Université Ibn khaldoun de Tiaret

Département des Sciences et des technologies

Polycopie de cours du module :

Informatique 1

1ière année tronc commun-LMD Sciences et Technologies

Abdelfettah MAATOUG
a_maatoug@esi.dz

Expertisé par :
Mr KHAROUBI Sahraoui, MCA, sahraoui.kharroubi@univ-tiaret.dz .
Mr MOSTEFAOUI Sid-ahmed Mokhtar, MCA, s_mostefaoui@esi.dz.

Année universitaire : 2022/2023

Sommaire

Chapitre III : Les variables Indicées

Avant-propos

L'objectif de la matière est de permettre aux étudiants d'apprendre à programmer avec un langage évolué (Fortran, Pascal ou C). Le choix du langage est laissé à l'appréciation de chaque établissement. La notion d'algorithme doit être prise en charge implicitement durant l'apprentissage du langage.

Les TP ont pour objectif d'illustrer les notions enseignées durant le cours. Ces derniers doivent débuter avec les cours selon le planning suivant :

- TP's initiatiques de familiarisation avec la machine informatique d'un point de vu matériels et systèmes d'exploitation (exploration des différentes fonctionnalités des OS)
- TP's d'initiation à l'utilisation d'un environnement de programmation (Edition, assemblage, compilation etc…)
- TP's applicatifs des techniques de programmation vues en cours.

Introduction générale

Introduction générale :

Les ordinateurs sont omniprésents dans le monde moderne, mais que sont-ils ? Découvrez une définition complète, une introduction aux cas d'utilisation et aux technologies, ainsi qu'une liste des compétences nécessaires pour travailler sur le terrain.

Les ordinateurs, appareils mobiles et autres objets connectés occupent une place centrale dans notre monde moderne. Au travail comme à la maison, nous passons la majeure partie de notre journée devant des écrans.

Les ordinateurs sont devenus une partie intégrante de nos vies... à tel point que beaucoup de gens ne parviennent pas à définir le terme avec précision. Découvrez tout ce que vous devez savoir sur l'informatique avec ce document.

Qu'est-ce que l'informatique ?

L'informatique comprend l'utilisation d'ordinateurs ou d'autres équipements, infrastructures et processus pour créer, traiter, stocker, sécuriser ou échanger des données électroniques sous quelque forme que ce soit.

L'informatique a beaucoup changé depuis sa création au milieu du XXe siècle. Cette évolution se poursuit aujourd'hui avec l'avènement de nouvelles technologies telles que le cloud computing et l'informatique quantique.

L'informatique comprend à la fois des éléments physiques (matériels) et logiciels (logiciels). Parmi ces différents éléments, on peut citer la virtualisation, les systèmes de gestion, les outils d'automatisation, les systèmes d'exploitation ou encore les applications. De même, différents périphériques peuvent être inclus.

Il existe deux types de logiciels : les logiciels système et les applications. La catégorie des logiciels système comprend les systèmes d'exploitation, le BIOS, les chargeurs de démarrage, les assembleurs et les pilotes.

Parmi les applications logicielles utilisées par les entreprises, on peut citer les bases de données, les systèmes transactionnels, les serveurs de messagerie, les serveurs web comme Apache, les systèmes de gestion de la relation client ou encore les systèmes de planification des ressources. Les applications mobiles compatibles avec les smartphones et les tablettes ont élargi l'informatique et créé une nouvelle catégorie de logiciels.

En matière d'équipement ou de matériel informatique, on peut citer les serveurs, les réseaux, les disques durs, la RAM, et toutes sortes d'équipements de télécommunications utilisés pour connecter les appareils entre eux et à Internet.

L'architecture informatique moderne comprend également la virtualisation et le cloud computing. La technologie est alimentée par des serveurs distants situés dans des centres de données via Internet.

Les données sont désormais l'une des ressources les plus précieuses pour les entreprises de tous les secteurs. Les organisations de toutes tailles doivent désormais collecter et analyser des données pour rester compétitives.

Cependant, l'informatique fournit les moyens de développer, traiter, analyser, échanger, stocker et protéger l'information. Le traitement des données joue un rôle clé dans le développement et la conception des produits, le marketing, les ventes, l'acquisition et la fidélisation des clients, la comptabilité et les ressources humaines.

L'informatique est désormais omniprésente dans presque tous les secteurs de l'entreprise, même dans notre vie personnelle. Il ne se limite plus aux PC et aux serveurs, mais inclut les smartphones, les tablettes, les consoles de jeux et même les objets connectés.

Tous ces appareils informatiques sont interconnectés via Internet. Par conséquent, une expertise informatique est essentielle pour gérer, sécuriser et maintenir cet environnement complexe et potentiellement dangereux.

Chapitre 1 :

Introduction à l'informatique

I. Introduction à l'informatique

I.1. Définition de l'informatique :

Le terme informatique a été proposé en 1962 par PHILIPPE DREYFUS, pour caractériser le traitement automatique de l'information; il est composé de l'expression « INFORMAtion automaTIQUE ».

Informatique est la science du traitement automatique de l'information, la science qui permet de collecter, traiter et de transmettre l'information par Ordinateur.

I.2. Evolution de l'informatique et des ordinateurs :

Pour relater chronologiquement les avancées de l'informatique, on se basera ici principalement sur les progrès concernant les aspects matériels. Le cas des développements logiciels sera ensuite examiné.

I.2.1. 1ère époque : la mécanique

Les principes physiques qui furent utilisés dans les premières machines à calculer, ancêtres de l'ordinateur, relevaient exclusivement de la mécanique, seule science véritable de l'époque.

Les "plateaux à calcul" (1000 ans avant Jésus-Christ) (remarquons au passage que le mot latin calculi (cailloux) donna naissance au mot calcul) laissèrent vers 500 avant JC la place au boulier chinois. Cette première machine est d'ailleurs encore employée de nos jours.

Il faut attendre le 17ème siècle pour voir apparaître des machines fonctionnant à l'aide de roues ou cylindres à ergots :

- 1623 : machine de Schickardt, mathématicien allemand et universitaire ; cette machine fut construite pour effectuer des calcules d'astronomie (donc des calculs astronomiques !).

- 1643 : machine de Pascal (qui ne fonctionna paraît-il qu'en 1645). Pascal avait inventé cette machine pour venir en aide à son père, intendant des finances, afin d'effectuer des calculs financiers. De fait, cette machine effectuait des additions, soustractions et des conversions de monnaie.

- 1673 machine de Leibniz, capable théoriquement (la machine !) d'effectuer des additions et des multiplications. A cause des difficultés techniques, le premier exemplaire ne vit le jour que vingt ans après.

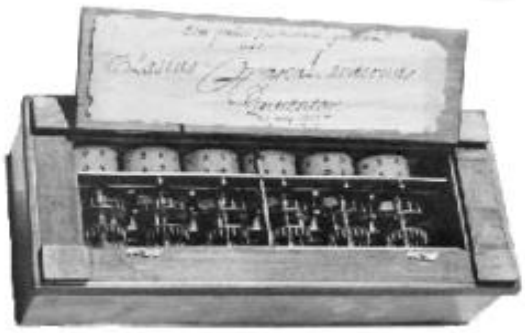


Figure I.1 : machine de Pascal 1642 (la "Pascaline")

"La machine arithmétique fait des effets qui approchent plus de la pensée que tout ce que font les animaux mais elle ne fait rien qui puisse faire dire qu'elle a de la volonté, comme les animaux." (Les Pensées de Blaise Pascal).

Figure I.2 : Machine de Wilhelm Leibniz,

présentée en 1673 à la Société Royale de Londres. La date est controversée, mais il est sûr que le philosophe et mathématicien conçut une telle calculatrice dans le même temps que Newton posait les bases du calcul différentiel et intégral.

Il est à noter que, probablement, ces machines furent inventées indépendamment les unes des autres et qu'elles utilisaient toutes les techniques dérivées de l'horlogerie : roues dentées, ergots, ...

Dans un domaine tout différent de premier abord, apparurent en 1728 le métier à tisser de Falcon, puis en 1801 le métier à tisse de Jacquard. Dans ces machines, le procédé de tissage était communiqué aux organes actifs par l'intermédiaire de bandes de carton perforé (comme dans les orgues de barbarie). On peut considérer qu'il s'agissait là, d'une part, de la première utilisation d'un programme et de cartes perforées, d'autre part, des débuts de l'informatique industrielle.

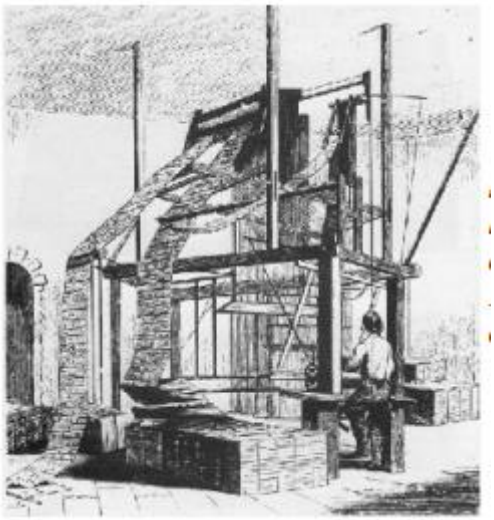


Figure I.3: Métier à tisser de Jacquard (1805). Il fut précédé par le métier à tisser de Basile Bouchon en 1725 qui utilisait déjà les rubans perforés, puis par celui de Falcon en 1728. Vaucanson s'inspira de ces travaux pour ses automates.

