prenom[40], adresse[100];
```

Les constantes chaînes de caractères sont indiquées entre guillemets "".
Syntaxe de l'accès aux éléments d'une chaîne de caractères :

char identificateur[longueur] = constante ;

Où constante est une constante chaîne de caractères.
☞ Exemple :

```
char chaine[5] = "Hello"; /* Le compilateur réserve un espace de 6 octets (5+1) pour
    stocker la chaîne */
char chaine[] = "Hello";  // C'est exactement le même effet que la précédente
char mois[3] = "JAN" ;/* Le compilateur réserve un espace de 4 octets (3+1) pour stocker la
     chaîne */
```

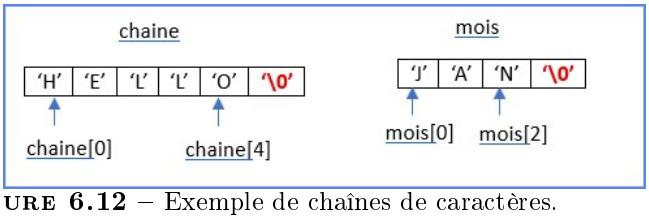


Figure 6.12 – Exemple de chaînes de caractères.

- En utilisant un pointeur sur les caractères de la chaîne, la longueur de la chaîne est a priori inconnue :
  Syntaxe de déclaration :

  char *pointeur = constante ;

  Où
  pointeur : représente l'adresse du premier caractère de la constante chaîne de caractères.
  ☞ Exemple :

```
char *ptr = "bonjour" ; /* ptr est une variable pointeur sur le premier caractère 'b' de la
     chaîne "bonjour" */
```

Il est possible d'allouer dynamiquement de l'espace mémoire pour une chaîne de caractères, comme le montre l'exemple ci-dessous :

```
char *chaine1;
int l=10 ;
chaine =(char*) malloc(longueur+1) ; /* La chaîne peut contenir 10 caractères tout au plus,
     le 11ème caractère est le délimiteur de la chaîne */
```

En tenant compte de l'ordre léxicographique des caractères, on peut contrôler le type du caractère (chiffre, majuscule, minuscule).
La précédence des caractères dépend du code utilisé. Pour le code ASCII, on peut constater l'ordre suivant : . . . ,0,1,2, ... ,9, . . . ,A,B,C, ... ,Z, . . . ,a,b,c, ... ,z, . . .
Exemple :

```
char c;
car = getchar();
if (car>='0' && car<='9') printf("Le caractere %c est un chiffre\n", car);
if (car>='A' && car<='Z')  printf("Le caractere  %c  est majuscule\n", car) ;
if (car>='a' && car<='z') printf("Le caractere %c est minuscule\n", car);
```

6.5.1 Tableau de chaînes de caractères

Un tableau de chaînes de caractères est un tableau à deux dimensions du type char, où chaque ligne contient une chaîne de caractères. Syntaxe de déclaration :

char identificateur[Taille][Longueur] ;

Où :
Taille : est le nombre maximal de chaines de caractères dans le tableau. Longueur : est la longueur maximale d'une chaîne de caractères. ☞ Exemple : déclaration et initialisation d'une chaîne de caractères.

```
char  couleur[4][7] = {"blanc", "bleu"  , "rouge" , "noir"} ;
```

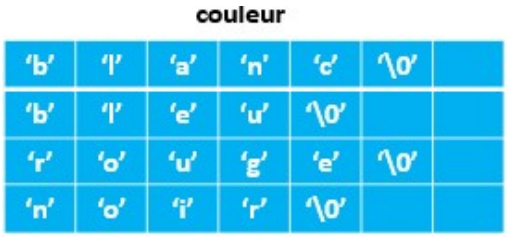


Figure 6.13 – Tableau de chaînes de caractères. Les chaînes de caractères sont rangées ligne par ligne.

6.5.2 Fonctions manipulant les chaînes de caractères

Il existe de nombreuses fonctions permettant de manipuler les chaînes de caractères. Elles se trouvent dans la librairie standard **string.h** du C ou **cstring.h** du C++, **stdio.h** et **strlib.h**. Donnons quelques fonctions implémentées dans les librairies citées :

Quelques fonctions de string.h :

- strlen(s) : longueur de la chaîne s, elle retourne 0 si la chaîne est vide.
  Exemple : strlen("Bonjour")=7 et strlen("")=0.

- strcpy(s,t) : copie la chaîne t vers la chaîne s.
  Exemple : strcpy("sisi","alibaba") $\Longrightarrow$ "alibaba".
  strcpy("alibaba","sisi") $\Longrightarrow$ "sisiaba".

- strcat(s,t) : ajoute la chaîne t à la fin de la chaîne s, le résultat sera dans la chaîne s.
  Exemple : strcat("sisi","alibaba") $\Longrightarrow$ "sisialibaba".
- strcmp(s,t) : compare les chaîne s et t lexicographiquement (relativement au code AS–CII) et fournit le résultat suivant :
  -1 si s précède t.
  0 si s et t sont égales.
  1 si s suit t.
  Exemple : strcmp("sisi","alibaba") = 1, strcmp("alibaba","sisi")=-1 et strcmp("alibaba","alibaba")
- strncpy(s, t, n) : copie au plus n caractères de t vers s.
  Exemple : strncpy("alibaba,"sisi",2) "siibaba".
- strncat(s, t, n) : ajoute au plus n caractères de t à la fin de s.
  Exemple : strncat ("alibaba","sisi",2) "alibabasi"
- strchr(s,c) : cherche le caractère c dans la chaîne de caractère s et retourne **NULL** si c ne s'y trouve pas, dans le cas contraire, elle retourne la sous-chaîne qui commence par c.
  Exemple :