En 1820, la première machine à calculer "commercialisable fait son apparition sous le nom d'Arithmétomètre de Thomas de Colmar de la Compagnie d'Assurances "Le Soleil" : 1500 exemplaires vendus en 60 ans. C'est le début d'un marché qui deviendra de nos jours celui de l'informatique.

Figure I.4: Arithmétomètre de Charles Xavier de Colmar (1821).

Les grands principes des machines à calculer à programme externe furent développés par Charles Babbage vers 1830. En se basant sur les idées de Jacquard et les principes d'horlogerie des Jacquemarts (automates), il entreprit de construire une machine révolutionnaire.

Il n'y arrivera jamais faute de moyens techniques suffisants ; cependant ses plans et ses idées furent repris plus tard vers 1938. Le programme externe consistait en cartes perforées introduites dans une unité d'entrée ; le corps de la machine était constitué d'une mémoire (store), d'une unité arithmétique et logique (mill) et d'une unité de commande (control unit) ; enfin, était prévue une unité de sortie pour la restitution des résultats du calcul.

Le but de la machine était d'effectuer des opérations complexes définies comme une séquence d'opérations élémentaires. Notons que Babbage fut aidé et encouragé par Ada Lovelace, fille du poète Byron, dont le prénom a été donné à un langage de programmation moderne : ADA ; elle peut, à juste titre, être considérée comme le premier programmeur (programmeuse ?) de l'histoire.

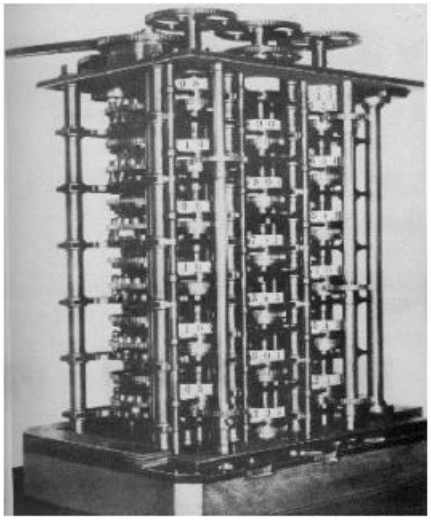


Figure I.5: machine de Charles Babbage (1833)

Figure I.6: Charles Babbage, le "premier informaticien", ne put jamais réaliser sa machine, mais l'architecture prévue est la base des prochains ordinateurs

Pour terminer la période de la mécanique, signalons la machine de Burroughs (nom célèbre aujourd'hui encore), destinée à des applications de gestion, qui était très complexe et qui réalisait une impression automatique des résultats. Burroughs créa, d'ailleurs, une société, la American Arithmetometer Co ; ce personnage était haut en couleur : en 1889, déçu de ne pas avoir pu vendre suffisamment de machines, il jeta par la fenêtre les 50 exemplaires qui lui restaient sur les bras ! Puis vint la machine de Bollée (3 exemplaires construits) en 1888 qui effectuait la multiplication par voie directe (et non par additions successives) (le père de Bollée avait construit au Mans une automobile à vapeur ; elle est à l'origine de la fameuse course des "24 heures du Mans"). En 1895 apparut la machine de Steiger ("Le Millionnaire") qui effectuait aussi les multiplications par voie directe ; elle fut vendue à plus de 1000 exemplaires en 3 ans.

Enfin, en 1912, Monroe produisit une machine qui effectuait les multiplications et les divisions par voie directe.

Mais il était difficile de faire mieux seulement avec la mécanique.

2ème époque : l'électromécanique

Les possibilités de la mécanique étaient insuffisantes (elles furent la cause de l'échec de Babbage). L'utilisation de l'électricité couplée à la mécanique fit faire de nouveaux progrès.

A cette époque, deux tendances eurent cours : celle des machines analogiques et celle des machines numériques. Les machines analogiques sont basées sur le principe très simple suivant : les nombres sont représentés par des grandeurs physiques : tensions, intensités, résistances,...

Nous ne les étudierons pas ; signalons qu'elles présentaient des inconvénients sérieux : non universelles, peu précises relativement, mémorisation difficile. Elles ne sont de nos jours utilisées que pour des applications bien précises (on peut voir un calculateur de cette sorte au CETIM à Senlis).

En 1890, Hollerith, ingénieur au bureau de recensement US, construisit une machine électromécanique pour faciliter les opérations de recensement

(essentiellement des tris et des comptages). Cette machine, appelée machine mécanographique, utilisait des cartes perforées (12cm x 16cm : 210 cases). Une "sorting box" permettait d'opérer des tris de manière automatique.

Figure I.7: Hermann Hollerith (1860-1929) utilise des machines à cartes perforées pour le recensement américain de 1890. En 1896, il fonde une société, la Tabulating Machine Company, ancêtre d'IBM.

Figure I.8: machine mécanographique de Hollerith (1890). Elle utilisait des cartes perforées, déjà en usage dans les premiers métiers à tisser.

Figure I.9 : La machine "à compter" d'Hollerith fut l'ancêtre des machines mécanographiques, appelées aussi tabulatrices.

Vers 1922, Bull, ingénieur norvégien, qui avait déjà déposé des brevets concernant la réalisation d'un ensemble de calcul électromécanique, fonda une société de construction de machines mécanographiques.

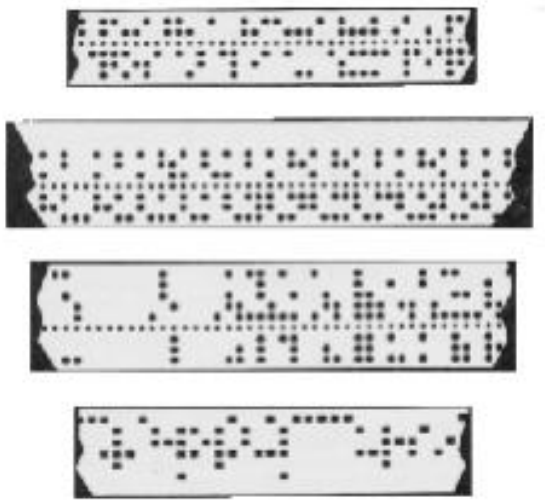


Figure I.10: rubans perforés, longtemps utilisés pour le codage de l'information et sa mémorisation

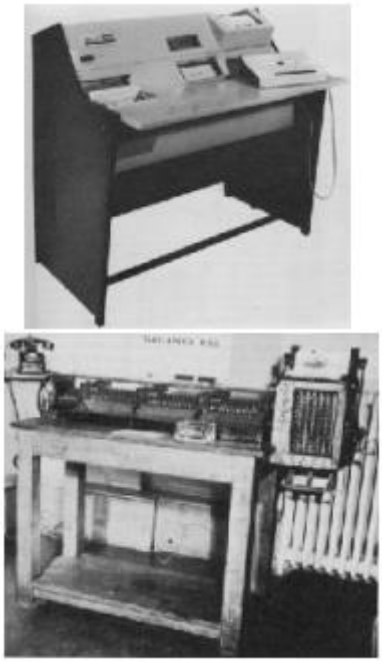


Figure I.11: perforatrices et tabulatrices sont les machines indispensables pour le codage de l'information et la manipulation des cartes perforées

Figure I.12: carte perforée au standadrd de codage IBM/Hollerith

Vers 1936, les principes qui allaient être mis en oeuvre dans la construction des futurs calculateurs furent établis par L.Couffignal (France), K.Zuse (Allemagne),

A.M.Turing (Grande-Bretagne). On vit apparaître alors les premiers calculateurs binaires : le Bell Labs Relays Computer de Stibitz (1937), machine qui utilisait des relais électromécaniques, puis le Z1 (1938) de Zuse ; il fut suivi du Z2 (1939), puis du Z3 (1941) qui disparut dans les bombardements.

Figure I.13: Konrad Zuse développa le système de calcul en "virgule flottante" (de même que George Stibitz). Il présente ici sa machine Z1.

Figure I.14: Alan Turing (1912-1954) a travaillé sur la théorie des calculateurs et a défini en 1937 le principe de la machine algorithmique et participé au projet Colossus

Enfin et surtout apparut en 1944 la machine Harvard-IBM, appelée plus tard MARK 1, réalisée conjointement par Aiken (université de Harvard) et par IBM. Cette machine concrétisait les idées de Babbage. Elle pesait 5 tonnes, avait 16,6 m de long et 2,60 de haut.

Elle comprenait 800 000 éléments dont 3 000 relais. Les informations étaient transmises par ruban perforé et la mémoire était constituée d'accumulateurs à roues et cadrans. Elle possédait une horloge de synchronisation et nécessitait un refroidissement avec des tonnes de glace.

Elle réalisait une addition en 0,6 s, une multiplication en 6 s et une division en 12 s. Des versions suivantes furent élaborées : MARK II (1944), MARK III (1949) et MARK IV (1952).