```
#include<stdio.h>
#include<string.h>
int main(){
char chaine[]="Salut tout le monde, soyez les bienvenus!", c1='t', c2='x';
printf("%s\n",strchr(chaine,c1));
printf("%s",strchr(chaine,c2));
}
```

Exécution :

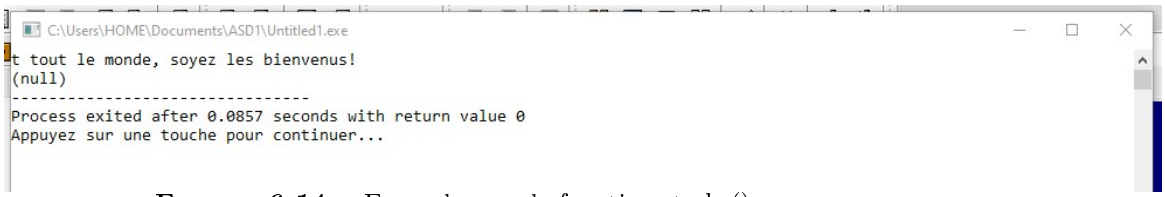


Figure 6.14 – Exemple avec la fonction strchr().

Quelques fonctions de **stdio.h** relatives aux chaînes de caractères :
- **puts**(s) : écrit la chaîne s sur la sortie standard **stdout** et provoque un retour à la ligne.
- **gets**(s) : lit une ligne de caractères de l'entrée standard **stdin** et la copie à l'adresse indiquée par s. Le retour à la ligne est remplacé par le caractère `\0`.

La saisie d'une chaîne de caractère par la fonction **scanf()** ne nécessite pas l'opérateur « & », puisque c'est un tableau (dont le nom est implicitement une adresse).

☞ Exemple :

```
int main()
{char chaine[100] ;
printf("Taper une chaîne : ") ;
scanf("%s",chaine) ;// pas besoin de & car c'est une chaine de caractères
puts(chaine);
}
```

Quelques fonctions de **stdlib.h** relatives aux conversion de chaînes de caractères :

- **atoi**(s) : retourne la valeur numérique de la chaîne s en type **int**. La fonction retourne la valeur 0 si la chaîne de caractères ne contient pas une représentation de valeur numérique.
  Exemple : atoi("alibaba")=0, atoi("ali123") = 0, atoi("123ali")=123 et atoi("123") = 123.
- **atol**(s) : retourne la valeur numérique de la chaîne s en type **long**.
- **atof**(s) : retourne la valeur numérique de la chaîne s en type **double**.
- itoa(nbre,s,b) : retourne une chaine s correspondant à la conversion du nombre entier nbre dans la base de numération b.
- ltoa(nbre,s,b) : idem pour le nombre nbre qui est de type long. item ultoa(nbre,s,b) : idem pour le nombre nbre qui est de type unsigned long.

  ☞ Exemple : conversion d'un nombre :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{int n;
  char s[50];
  printf ("Enter le nombre : ");
  scanf ("%d",&n);
  itoa (n,s,10);       // Conversion en décimal
  printf ("Le nombre %d en decimal : %s\n",n,s);
  itoa (n,s,2);      // Conversion en binaire
  printf ("Le nombre %d en binaire : %s\n",n,s);
  itoa (n,s,16);  // Conversion en héxadécimal
  printf ("Le nombre %d en hexadecimal : %s\n",n,s);
    return 0;
}
```

Résultat de l'exécution :

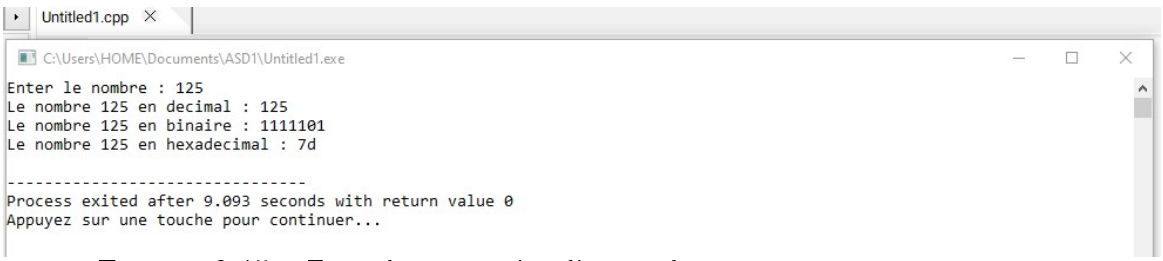


Figure 6.15 – Exemple : conversion d'un nombre.

A ne pas faire :

- Affectation entre deux variables chaînes de caractères, l'opérateur (=) n'est pas permis pour cela.
- Utilisation de l'opérateur (+) pour concaténer des chaînes de caractères, ce n'est valide, la fonction **strcat** existe justement pour ça.
- Comparaison de deux chaînes de caractères à l'aide de l'opérateur (==), ce n'est pas valide, la fonction **strcmp** existe justement pour ça.
- Considérer une variable de type caractère comme étant une variable de type chaîne de caractères, ce n'est pas la même chose. Ainsi, le caractère 'x' et la chaîne "x" sont de types incompatibles et sont représentés en mémoire différemment.
- la saisie d'une variable de type chaîne de caractère par l'instruction **scanf** en utilisation le caractère **&**. Ce dernier est omis puisqu'une chaîne de caractère est un tableau.