Figure I.15: Connu d'abord sous le nom de ASCC (Automatic Sequence Controlled Calculator), le MARK1 s'inspirait de la machine de Babbage.
3ème époque : l'électronique

Ici commence en fait l'histoire de l'informatique. En effet, les problèmes techniques de la réalisation des ordinateurs ne furent vraiment rsolus que par l'arrivée de l'électronique. L'usage fait que l'on parle de "générations" en ce qui concerne les phases successives du développement des ordinateurs. Cette notion de génération quoique discutable permet cependant de jalonneles progrès réalisés depuis la fin de la seconde guerre mondiale.

1ère génération : les tubes électroniques (1945-1958)

Les tubes électroniques ont permis de remplacer les relais. On peut encore en voir dans les très anciens postes de radio ou de télévision. Ils fonctionnaient en tout ou rien (le courant passe ou ne passe pas) et ressemblaient à des ampoules électriques.

Figure I.16: tubes électroniques sur la façade arrière des calculateurs de l'époque

Le premier calculateur électronique opérationnel fut l'ENIAC (Electronic Numerical Integrator And Calculator) réalisé en 1946 par J.P.Eckert et J.W.Mauchly. Cette machine qui pesait 30 tonnes occupait un volume de 85 m3 et une surface au sol de 160 m2 , contenait 18 000 tubes et consommait 150 kWh (équivalent au chauffage d'un immeuble). Elle fut élaborée à Philadelphie à l'Université de Pennsylvanie (Moore School).

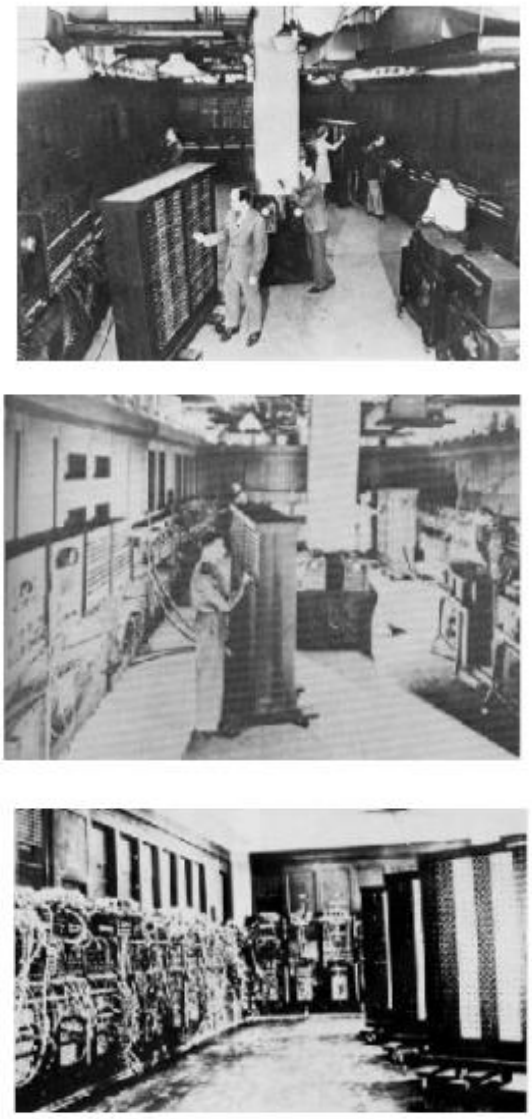


Figure I.17: L'ENIAC, est le premier grand calculateur électronique, achevé en 1943.

Figure I.18: J.Presper Eckert se passionnait pour l'électronique. Il définit une nouvelle conception des tubes à vide.

Figure I.19: John W. Mauchly (1908-1980) quitta la direction du département de Physique de l'Ursinus College en 1941 pour rejoindre la Moore School qui signa un contrat avec le Ballistic Research Lab en juin 1943 pour réaliser une machine à calculer électronique.

Dans la machine précédente, le programme était communiqué à l'unité de traitement au fur et à mesure de l'exécution, de l'extérieur. C'est en 1948 que fut réalisée la première machine à programme enregistré et donc, nous pouvons le dire, le premier ordinateur : l'IBM SSEC (Selective Sequence Electronic Calculator). 100 fois plus rapide que MARK 1, cette machine possédait 21 400 relais électromécaniques et 13 500 tubes à vide et les instructions des programmes étaient traitées comme des données. Cependant, cette machine n'était pas totalement électronique ; ce fut par contre le cas de l'ordinateur EDSAC (Electronic Delay Storage Automatic Calculator) en 1949 et de la machine de Manchester de Williams, Newmann, Kilburn, Good en 1948. .

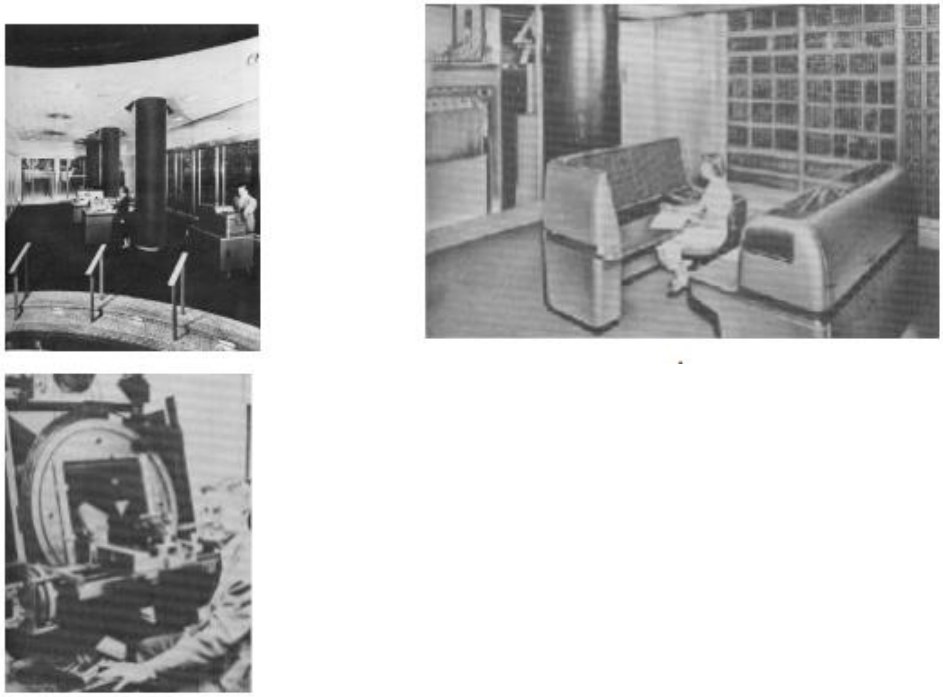


Figure I.20: Construit sous la responsabilité de Franck Hamilton et Rex Sheeber et sur les conseils de Wallace J. Eckert (à droite) , l'IBM SSEC (à gauche) est considéré comme le premier ordinateur

Figure I.21: L'Edsac pourrait être considérée comme le premier ordinateur totalement électronique, titre revendiqué également par le BINAC et le WHIRLWIND

La période qui suivit vit se développer la technologie des "lignes à retard" employée dans BINAC (1949), WHIRLWIND (1950), UNIVAC I (1951). Puis les tubes électroniques remplacèrent les lignes à retard et furent mis en oeuvre sur IBM 701 (1953). Une autre technologie des mémoires vit le jour, celle des tambours magnétiques employée sur le GAMMA ET de Bull en 1958. Enfin apparurent la technologie performante des mémoires à tores de ferrite sur UNIVAC 1103A (1954) et IBM 704 (1954).

Figure I.22: Construit par la firme anglaise Ferranti, l'Univac1 fut livré en février 1951 au Royal Society Computing Machine Laboratory de l'Université de Manchester.

Figure I.23: L'IBM 704 fut conçu par Gene Amdahl et possédait une mémoire centrale à tores de ferrite

2ème génération : les semi-conducteurs (1958-1964)

L'emploi des tubes à vide avait trois gros inconvénients : l'encombrement, la mauvaise fiabilité, la grande consommation d'énergie. L'utilisation des diodes et des transistors à semi-conducteurs (le transistor avait été découvert en1947 par Bardeen, Brattain, Shockley) à la place des tubes permit de réduire très fortement ces inconvénients. Les premiers ordinateurs entièrement transistorisés apparurent vers 1959 : GE 210, IBM 1401, NCR 304, RCA 501,... A cette époque furent construites les machines les plus puissantes possibles : LARC (1959), IBM 7030 également appelée STRETCH (1960), GAMMA 60 (1960). Voici quelques exemples de performances : addition : quelques microsecondes ; virgule flottante ; capacité mémoire allant jusqu'à 72 millions d'éléments binaires.

Figure I.24: L'architecture du Gamma60 de Bull lui permettait d'assurer trois activités : la mémoire, le multitraitement, la coordination entre les traitements et leurs ressources.