6.5.3 Quelques exemples utilisant les chaînes de caractères

Exercice 14 *Saisir une chaîne de caractères et afficher sa longueur sans utiliser la librairie standard et en utilisant la librairie standard <tring.h>.*

```
#include<stdio.h>
#include<string.h>
int main(){
char chaine[80];
int i=0;
// Lecture d'une chaine de caractères
printf("Saisir une chaine : ");
gets(chaine);
while(chaine[i]!='\0')
   i++;
printf("Longueur calculee de la chaine %s = %d\n",chaine,i);
printf("Longueur retournee par strlen de la chaine %s = %d\n",chaine,(int) strlen(
    chaine));
}
```

Résultat de l'exécution :

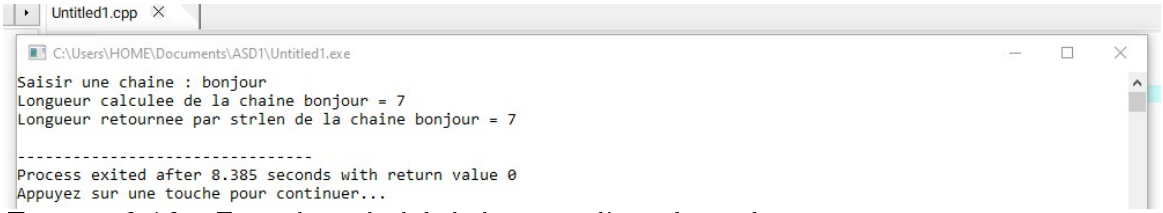


FIGURE 6.16 – Exemple : calcul de la longueur d'une chaîne de caractères.

Exercice 15 *Saisir un mot/phrase ou une phrase et afficher son mot/phrase miroir.*

```
#include <stdio.h>
#include <string.h>
#define taille 256
int main(){
char mot[taille], miroir[taille]; // Réservation statique des tableaux mot et miroir
int i,n ;
printf("Entrer le mot : ");
gets(mot);
printf("Le mot a inverser est : ");
puts(mot);
n = strlen(mot);  // Longueur du mot
for(i=0;i<n;i++)
     miroir[i] = mot[n-i-1]  ;
 printf("\nLe mot miroir est : %s\n ",miroir);
 return 0;
}
```

Résultat de l'exécution :

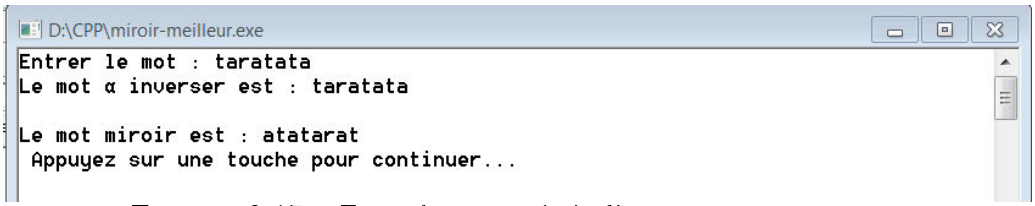


Figure 6.17 – Exemple : mot miroir d'un mot.

Exercice 16 *Compter le nombre de mots dans une phrase saisie au clavier. Pour simplifier le code, on suppose que le seul séparateur entre deux mots est le caractère blanc (' ') sans duplication (un seul espace blanc).*

```
#include <stdio.h>
#include <string.h>
#define taille 256
int main()
{char phrase[taille];
int i=0,nbmots=0;
printf("Tapez du texte : ");
gets(phrase);
if(strlen(phrase)==0 || (strlen(phrase)==1 && phrase[0]==' ')) /* Phrase vide ou limité
    e à un seul blanc */
printf("\nLe nombre de mots est : %d\n",nbmots);
else {
   while(phrase[i]!='\0'){
     if(phrase[i]==' ')
         nbmots++;
     i++; }
 printf("\nLe nombre de mots est : %d\n ",nbmots+1);
}}
```

Résultat de l'exécution :

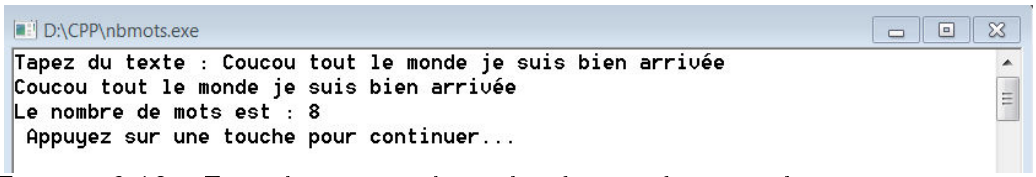


FIGURE 6.18 – Exemple : compter le nombre de mots dans une phrase.

Exercice 17 *Compter le nombre de voyelles dans une chaîne de caractères.*

```
#include<stdio.h>
#include<string.h>
int main()
{char mot[256];
int i=0, nbvoyelles=0;
printf("Saisir le mot\n");
gets(mot);
while(mot[i]!='\0')
{ if(mot[i]=='a' || mot[i]=='A' || mot[i]=='e' || mot[i]=='E' || mot[i]=='u' ||
 mot[i]=='U' || mot[i]=='i' || mot[i]=='I' || mot[i]=='o' || mot[i]=='O'|| mot[i]=='y'
    || mot[i]=='Y')
    nbvoyelles++;
 i++;
}
printf("Le nombre de voyelles dans le mot : %s est: %d",mot,nbvoyelles);
}
```

Ou bien

```
#include<stdio.h>
#include<string.h>
int main()
{char mot[256], voyelle[]="AaEeIiUuOoYy";
 int i=0,nbvoyelles=0;
 printf("Entrez le mot : ");
 gets(mot);
 while(mot[i]!='\0')
  {if(strchr(voyelle,mot[i]) != NULL)
    nbvoyelles++;
  i++;
  }
    printf("Le nombre de voyelles dans le mot %s est: %d",mot,nbvoyelles);
return 0;
}
```

Résultat de l'exécution :