3ème génération : les circuits imprimés (1964-1970)

Les ordinateurs devenant de plus en plus complexes, il fut nécessaire de simplifier les circuits ce qui fut fait avec l'empli des circuits intégrés. Les techniques RTL (Resistor-Transistor-Logic), DTL (Diode-Transistor-Logic), puis TTL (Transistor-Transistor-Logic) se succédèrent. Ces techniques permirent de réduire considérablement la taille des ordinateurs :

- SSI (Small Size Integration : 100 transistors sur 1 cm2

- MSI (Medium Scale Integration) : 100 à 1000 transistors sur une puce de 30 mm2

- LSI (Large Scale Integration) : 1000 à 10 000 transistors sur une puce. et aujourd'hui VLSI (Very), et VVLSI (Very very)

Figure I.25: au centre, une micro-plaquette ou puce et, autour, les composants qu'elle remplace.

Figure I.26: IBM utilisa de manière industrielle les circuits intégrés

Figure I.27: un circuit intégré des années 80 dans son "pack" en céramique (Dual In Line) prévu pour l'insérer sur des circuits imprimés.

Figure I.28: détail d'un circuit intégré

En 1955 apparurent les mémoires à semi-conducteurs qui allaient peu à peu remplacer les mémoires à tores de ferrite. Les machines typiques de cette époque sont CDC 6600, IBM série 360, UNIVAC 1107, 1108,...

4ème génération : intégration poussée et micro-informatique (1970-...)

Cette génération qui débute vers 1970 concerne l'ère moderne de l'informatique. On pourrait la subdiviser en périodes successives, mais le

progrès étant si important, il est difficile de faire un découpage qui serait admis par tous.

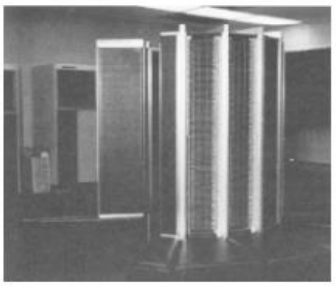


Figure I.29: Le Cray 1, imaginé par Seymour Cray, fut l'un des ordinateurs les plus puissants de la fin des années 70

La génération présente correspond aux faits marquants suivants :

- la miniaturisation n'a cessé de croître (paradoxal, non !) pour aboutir vers 1975 à la VLSI (Very Large Scale Integration), puis à l'ULSI (Ultra Large Scale Integration)
- l'apparition des microprocesseurs en 1971, puis des micro-ordinateurs en 1973 a permis le développement de l'informatique personnelle ou micro-informatique

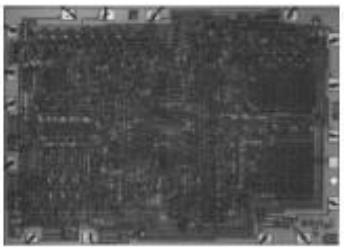


Figure I.30: Intel 4004, le premier microprocesseur,fut suivi du 8008, puis du 8080.

Figure I.31: Un an après l'invention du premier microprocesseur, des prototypes de micro-ordinateurs naissent en France (François Gernelle et André Truong Thi de la société R2E) comme le Micral N et aux Etats-Unis chez Xerox (l'Alto) et Digital. Puis vinrent le micro-ordinateur "en kit" proposé par la revue Radio-Electronics (1974) et inventé par John Titus, puis le kit Altair 8080.

Figure I.32: Clive Sinclair, inventeur britannique et virtuose du commerce des micro-ordinateurs, contribua a faire baisser les prix des micro-ordinateurs. Le jeu de la concurrence de l'époque en était la cause principale (Tandy, Commodore, Altos, Sinclair, et ... Apple)
La fin des années 70, puis les années 80 ont vu un essor particulièrement remarquable de l'informatique "personnelle".

Figure I.33: Steve Wozniak, Steve Job, Dan Kottke présentent l'Apple I (1976)

Figure I.34: L'IBM PC (1981)

Figure I.35: Le premier MacIntosh d'Apple

De fait, les innovations en matière de technologies électroniques n'ont cessé de se succéder : En 2010, Intel produit le Intel Core i7 (Gulftown) avec 1 170 000 000 transistors, une finesse de gravure 0,032 mm, (1 cheveu : 100 mm) et une puissance de 147 600 MIPS.

Figure I.36: La loi de Moore est bien vérifiée : Le nombre de transistors par puce quadruple tous les 3 ans.

Aujourd'hui concurrencé par Samsung, Apple s'est montré ces dernières années un leader en matière d'innovation s'orientant vers les téléphones intelligents comme l'Iphone ou vers les tablettes tactiles comme l'Ipad: Ipad 2


Ipad 2

- Les réseaux de transmission d'informations, les banques de données et la conjugaison de l'informatique avec les télécommunications ont donné naissance à la télématique dans un premier temps, à Internet de nos jours.

Figure I.37: Tim Berners Lee, l'inventeur du Web

- Les technologies employées pour la fabrication des composants électroniques ont fait chuter le coût des matériels de façon exponentielle (heureusement).

I.3. Concepts de base :

I.3.1. Système Informatique:

Ce système est composé de deux (02) parties essentielles et complémentaires :

- Partie Matérielle : (Hardware), c'est la partie touchable du système, c'est aussi l'ensemble des dispositifs (appareils, organes) physiques utilisés pour traiter automatiquement les informations (Ecran, souris, clavier, unité centrale, terminal, imprimante,…etc).
- Partie Logicielle : (Software), c'est la partie non touchable du système (partie logique), c'est aussi l'ensemble des logiciels, programmes et applications permettant de faire fonctionner un matériel informatique.

I.3.2. L'ordinateur:

Est une machine électronique permettant le traitement automatique des informations

I.3.3. Donnée & Information :

- Définitions:

- Donnée:
1) Représentation d'une Information sous une forme conventionnelle adaptée à son exploitation ou,
2) Est un élément de connaissance susceptible d'être représenté à l'aide des conventions pour être conservé, traité ou communiqué.
Exemple:
Pi=3.14 est une donnée

- Information:

Une information pour un individu est un signal qui produit un effet sur son comportement ou sur son état cognitif.
Exemple:
Mon âge est 32 ans,
Aujourd'hui, il fait froid

I.3.4. Traitement de l'information :

Est une suite d'opérations (finies et ordonnées), effectuée sur des données afin de résoudre un problème donné.

- Type de traitement:

1) Manuel: c'est un traitement qui nécessite l'intervention de l'être humain.
Exemple:
Préparation d'un gâteau, montage d'une chaudière…
2) Automatique: c'est un traitement qui nécessite l'intervention d'une machine intelligente (machine dotée d'une partie logicielle).
Exemple:
Calcul, trie, classement, la sélection…

- Processus de traitement:

Les entrées: Introduire les données dans la machine,
Le traitement: Appliquer les actions sur les données,
Les sorties: Restituer les résultats.

Figure I.38 : Processus de traitement de l'information

- Avantages et Inconvénients du traitement automatique

Avantages:
- Réduire l'intervention de l'être humain (minimiser les efforts),
- Gagner du Temps,
- Réduire le volume de stockage des informations (stocker une grande quantité d'informations sur une carte mémoire d'un seul Cm^2),
- Faciliter la recherche d'information
-… etc.

Inconvénients:
- La perte des informations à cause des pannes, des virus informatiques ou à cause des fausses manipulations,
- Problèmes relatifs à la sécurité des informations,
- … etc.

I.4. Les systèmes de codage des informations

Les informations traitées par ordinateur peuvent être de différents types (Texte, nombre, image, audio, vidéo, …etc.), mais elles sont toujours représentées et manipulées sous forme numérique, c.-à-d. suite de « 0 » et « 1 »,
Ces chiffres (0 et 1) sont appelés « Bit » et sont considérés comme l'unité d'information de base.

- Bit: (BInary digiT ou chiffre binaire) est la plus petite unité d'information (c'est une information élémentaire, c'est la plus petite case mémoire).
- Octet: (Byte),

Un octet est une unité d'information composée de 08 bits.
01 ctet = 08 bits

I.4.1. Les Unités de mesure de l'information

Nom	Symbole	Valeur en octet
Kilo octet	Ko	2^{10}
Méga octet	Mo	2^{20}
Giga octet	Go	2^{30}
Téra octet	To	2^{40}
Péta octet	Po	2^{50}

Remarque:
Il y a d'autres unités de mesure telles que: Exa octet, Zetta Octet et Yotta Octet

I.4.2. Codage de l'information

Le codage de l'information consiste à établir une correspondance la représentation externe (habituelle) de l'information (texte, image, son, vidéo) et sa représentation interne dans la machine, qui est toujours une suite de bits (0 et 1).
Exemple:

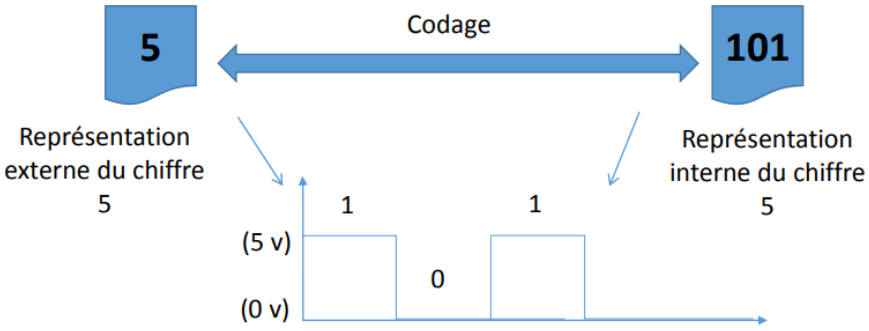


Figure I.39 : Codage de l'information

- Débit binaire:

Est la quantité d'information transmise (en bit) par unité de temps (en seconde)
D=Q/T
Tels que:
D: Débit binaire (Bit/sec)
Q: Quantité d'information (Bit)
T: Temps de transmission (Sec)

I.4.3. Système de numération

Un système de numération décrit la façon avec laquelle les nombres sont représentés.
Une Base b: La base b d'un système de numération est le nombre de différents symboles qu'utilise ce système pour représenter ses nombres.

- Bases de numération

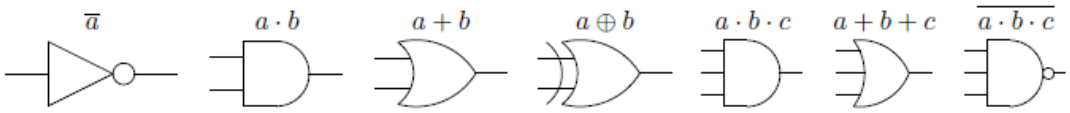
Un entier positif en base b est représenté par une suite de chiffres $(r_n r_{n-1} \ldots r_1 r_0)_B$ où les r_i sont des chiffres de la base B ($0 < ri < b$). Ce nombre a pour valeur :

$$r_n b^n + r_{n-1} b^{n-1} + \ldots + r_1 b^1 + r_0 b^0 = \sum_{i=0}^{i=n} r_i b^i$$

$$\begin{array}{lrcll} \textit{Exemples :} & (123)_{10} & = & 1.10^2 + 2.10^1 + 3.10^0 & = (123)_{10} \\ & (123)_5 & = & 1.5^2 + 2.5^1 + 3.5^0 & = (38)_{10} \\ & (123)_8 & = & 1.8^2 + 2.8^1 + 3.8^0 & = (83)_{10} \\ & (123)_{16} & = & 1.16^2 + 2.16^1 + 3.16^0 & = (291)_{10} \end{array}$$

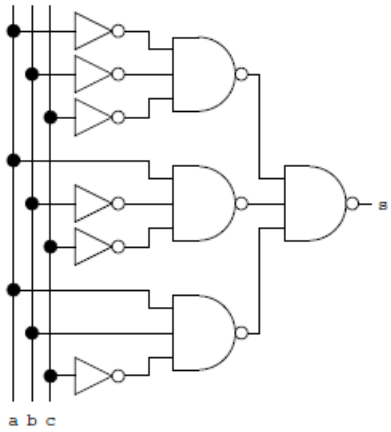
- **Circuits logiques**

Les conventions graphiques traditionnelles pour les opérateurs logiques sont:

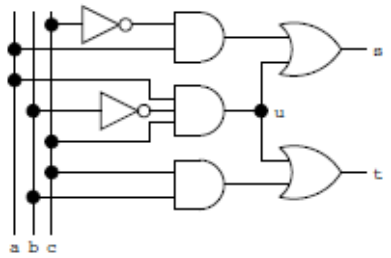


a⊕b = (a and not b) or (not a and b)

Exemples :




```
>>> s = not (not (not a and not b and not c)
...          and
...          not (a and not b and not c)
...          and
...          not (a and b and not c))
```




```
>>> u = a and (not b) and c
>>> s = (a and not c) or u
>>> t = (b and c) or u
```

- Base décimale $\boldsymbol{base10}$

Ce système de numération est très utilisé dans la vie quotidienne, il dispose de dix (10) chiffres qui sont : 0,1,2,3,4,5,6,7,8 et 9.
On écrit:
$(\boldsymbol{an-1an-2} \ldots \boldsymbol{a1a0})_{\boldsymbol{10}} = (\boldsymbol{a_{n-1}} \cdot 10^{n-1} + \boldsymbol{a_{n-2}} \cdot 10^{n-2} + \cdots + \boldsymbol{a_2} \cdot 10^2 + \boldsymbol{a_1} \cdot 10^1 + \boldsymbol{a_0} \cdot 10^0)$
Exemple:
$(2015)_{\boldsymbol{10}} \rightarrow 2015 = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 5 \cdot 10^0$

- Base binaire $\boldsymbol{base2}$

Ce système de numération utilise 2 chiffres (0 et 1), cette base souvent utilisée pour distinguer
les 2 états logiques fondamentaux (lampe allumée=1, lampe éteinte=0). C'est avec ce système
que fonctionnent les ordinateurs.
On écrit:
$(\boldsymbol{a_{n-1}a_{n-2}} \ldots \boldsymbol{a_2a_1a_0})\boldsymbol{2} = (\boldsymbol{a_{n-1}} \cdot 2^{n-1} + \boldsymbol{a_{n-2}} \cdot 2^{n-2} + \cdots + \boldsymbol{a_2} \cdot 2^2 + \boldsymbol{a_1} \cdot 2^1 + \boldsymbol{a_0} \cdot 2^0)$

Exemple:
$(4)_{\mathbf{10}} = (\mathbf{100})_{\mathbf{2}} = 1 \cdot 2^{2} + 0 \cdot 2^{1} + 0 \cdot 2^{0}$

- Base octale $\boldsymbol{base8}$

Cette base comporte les chiffres suivant : 0,1,2,3,4,5,6 et 7. Utilisée il y a un certain temps en Informatique, elle permet de coder 3 bits par un seul symbole.
On écrit :
$(\boldsymbol{a_{n-1}a_{n-2}} \ldots \boldsymbol{a_1a_0})_{\mathbf{8}} = (\boldsymbol{a_{n-1}} \cdot 8^{n-1} + \boldsymbol{a_{n-2}} \cdot 8^{n-2} + \cdots + \boldsymbol{a_2} \cdot 8^{2} + \boldsymbol{a_1} \cdot 8^{1} + \boldsymbol{a_0} \cdot 8^{0})$

Exemple:
$(125)_{\mathbf{10}} = (175)_{\mathbf{8}} = 1 \cdot 8^{2} + 7 \cdot 8^{1} + 5 \cdot 8^{0}$

- Base octale $\boldsymbol{base16}$

Cette base contient 16 alphabets qui sont : 0,1,2,3,4,5,6,8,9,A,B,C,D,E et F. Elle est très utilisée dans le monde de la micro-informatique, elle permet de coder 4 bits par un seul symbole. Avec: A=10, B=11, C=12, D=13, E=14 et F=15.
On écrit :
$(\boldsymbol{a_{n-1}a_{n-2}}\ \boldsymbol{a_1a_0})_{\mathbf{16}} = (\boldsymbol{a_{n-1}} \cdot 16^{n-1} + \boldsymbol{a_{n-2}} \cdot 16^{n-2} + \ + \boldsymbol{a_2} \cdot 16^{2} + \boldsymbol{a_1} \cdot 16^{1} + \boldsymbol{a_0} \cdot 16^{0})$
Exemple:
$(3098)_{\mathbf{10}} = (\mathrm{C1A})_{\mathbf{16}} = C \cdot 16^{2} + 1 \cdot 16^{1} + A \cdot 16^{0}$
$= 12 \cdot 16^{2} + 1 \cdot 16^{1} + 10 \cdot 16^{0}$

- Conversion (ou Transcodage de base):

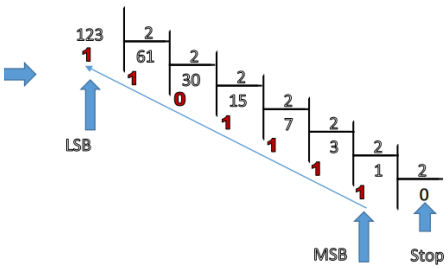
La conversion est l'opération qui permet de passer de la représentation d'un nombre exprimé dans une base à la représentation du même nombre mais exprimé dans une autre base.
Par la suite, on verra les conversions suivantes:
• Décimale vers Binaire, Octale et Hexadécimale,
• Binaire, Octale et Hexadécimale vers Décimale,
• Binaire vers Octale, Hexadécimale,
• Octale, Hexadécimale vers binaire,
• Octale vers Hexadécimale et Hexadécimale vers Octale.

- Décimale vers Binaire, Octale et Hexadécimale

Règle générale :
a) Partie Entière
On divise le nombre par la base b visée, Puis le quotient par la base b tjrs visée Ainsi de suite jusqu'à l'obtention d'un quotient nul (égal à « 0 »).
La suite des restes correspond aux symboles de la base visée dont le premier reste correspond au chiffre LSB et le dernier reste au chiffre MSB.
b) Partie Fractionnaire
On multiplie la partie fractionnaire fois la base b visée, en répétant cette opération toujours sur la partie fractionnaire du produit jusqu'à ce qu'elle soit nulle, la précision voulue soit atteinte ou une partie fractionnaire se répète.
La suite des chiffres décimaux obtenue de gauche à droite c'est la nouvelle partie fractionnaire dans la base visée .