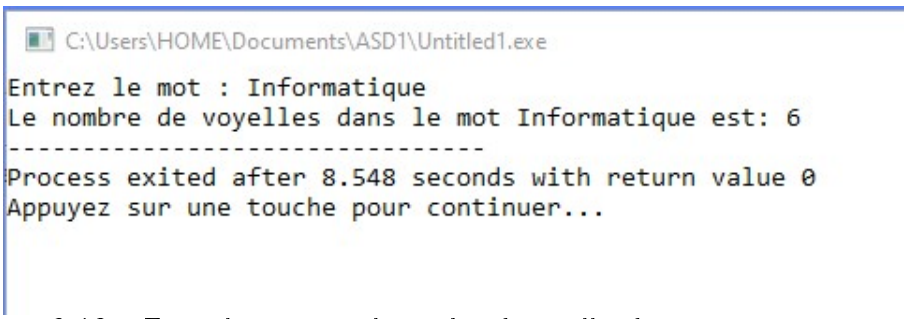


FIGURE 6.19 – Exemple : compter le nombre de voyelles dans un mot.

Exercice 18 *On considère un tableau comportant une séquence de chaînes de caractères. Ecrire un algorithme qui affiche et compte toutes les chaînes qui contiennent au moins deux occurrences du caractère 'A' quelquesoit la casse du caractère (pas de distinction entre une majuscule et une minuscule). Exemple : contenu du tableau : "Informatique", "Institut", "Avocat", "TARATATA". Résultats affichés : Avocat TARATATA.*

```c
#include <stdio.h>
#include<string.h>
#define Taille 50
int main()
{char T[Taille][256];
 int i,k,nboc,n;
 printf("Entrez le nombre de chaines : ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {printf("Entrez la chaine : ");
  gets(T[i]); // ou bien   scanf("%s",T[i]);
 }
// Affichage du tableau T de chaines de caractères
printf("Les chaines de caracteres a traiter sont : \n");
for(i=0;i<n;i++)
   puts(T[i]);
// Recherche du caractère A et calcul de son nombre d'occurrence
 for(i=0;i<n;i++)
 {nboc=0; k=0;
 while(T[i][k]!='\0')
  {if(T[i][k]=='A' || T[i][k]=='a')  nboc++;
  k++; }
   // Affichage du résultat
  if(nboc>=2)
    printf("La chaine  %s contient  : %d  A/a\n",T[i],nboc); }
return 0;
}
```

Résultat de l'exécution :

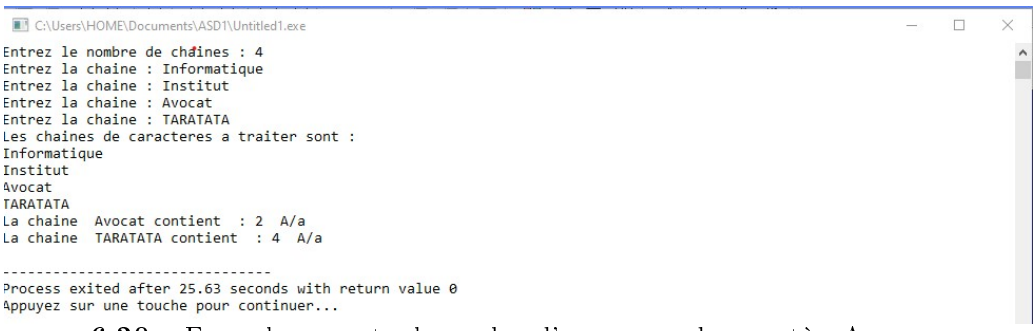


Figure 6.20 – Exemple : compter le nombre d'occurrence du caractère A.

CHAPITRE 7
LES TYPES PERSONNALISÉS

Sommaire

Figures

7.1 Introduction

En plus des types de base dits aussi primitifs qui sont prédéfinis dans le langage (int, char, double, bool), la programmation peut nécessiter des types personnalisés représentant des entités plus complexes répondant au besoin du problème à résoudre.

7.2 Énumérations

Le langage C nous permet de manipuler des types ordinaux avec le concept énumération. Un type énuméré est un ensemble fini d'éléments lesquels sont énumérés dans un certain ordre (mois de l'année, notes de musique, couleurs d'un drapeau, couleurs de l'arc en ciel, marques de voitures,...etc). Les valeurs du type énumération comme des constantes entières de type « int », elles sont converties dans l'ordre où elles apparaissent à partir de l'entier 0.
Syntaxe de déclaration :

enum identificateur $\{\text{constante}_1, constante_2, ..., constante_n\}$

Où
identificateur : représente le nom du type.
$constante_1, constante_2, ..., constant_n$: représentent les valeurs que peut prendre une variable de ce type.

☞ Exemple 1 : déclaration de quelques types énumération

```
enum jour{Dim,Lun,Mar,Mer,Jeu,Ven,Sam};
enum drapeau{vert,blanc,rouge};
enum moyen_voyage{voiture,camion,bus,train,avion,bateau};
.......................
enum jour j;    // la variable j est de type énumération jour
.......................
```

Le compilateur C associe automatiquement des valeurs entières à chaque constante du type énumération, comme le montre l'exemple suivant :
☞ Exemple 2 : représentation des constantes du type énumération.

```
#include <stdio.h>
enum couleur_claire{blanc,gris,bleu_ciel,rose,vert_eau};
int main()
{ enum couleur_claire clr=blanc;
  printf(" \nLa première couleur a la valeur : %d\n", (int)clr);
  printf(" \nLa couleur bleu ciel a la valeur  : %d\n", bleu_ciel);
}
```

Résultat de l'exécution :

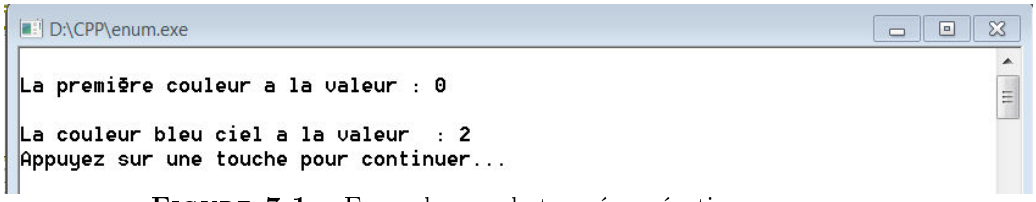


FIGURE 7.1 – Exemple avec le type énumération.

Il est possible de forcer les valeurs initialement affectées par le compilateur C, soit par exemple en modifiant les valeurs des constantes dans l'exemple précédent :

```
#include <stdio.h>
enum couleur\_claire\{blanc=4,gris=6,bleu\_ciel=8,rose=10,vert\_eau=12\};
int main()
{ enum couleur_claire clr=blanc;
  printf(" \nLa première couleur a la valeur : %d\n", (int)clr);
  printf(" \nLa couleur bleu ciel a la valeur  : %d\n", bleu_ciel);
}
```

Résultat de l'éxécution :

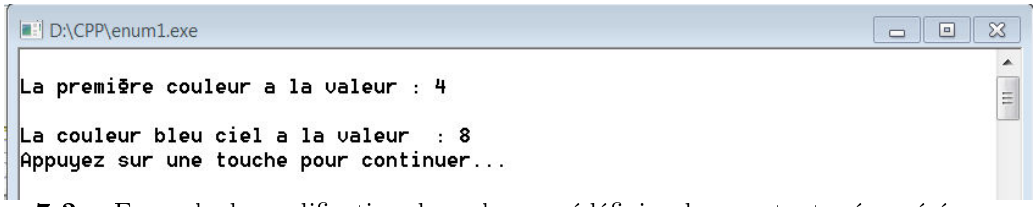


FIGURE 7.2 – Exemple de modification des valeurs prédéfinies des constantes énumérées.

Il est possible d'affecter des valeurs à certaines constantes pas forcément à toutes les constantes, comme nous pouvons le montrer en utilisant le même exemple :

```
#include <stdio.h>
enum couleur_claire{blanc,gris=50,bleu_ciel,rose=70,vert_eau};
int main()
{
printf(" \nblanc= %d\n", blanc);
printf(" \nbleu_ciel= %d\n", bleu_ciel);
printf(" \nvert_eau= %d\n", vert_eau);
}
```

Résultat de l'exécution :

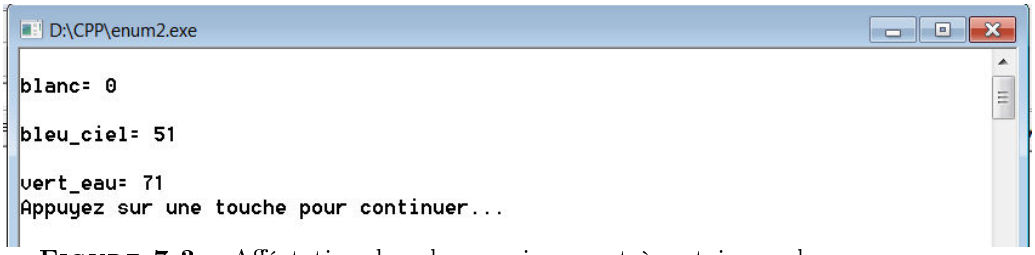


FIGURE 7.3 – Affectation de valeurs uniquement à certaines valeurs.

Il est possible de renommer le type énumération en utilisant le mot-clé « **typedef** » pour alléger sa manipulation comme suit :

enum note_musicale{do,ré,mi,fa,sol,la,si} ;
typedef *note_musicale* nm ;
. . . . . .
nm note ;

La variable note est du type nm qui est une abréviation du type énumération $note_musicale$.

7.3 Enregistrements (structures)

Le langage C/C++ nous offre une autre possibilité que les tableaux pour regrouper un ensemble de données à travers les structures, qui à la différence du type tableau, les données qu'elles représentent ne sont pas nécessairement de même type. Le type structure (**struct**) représente donc une collection de champs de types divers (types de base, tableaux, pointeurs et structures déclarées au préalable).

7.3.1 Déclaration d'une structure

Pour définir un type structure Etudiant, composé de quatre champs : nom, prenom, code et date de naissance, il suffit d'écrire les instructions suivantes :

```
/7 Définition du type struct date de naissance
struct Date{
int jour, mois, an;
};
```

Définition du type structure Etudiant :

```
// Définition du type struct Etudiant
struct Etudiant{
char nom[30], prenom[25];
int code ;
Date dns;
};
```

```
/*Déclaration d'un tableau E dont les valeurs sont de type struct Etudiant */ \\
struct Etudiant E[100];
```

On peut omettre le nom de la structure pour déclarer une variable comme suit :

```
// Définition de la structure
struct{
char nom[30], prenom[25];
int code ;
Date dns;
} E[100];// Déclaration d'un tableau E dont les valeurs sont de type struct
```

Déclaration d'une structure Livre :

```
struct Livre{
   int   cote;
   char  titre[50];
   char  auteur[50];
   char  resume[200];
};
```

Déclaration d'une structure Point où chaque variable de ce type est caractérisée par une abscisse x et une ordonnée y :

```
struct Point{
float x,y ;
};
```

La déclaration de variables structures est plus aisée si la structure est identifiée par un nom en utilisant le mot-clé **typedef**, comme suit :
typedef struct{
.....
}Personne ;
Personne p1,p2 ; le nom de type est simplement Personne et non pas struct Personne.