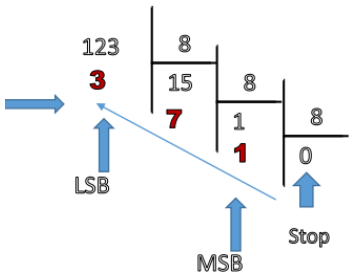
1) Décimale vers Binaire ($\boldsymbol{base10 \rightarrow base2}$)

Exemple:
$(123,25)_{10}=(\ ?\ )_2$
1) Partie Entière:
$(123)_{10}=(?\ )_2$
donc, $(123)_{10}=(1111011)_2$
Partie Fractionnaire:
$(0,25)_{10}=(\ ?\ )_2$
0,25 x 2 = 0,5 (0 à retenir)
0,5 x 2 = 1,0 (1 à retenir) d'où: (0,25)10=(0,01)**2**
Donc $(123,25)_{10}=(1111011,01)_2$



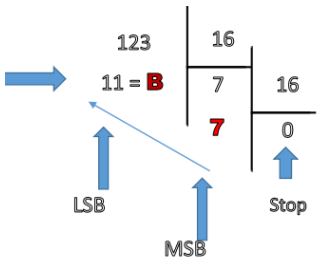
2) Décimale vers Octale ($\boldsymbol{base10 \rightarrow base8}$)

Exemple:
$(123,25)_{10}=(\ ?\ )_8$
Solution:
1) Partie Entière: $(123)_{10}=(?\ )_8$
donc, $(123)_{10}=(173)_8$
2) Partie Fractionnaire: $(0,25)_{10}=(\ ?\ )_8$
0,25 x 8 = 2,0 (2 à retenir) d'où: $(0,25)_{10}=(\ 0,2\ )_8$
Donc $(123,25)_{10}=(\ 173,2\ )_8$



3) Décimale vers Hexadécimale ($\boldsymbol{base10 \rightarrow base16}$)

Exemple:
$(123,25)_{10}=(\ ?\ )_{16}$
1) Partie Entière: $(123)_{10}=(?\ )_{16}$
donc, $(123)_{10}=(7B)_{16}$
2) Partie Fractionnaire: $(0,25)_{10}=(\ ?\ )_{16}$
0,25 x 16 = 4,0 (4 à retenir) d'où: $(0,25)_{10}=(\ 0,4\ )_{16}$
Donc $(123,25)_{10}=(\ 7B,4\ )_{16}$



- Binaire, Octale et Hexadécimale vers Décimale

Règle générale :
Soit X un nombre écrit dans une base « b » dont on veut le convertir vers la base décimale « 10 », donc, on écrit:
$X=(a_{n-1}a_{n-2}\ \dots\ \dots\ .\ a_2a_1a_0,\ a_{-1}a_{-2}a_{-3}\ \dots\ a_{-m})_b$
Partie entière Partie Fractionnaire
Donc: $(X)_{10} = a_{n-1}.\ b^{n-1}+a_{n-2}.\ b^{n-2}+\dots+a_2.\ b^2+a_1.\ b^1+a_0.\ b^0+a_{-1}.\ b^{-1}+a_{-2}.\ b^{-2}+\dots+a_{-m}.\ b^{-m}$

1) Binaire vers Décimale ($\boldsymbol{base2 \rightarrow base10}$)

Exemple:
$(1111011,01)_2=(?)_{10}$
Solution: 6 5 4 3 2 1 0 -1 -2 ← Positions des chiffres dans le nombre
$(X)_{10}=(1111011,\ 0\ 1\ )_2$

$=\mathbf{1}.\ 2^6 + \mathbf{1}.\ 2^5 + \mathbf{1}.\ 2^4 + \mathbf{1}.\ 2^3 + \mathbf{0}.\ 2^2 + \mathbf{1}.\ 2^1 + \mathbf{1}.\ 2^0 + \mathbf{0}.\ 2^{-1} + \mathbf{1}.\ 2^{-2} = 64 + 32 + 16 + 8 + 0 + 2 + 1 + 0\ .\ 0{,}5 + 0{,}25$

$= (123{,}25)_{\mathbf{10}}$

2) Octale vers Décimale ($\boldsymbol{base8 \rightarrow base10}$)

Exemple:

$(173{,}2)_{\mathbf{8}} = (?)_{\mathbf{10}}$

Solution: 2 1 0 -1 ← Positions des chiffres dans le nombre

$(X)_{\mathbf{10}} = (173,\ 2\ )_{\mathbf{8}}$

$= \mathbf{1}.\ 8^2 + \mathbf{7}.\ 8^1 + \mathbf{3}.\ 8^0 + \mathbf{2}.\ 8^{-1}$

$= 64 + 56 + 3 + 0{,}25$

$= (123{,}25)_{\mathbf{10}}$

3) Hexadécimale vers Décimale ($\boldsymbol{base16 \rightarrow base10}$)

Exemple:

$(7B{,}4)_{\mathbf{16}} = (?)_{\mathbf{10}}$

Solution: 1 0 -1 ← Positions des chiffres dans le nombre

$(X)_{\mathbf{10}} = (7\ B\ ,\ 4\ )_{\mathbf{16}}$

$= \mathbf{7}.\ 16^1 + \boldsymbol{B}.\ 16^0 + 4.\ 16^{-1}$

$= 7\ .\ 16 + 11\ .\ 1 + 0{,}25$

$= 112 + 11 + 0{,}25$

$= (123{,}25)_{\mathbf{10}}$

- Binaire vers Octale et Hexadécimale

1) Base 2 vers Base $\boldsymbol{2^n}$

Règle générale :

Chaque n bits du nombre écrit dans la base 2 seront remplacés par un seul(1) chiffre de la base 2^n c-à-d:

1. Base 2 → base 8, sachant que 8=2

3 donc chaque 3 bits de ce nombre binaire seront remplacés par un (1) chiffre du nouveau nombre de la base Octale.

2. Base 2 → base 16, sachant que 8=2

4 donc chaque 4 bits de ce nombre binaire seront remplacés par un seul (1) chiffre du nouveau nombre de la base Hexadécimale.

2) Base 2 vers base 8 ($2 \rightarrow 8 = 2^3$)

Exemple:

$(1111011{,}01)_{\mathbf{2}} = (\ ?\ )_{\mathbf{8}}$

Solution: (c'est un regroupement de 3 bits)

P entière = de Droite à gauche P Fractionnaire : de Gauche à droite

$(1111011{,}01)_{\mathbf{2}} = (001\ 111\ 011{,}010)_{\mathbf{2}}$

Donc: $(001)_{\mathbf{2}} = (\mathbf{1})_{\mathbf{10}} = (\mathbf{1})_{\mathbf{8}}$

$(111)_{\mathbf{2}} = (\mathbf{7})_{\mathbf{10}} = (\mathbf{7})_{\mathbf{8}}$

$(011)_{\mathbf{2}} = (\mathbf{3})_{\mathbf{10}} = (\mathbf{3})_{\mathbf{8}}$

$(010)_{\mathbf{2}} = (\mathbf{2})_{\mathbf{10}} = (\mathbf{2})_{\mathbf{8}}$

D'où : $(1111011{,}01)_{\mathbf{2}} = (\mathbf{173},\ \mathbf{2})_{\mathbf{8}}$

Remarque:

Compléter les bits qui manquent par (0) à gauche de la P.E et à droite de la P.F

Binaire	Décimale	Octale
000	0	0
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7

3) Base 2 vers base 16 ($2 \rightarrow 8=2^4$)

Exemple: $(1111011,01)_{\mathbf{2}}=(\ ?\ )_{\mathbf{16}}$

Solution: (c'est un regroupement de 4 bits)

P entière = de Droite à gauche P Fractionnaire : de Gauche à droite

$(1111011,01)_{\mathbf{2}}=(\ 0111\ 1011,\ 0100)_{\mathbf{2}}$

Donc: $(0111)_{\mathbf{2}}=(\mathbf{7})_{\mathbf{10}}=(\mathbf{7})_{\mathbf{16}}$

$(1011)_{\mathbf{2}}=(\mathbf{11})_{\mathbf{10}}=(\boldsymbol{B})_{\mathbf{16}}$

$(0100)_{\mathbf{2}}=(\mathbf{2})_{\mathbf{10}}=(\mathbf{4})_{\mathbf{16}}$

D'où : $(1111011,01)_{\mathbf{2}} = (\mathbf{7}\boldsymbol{B},\ \mathbf{2})_{\mathbf{16}}$

Remarque: Compléter les bits qui manquent par (0) à gauche de la P.E et à droite de la P.F

Binaire	Décimale	Hexadécimale
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

- Octale, Hexadécimale vers Binaire

1) Base $\mathbf{2^n}$ vers Base $\mathbf{2}$

Règle générale : Chaque chiffre (un seul chiffre) du nombre écrit dans la base $\mathbf{2^n}$ sera remplacé par n bits du nombre binaire c-à-d:

1. Base $8=2^{\mathbf{3}} \rightarrow$ base 2, donc chaque chiffre de ce nombre octal sera remplacé par 03 bits du nouveau nombre de la base binaire.
2. Base $16=2^{\mathbf{4}} \rightarrow$ base 16, donc chaque chiffre de ce nombre hexadécimal sera remplacé par 04 bits du nouveau nombre de la base binaire.