7.3.2 Initialisation d'une structure

L'initialisation d'une structure peut se faire de la même manière que l'initialisation d'un tableau. L'initialisation d'une variable de type structure Point se fait comme suit :

```
struct Point{
float x,y ;
};
struct Point p = {12.5,2.3}  /* la valeur 12.5 est l'abscisse et 2.3 est l'ordonnée du
    point p */
```

Initialisation des membres d'une structure lors de la déclaration de la structure :

Lorsqu'un type structure est déclaré, aucune mémoire n'est allouée. La mémoire est allouée uniquement lorsque des variables de ce type sont créées.

☞ Exemple : cette déclaration n'est pas permise.

```
struct Point{
float x=0.5,y=12 ; // L'erreur est signalée au moment de la compilation
};
```

7.3.3 Opérations sur les structures

Accès aux champs d'une structure : l'accès à un champ d'une variable de type structure ce fait à l'aide du « . » comme suit :

variable.champ

☞ Exemple :

```
E[10]. nom = "William Henry" ;
          E[10]. prenom = "Gates" ;
          E[10].dns.jour = 28 ;
               E[10].dns.mois = 10 ;
               E[10].dns.an = 1955 ;
```

Pour accéder à un membre de la structure de type pointeur c'est plutôt le symbole « -> » qui est utilisé.

Affectation de structures : il est possible d'affecter une variable de type structure à une autre, comme suit :

```
struct{
char nom[20];
char prenom[30];
int age;
}Personne;
struct Personne p1,p2;
.....................
p1 = p2;
.....................
```

Pointeurs vers une structure : il est possible de déclarer une variable de type pointeur vers une structure comme suit :

```
struct Personne{
char nom[20];
char prenom[30];
int age;
};
struct Personne *p;
struct Personne personne;
.....................
p = &personne;
.....................
printf("%s\t%s\t%d\n", p->nom, p->prenom, p->age);
```

Comparaison de structures :
Aucune comparaison n'est possible entre deux structures. Les opérateurs (==) et (!=) ne peuvent être employés.

Exercice 19 *Evaluation d'un polynôme composé de plusieurs termes et chaque terme est une structure.*

```
#include <stdio.h>
#include <math.h>
#define Max_termes 10
typedef struct
{ int coef;
  int degre;
} Terme;
Terme p[Max_termes];
int main(){
int i,n;
float x , valeur=0;
// Lecture du degré, coefficients du polynôme et de x
printf("Nombre de termes du polynome : ");
```

```
scanf("%d",&n);
printf("Entrez la valeur de x : ");
scanf("%f",&x);
  for(i=0;i<n;i++)
    {printf("Entrez le coefficient a(%d) : ", i);
     scanf("%d",&p[i].coef);
     printf("Entrez le degre(%d) : ", i);
     scanf("%d",&p[i].degre);
     }
// Calcul dela valeur du polynôme
for(i=0;i<n;i++)
   valeur=valeur+p[i].coef*pow(x,p[i].degre);
// Affichage du résultat
printf("Valeur du polynome = %3.2f\n",valeur);
}
```

Résultat de l'exécution :

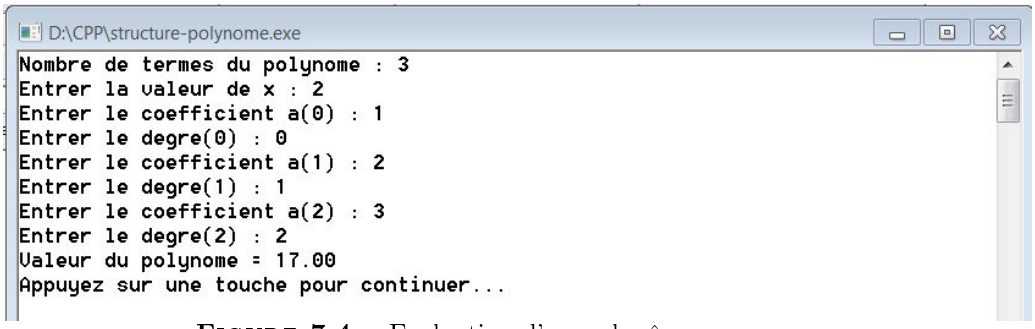


FIGURE 7.4 – Evaluation d'un polynôme.

Exercice 20 *Lire 10 points avec une structure point ayant un nom à un seul caractère généré automatiquement à partir de la lettre 'A', une abscisse x et une ordonnée y. le code affiche le nombre de points situés sur l'axe des x et le nombre de points situés sur l'axe des y. Il les affiche avec le nom et leurs coordonées.*

```
#include <stdio.h>
struct point {
    char nom;
    float x;
    float y;
};

int main() {
struct point points[10];
int i, nbx = 0;  // Nombre de points sur l'axe des x (y = 0)
int nby = 0;  // Nombre de points sur l'axe des y (x = 0)
// Lecture des 26 points
printf("Entrez les coordonnees des 10 points :\n");
for (i = 0; i < 10; i++) {
   points[i].nom = 'A' + i;  // Génération automatique du nom
   printf("Point %c - x et y : ", points[i].nom);
   scanf("%f %f", &points[i].x, &points[i].y);
  }
```

```
// Comptage et identification des points
printf("\n--- Points sur l'axe des X (y = 0) ---\n");
for (i = 0; i < 10; i++)
   if (points[i].y == 0) {
      printf("Point %c : (%.2f, %.2f)\n",
      points[i].nom, points[i].x, points[i].y);
            nbx++;
     }
printf("Nombre de points sur l'axe des X : %d\n", nbx);
printf("\n--- Points sur l'axe des Y (x = 0) ---\n");
for (i = 0; i < 10; i++)
  if (points[i].x == 0) {
            printf("Point %c : (%.2f, %.2f)\n",
            points[i].nom, points[i].x, points[i].y);
            nby++;
        }
   printf("Nombre de points sur l'axe des Y : %d\n", nby);
}
```

Résultat de l'exécution :

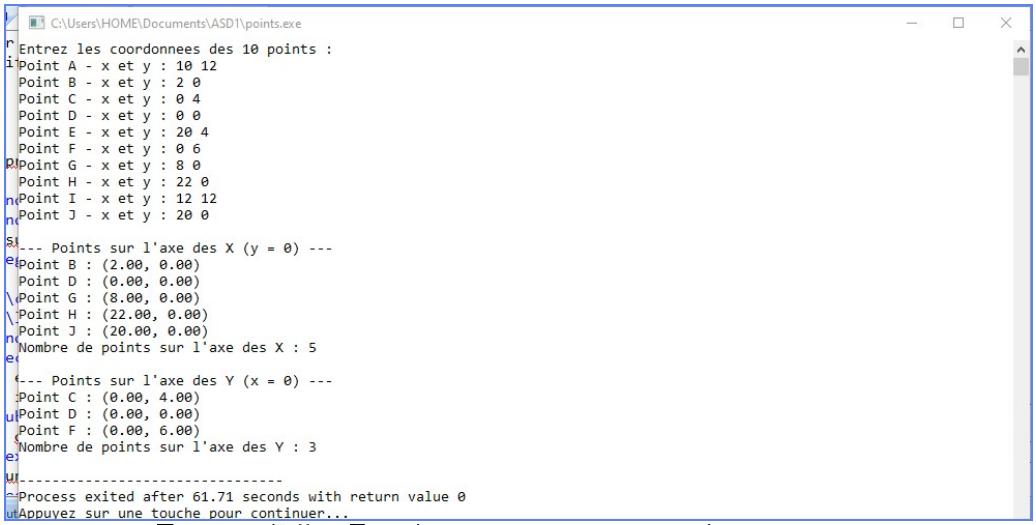


Figure 7.5 – Exercice avec une structure point.

7.4 Autres possibilités de définition de type

Il est aussi possible de définir un nouveau type analogue au type structure mais qui ne peut contenir un seul de ses membre à la fois en un instant donné, c'est le type union. Tout omme le type structure, au moment de la déclaration du type union aucune mémoire n'est allouée.

7.4.1 Déclaration d'une union

La décaration du type union est identique à celle d'une structure avec juste la substitution du mot-clé **struct** par le mot clé **union**. La taille de l'union correspond à la taille du plus grand type stocké puisque ce type ne mémorise qu'un seul membre de l'union à la fois, comme le montre l'exemple qui suit.

```c
/* Définition du type union machin */
#include<stdio.h>
union machin_union{
char code[10];
int jour;
float note;
double moyenne;
};
struct machin_struct{
char code[10];
int jour;
float note;
double moyenne;
};
int main()
{
printf("\nNombre d'octets necessaires pour l'union machin_union = %d\n", sizeof(
    machin_union));
printf("\nNombre d'octets necessaires pour la structure machin_struct = %d\n", sizeof(
    machin_struct));
}
```

Résultat de l'exécution :

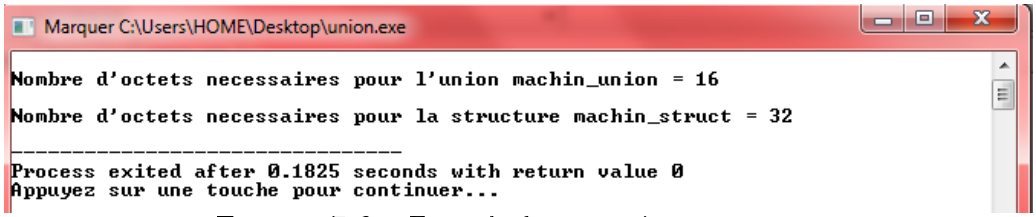


FIGURE 7.6 – Exemple de type union.

7.4.2 Accès aux membres d'une union

Pour accéder aux membres d'une union, le symbole point « . » est utilisé et pour accèder aux variables de type pointeur c'est le symbole « -> » qui est utilisé, tout comme le type structure.

```c
#include<stdio.h>
union machin{
char code[10];
int jour;
float note;
double moyenne;
};
int main()
{union machin m;
m.jour=12;
printf("\nJour avant d'initialiser note'= %d\n", m.jour);

m.note=13.5;
```

```
printf("\n\nJour apres initialisation de note = %d\n", m.jour);
printf("\nNote avant d'initialiser moyenne = %f\n", m.note);
m.moyenne=10;

printf("\n\nJour apres initialisation de moyenne = %d\n", m.jour);
printf("\nNote apres initialisation de moyenne = %f\n", m.note);
printf("\nMoyenne = %lf\n", m.moyenne);
}
```

Résultat de l'exécution :

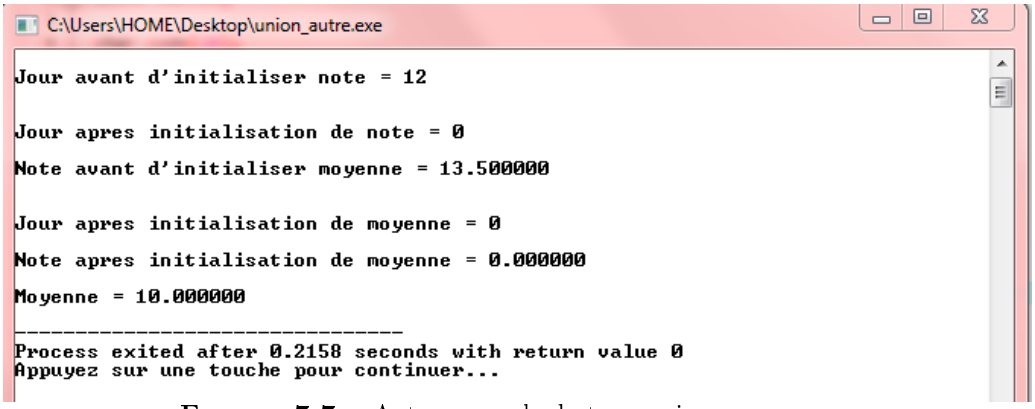


Figure 7.7 – Autre exemple de type union.