2) Base 8 vers base 2 ($8=2^3 \rightarrow 2$)

Exemple: $(173,2)_{\mathbf{8}}=(\ ?\ )_{\mathbf{2}}$

Solution: (c'est un éclatement de 3 bits) Donc:

$(\mathbf{1})_{\mathbf{8}}=(\mathbf{1})_{\mathbf{10}}=(001)_{\mathbf{2}}$

$(\mathbf{7})_{\mathbf{8}}=(\mathbf{7})_{\mathbf{10}}=(111)_{\mathbf{2}}$

$(\mathbf{3})_{\mathbf{8}}=(\mathbf{3})_{\mathbf{10}}=(011)_{\mathbf{2}}$

$(\mathbf{2})_{\mathbf{8}}=(\mathbf{2})_{\mathbf{10}}=(010)_{\mathbf{2}}$

D'où : $(\mathbf{173, 2})_{\mathbf{8}} = (001\ 111\ 011, 010)_{\mathbf{2}}$

Remarque: Les bits à gauche de la P.E et à droite de la P.F sont des bits inutiles et peuvent être supprimés

Binaire	Décimale	Octale
000	0	0
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7

3) Base 16 vers base 2 (16=24□2)

Exemple: $(7B,4)_{\mathbf{16}}=(\ ?\ )_{\mathbf{2}}$

Solution: (c'est un éclatement de 4 bits) Donc:

$(\mathbf{7})_{\mathbf{16}}=(\mathbf{7})_{\mathbf{10}}=(0111)_{\mathbf{2}}$

$(\boldsymbol{B})_{\mathbf{16}}=(\mathbf{11})_{\mathbf{10}}=(1011)_{\mathbf{2}}$

$(\mathbf{4})_{\mathbf{16}}=(\mathbf{2})_{\mathbf{10}}=(0100)_{\mathbf{2}}$

D'où : $(\mathbf{7}\boldsymbol{B}, \mathbf{4})_{\mathbf{16}} = (0111\ 1011, 0100)_{\mathbf{2}}$

Remarque: Les bits à gauche de la P.E et à droite de la P.F sont des bits inutiles et peuvent être supprimés

Binaire	Décimale	Hexadécimale
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

- Octale vers Hexadécimale et Hexadécimale vers Octale

Règle générale :
Pas de méthode de conversion directe,
Il faut utiliser une base intermédiaire →(base 2 ou base 10) C-à-d:
Base 8 → Base 10 → base 16 ou
Base 8 → Base 2 → base 16 Et
Base 16 → Base 10 → base 8 ou
Base 16 → Base 2 → base 8

- Opérations sur les nombres binaires

1) Addition (+)

0	0	1	1
+ 0	+1	+0	+1
0	1	1	10

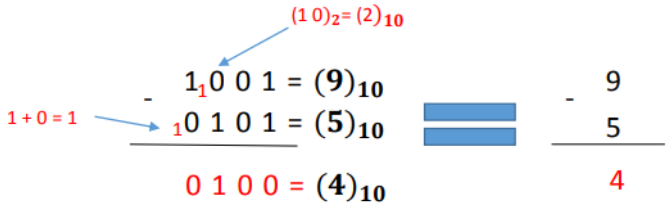
Exemple:

$$\begin{array}{r} 1\ 1\ 0\ 0\ 0\ 1\ 1 \quad =(\mathbf{99})_{10} \\ +\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1 \quad =(\mathbf{139})_{10} \\ \hline 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0 \quad =(\mathbf{238})_{10} \end{array}$$

2) Soustraction (-)

0	0	1	1
- 1	-0	-0	-1
1	0	1	0

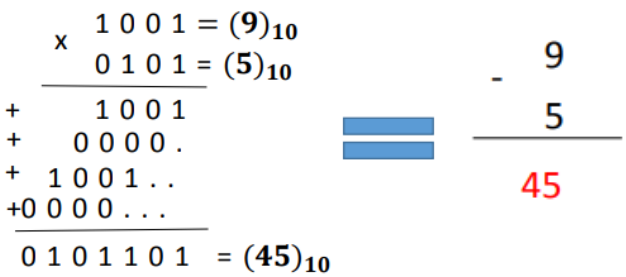
Exemple:



3) Multiplication (x)

0	0	1	1
* 1	* 0	* 0	* 1
0	0	0	1

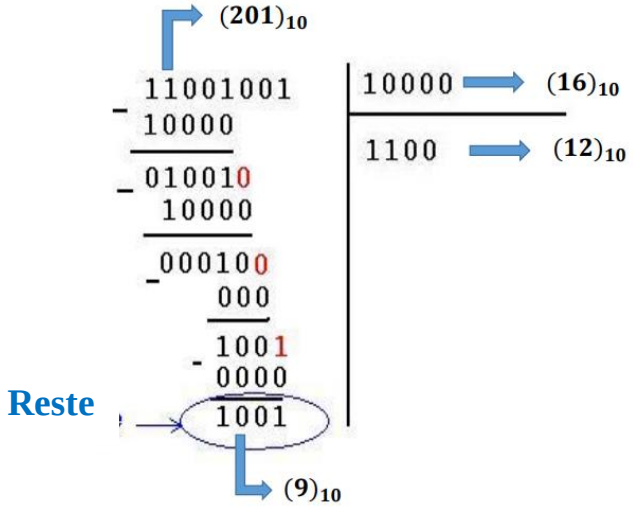
Exemple:



4) Division (/)

Exactement comme la division décimale.

Exemple:



I.4.3.4 Codage des Images

1) Définition :

Une image numérique est composée de pixels (contraction de PICTure ELements). Ses caractéristiques principales sont :
Sa largeur et sa hauteur en pixels, son nombre total de pixels et le nombre de couleurs que chaque pixel peut prendre : on parle de son codage.
Nous avons: V = Largeur x Hauteur (pixel)

2) Types d'images :

- Image en Noir et Blanc:
  01 pixel = 01 bit
- Image en Niveau de gris:
  01 pixel = 08 bits
- Image en Couleur RVB (R:Rouge, V: vert et B:Bleu)
  01 pixel = 24 bits

Fin du chapitre.

Chapitre 2 :

Notions d'algorithme et de programme

II.1 Introduction

Le terme *informatique* est un néologisme (nouveau mot) proposé en 1962 par *Philippe Dreyfus* pour caractériser **le traitement automatique de l'information** : il est construit sur la contraction de l'expression « information automatique ». Ce terme a été accepté par l'Académie Française en avril 1966, et l'informatique devint alors officiellement la science du traitement automatique de l'information, où l'information est considérée comme le support des connaissances humaines et des communications dans les domaines techniques, économiques et sociaux. Le mot INFORMATIQUE n'a pas vraiment d'équivalent aux Etats-Unis où l'on parle de *Computing Science* (science du calcul) alors que *Informatics* est admis par les Britanniques.

II.2 Rappel

L'informatique est la science du traitement automatique de l'information.

II.2.1 Matériel

Le matériel informatique est un ensemble de dispositifs physiques utilisés pour traiter automatiquement des informations.

II.2.2 Logiciel

Le logiciel est un ensemble structuré d'instructions décrivant un traitement d'informations à faire réaliser par un matériel informatique.

II.2.3 Ordinateur

Une machine effectuant des opérations simples sur des séquences de signaux électriques, lesquels sont conditionnés de manière à ne pouvoir prendre que deux états seulement (un potentiel électrique maximum ou minimum). Ces séquences de signaux obéissent à une logique binaire du type « tout ou rien » et peuvent donc être considérés conventionnellement comme des suites de nombres ne prenant jamais que les deux valeurs 0 et 1(nombres binaires).

II.3 Algorithmique

II.3.1 Algorithme

Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.

II.3.2 Algorithmique

L'algorithmique est la science des algorithmes.

III.3.3 Validité d'un algorithme

La validité d'un algorithme est son aptitude à réaliser exactement la tâche pour laquelle il a été conçu.

II.3.4 Complexité d'un algorithme

La complexité d'un algorithme est le nombre d'instructions élémentaires à exécuter pour réaliser la tâche pour laquelle il a été conçu.

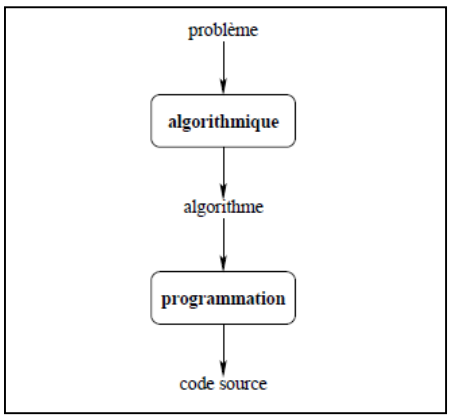


Figure II.1 : Du problème au code source

L'algorithmique permet ainsi de passer d'un problème à résoudre à un algorithme qui décrit la démarche de résolution du problème. La programmation a alors pour rôle de traduire cet algorithme dans un langage « compréhensible » par l'ordinateur afin qu'il puisse exécuter l'algorithme automatiquement (figure 1).