Le résultat montre bien qu'il n'est possible d'accéder qu'à un seul membre d'une variable de type union à un instant donné alors que dans le cas d'une structure tous les membres son accessibles à un même instant puisque la mémoire est allouée à tous les membres d'une variable.

BIBLIOGRAPHIE

.

Amblard, P., J. Fernandez, F. Lagnier, F. Maraninchi, P. Sicard, and P. Waille
2000. *Architectures Logicielles et Matérielles.* Dunod.

Arora, D.
Derniére mise á jour : 12 november 2020. "comprendre la complexit é du temps avec des exemples simples". https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/.

Beauquier, D., J. Berstel, and P. Chrenne
1992. *Éléments d'Algorithmique.* Masson.

Berthet, D. and V. Labatut
2014a. *Algorithmique & programmation en langage C - vol.1 : Supports de cours. Licence. Algorithmique et Programmation.* Istanbul, Turquie.

Berthet, D. and V. Labatut
2014b. *Algorithmique & programmation en langage C - vol.2 : Supports de cours. Licence. Algorithmique et Programmation.* Istanbul, Turquie.

Berthet, D. and V. Labatut
2014c. *Algorithmique & programmation en langage C - vol.3 : Corrigés de travaux pratiques. Licence. Algorithmique et Programmation.* Istanbul, Turquie.

Burks, A., H. Goldstine, and J. V. Neumann
1963. *Preliminary discussion of the logical design of an electronic computing instrument.* 15

Canteaut, A.
. "programmation en langage C". https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/cours.pdf.

Champin, P. A.
. "Listes chainées". http://liris.cnrs.fr/pierre-antoine.champin/enseignement/algo/listes_chainees/.

Cordier, A.
2015. "activité d'introduction á l'algorithmique". https://perso.liris.cnrs.fr/amelie.cordier/teaching/algo/aut2015/IntroAlAlgo.pdf. 13

Cormen, T.
2010. *Algorithmique*. Dunod.

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

Cormen, T. H., C. E. Leiserson, and R. L. Rivest
2010. *Algorithmique - Cours avec 957 exercices et 158 problémes*, 3éme edition. Dunod.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein
2009. *Introduction to Algorithms*, 3rd edition. The MIT Press.

Delannoy, C.
1991. *Apprendre á programmer en Turbo C*. EYROLLES.

Duret-Lutz, A.
Date de publication : 7 novembre 2014. "algorithmique". https://www.coursehero.com/file/69224304/algopdf/.

Edouard, J. M. C.
Date de publication : 07 avril 2008 Derni mise ur : 09 janvier 2010. "manipulation des fichiers en c". https://perso.esiee.fr/~landschm/IN3S03/fichiers.pdf.

en ligne.net, A.
. Une petite histoire de l'informatique. https://www.apprendre-en-ligne.net/info/histoire/histoire.pdf.

FabLab
. Petite histoire de linformatique. https://www.bm-lyon.fr/nos-blogs/fablab/l-idee-du-mois/article/petite-histoire-de-l-informatique.

Froidevaux, C., M. Gaudel, and M. Soria
1990. *Types de donn et algorithmes*. McGraw-Hill.

Geerarets, G.
Année Académique 2008–2009 (2^e édition). "Notes du cours FS/1/6584. Année préparatoire au Master en Informatique". https://perso.esiee.fr/~landschm/IN3S03/fichiers.pdf.

Griffiths, M.
1992. *Algorithmique et programmation*. Hes.

Kernighan, B. and D. Richie
1988. *The C programming language*, 2nd edition. 38

Knuth, D.
1998. *The Art Of Computer Programming(TAOCP)*, 3 edition. Sorting and Searching. Addison-Wesley.

Lucas, E.

1892. *Récréations mathématiques*. Librairie Albert Blanchard, tome3. 27

Malgouyres, R., R. Zrour, and F. Feschet

2011. *Initiation á l'algorithmique et á la programmation en C : cours avec 129 exercices corrigés*, 2éme edition. Dunod, Paris. ISBN : 978-2-10-055703-5.

Posamentier, A. and I. Lehmann

2007. *The Fabulous Fibonacci Numbers*, 1 edition. Amherst, N.Y : Prometheus Books.

Quercia, M.

2002. *Algorithmique. Cours complet, exercices et probls résolus, travaux pratiques*. Vuibert.

Raaf, H.

2020. *Algorithmique et Structures de Données 1*. Notes de cours.

Régner, P.

IRIT - Université Paul Sabatier, 2010-2011. "histoire de l'informatique". https://www.irit.fr/~Pierre.Regnier/SitePierre/Enseignement_files/CoursHistoire.pdf.

Renault, E.

Date de publication : 5 octobre 2016. "Introduction á l'algorithmique : notes de cours". https://www.lrde.epita.fr/~renault/teaching/algo/cours.pdf.

Techno-Science.Net

. Histoire de l'informatique - définition. https://www.techno-science.net/glossaire-definition/Histoire-de-l-informatique.html.