II.4 Programmation

II.4.1 Bit

Un bit est un chiffre binaire (0 ou 1). C'est l'unité élémentaire d'information.

II.4.2 Octet

Un octet est une unité d'information composée de 8 bits.

Pour « parler » à un ordinateur, il nous faudra donc utiliser des systèmes de traduction automatiques, capables de convertir en nombres binaires des suites de caractères formant des mots-clés (anglais en général) qui seront plus significatifs pour nous. Le système de traduction proprement dit s'appellera interpréteur ou bien compilateur, suivant la méthode utilisée pour effectuer la traduction (figure 2).

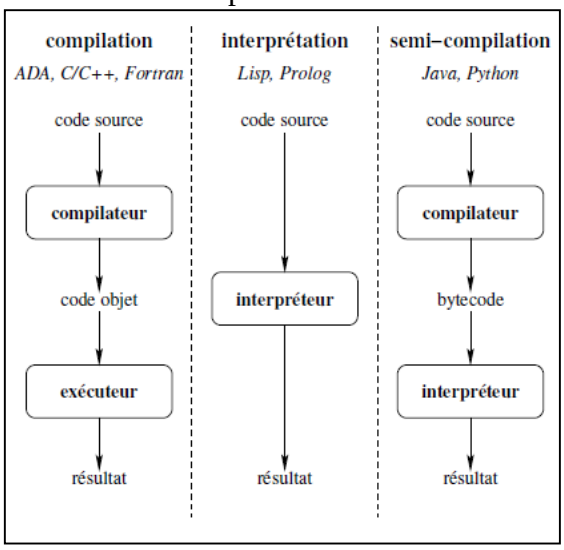


Figure II.2 : Du code source à son exécution

II.4.3 Compilateur

Un compilateur est un programme informatique qui traduit un langage, le langage source, en un autre, appelé le langage cible.

II.4.4 Interpréteur

Un interpréteur est un outil informatique (logiciel ou matériel) ayant pour tâche d'analyser et d'exécuter un programme écrit dans un langage source.

II.4.5 Langage de programmation

Un langage de programmation est un langage informatique, permettant à un humain d'´ecrire un code source qui sera analysé par un ordinateur.

II.4.6 Programmation

La programmation est l'activité de rédaction du code source d'un programme.

II.5 les variables

II.5.1 A quoi servent les variables ?

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs. Il peut s'agir de données issues du disque dur, fournies par l'utilisateur (frappées au clavier), etc. Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs. Ces données peuvent être de plusieurs types (on en reparlera) : elles peuvent être des nombres, du texte, etc. Toujours est-il que dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une variable.

Déclaration des variables

La première chose à faire avant de pouvoir utiliser une variable est de créer la boîte et de lui coller une étiquette. Ceci se fait tout au début de l'algorithme, avant même les instructions proprement dites. C'est ce qu'on appelle la déclaration des variables. Le nom de la variable (l'étiquette de la boîte) obéit à des impératifs changeant selon les langages. Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces.

Lorsqu'on déclare une variable, il ne suffit pas de créer une boîte (réserver un emplacement mémoire) ; encore doit-on préciser ce que l'on voudra mettre dedans, car de cela dépendent la taille de la boîte (de l'emplacement mémoire) et le type de codage utilisé.

II.5.2 Types numériques classiques

Commençons par le cas très fréquent, celui d'une variable destinée à recevoir des nombres. Si l'on réserve un octet pour coder un nombre, en rappel on ne pourra coder que $2^8 = 256$ valeurs différentes. Cela peut signifier par exemple les nombres entiers de 1 à 256, ou de 0 à 255, ou de –127 à +128… Si l'on réserve deux octets, on a droit à 65 536 valeurs ; avec trois octets, 16 777 216, etc. Et là se pose un autre problème : ce codage doit-il représenter des nombres décimaux ? Des nombres négatifs ?Q

Bref, le type de codage (autrement dit, le type de variable) choisi pour un nombre va déterminer :

- Les valeurs maximales et minimales des nombres pouvant être stockés dans la variable
- La précision de ces nombres (dans le cas de nombres décimaux).

Tous les langages, quels qu'ils soient offrent un « bouquet » de types numériques, dont le détail est susceptible de varier légèrement d'un langage à l'autre. Grosso modo, on retrouve cependant les types suivants :

Type Numérique	Plage
Byte (octet)	0 à 255
Entier simple	-32 768 à 32 767
Entier long	-2 147 483 648 à 2 147 483 647
Réel simple	$-3{,}40\times10^{38}$ à $-1{,}40\times10^{45}$ pour les valeurs négatives $1{,}40\times10^{-45}$ à $3{,}40\times10^{38}$ pour les valeurs positives
Réel double	$1{,}79\times10^{308}$ à $-4{,}94\times10^{-324}$ pour les valeurs négatives $4{,}94\times10^{-324}$ à $1{,}79\times10^{308}$ pour les valeurs positives

Certains langages autorisent d'autres types numériques, notamment :
- le type monétaire (avec strictement deux chiffres après la virgule)
- le type date (jour / mois / année).

II.5.3 Type alphanumérique

Fort heureusement, les boîtes que sont les variables peuvent contenir bien d'autres informations que des nombres. Sans cela, on serait un peu embêté dès que l'on devrait stocker un nom de famille, par exemple.
On dispose donc également du type alphanumérique (également appelé type caractère, type chaîne ou en anglais, le type *string*). Dans une variable de ce type, on stocke des caractères, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou même de chiffres. Le nombre maximal de caractères pouvant être stockés dans une seule variable string dépend du langage utilisé. Un groupe de caractères (y compris un groupe de un, ou de zéro caractères), qu'il soit ou non stocké dans une variable, d'ailleurs, est donc souvent appelé chaîne de caractères.
En pseudo-code, une chaîne de caractères est toujours notée entre guillemets
Pourquoi ? Pour éviter deux sources principales de possibles confusions :
- la confusion entre des nombres et des suites de chiffres. Par exemple, 423 peut représenter le nombre 423 (quatre cent vingt-trois), ou la suite de caractères 4, 2, et 3. Et ce n'est pas du tout la même chose ! Avec le premier, on peut faire des calculs, avec le second, point du tout. Dès lors, les guillemets permettent d'éviter toute ambiguïté : s'il n'y en a pas, 423 est quatre cent vingt trois. S'il y en a, "423" représente la suite des chiffres 4, 2, 3.
- …Mais ce n'est pas le pire. L'autre confusion, bien plus grave - et bien plus fréquente – consiste à se mélanger les pinceaux entre le nom d'une variable et son contenu. Pour parler simplement, cela consiste à confondre l'étiquette d'une boîte et ce qu'il y a à l'intérieur… On reviendra sur ce point crucial dans quelques instants.

II.5.4 Type booléen

Le dernier type de variables est le type booléen : on y stocke uniquement les valeurs logiques VRAI et FAUX. On peut représenter ces notions abstraites de VRAI et de FAUX par tout ce qu'on veut : de l'anglais (TRUE et FALSE) ou des nombres (0 et 1). Peu importe. Ce qui compte, c'est de comprendre que le type booléen est très économique en termes de place mémoire occupée, puisque pour stocker une telle information binaire, un seul bit suffit.

II.6 Instructions de base

Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents. Ainsi quand on définit un algorithme,

celui-ci ne doit contenir que des instructions compréhensibles par celui qui devra l'exécuter.

II.6.1 Jeu d'instructions

Chaque microprocesseur a son jeu d'instructions de base dont le nombre varie typiquement de quelques dizaines à quelques centaines selon le type d'architecture du processeur. On peut classer ces instructions de base en 5 grands groupes : les opérations arithmétiques (+, -, *, /. . .), les opérations logiques (not, and, or. . .), les instructions de transferts de données (load, store, move. . .), les instructions de contrôle du flux d'instructions (branchements impératifs et conditionnels, boucles, appels de procédure. . .), et les instructions d'entrée-sortie (read, write. . .).

Le traitement des ces instructions par le microprocesseur passe par 5 étapes :
1. fetch : *chargement depuis la mémoire de la prochaine instruction à exécuter,*
2. decode : *décodage de l'instruction,*
3. load operand : *chargement des données nécessaires à l'instruction,*
4. execute : *exécution de l'instruction,*
5. result write back : *mise à jour du résultat dans un registre ou en mémoire.*

Le langage machine est le langage compris par le microprocesseur. Ce langage est difficile à maîtriser puisque chaque instruction est codée par une séquence donnée de bits. Afin de faciliter la tâche du programmeur, on a d'abord créé le langage assembleur qui utilise des mnémoniques pour le codage des instructions puis les langages de plus haut niveau d'expressivité (fortran, C, Java, Python. . .). Le tableau ci-dessous compare les codes équivalents pour décrire l'addition de 2 entiers dans différents langages informatiques : le langage machine, le langage assembleur, le langage Pascal et le langage Python. On constate sur cet exemple une évolution progressive du pouvoir d'expressivité des langages, du langage machine aux langages de haut niveau.

machine	assembleur	PASCAL	PYTHON

A1 00 01
8B 1E 02 01
01 D8
A3 04 01
	MOV AX,[100h]
	MOV BX,[102h]
	ADD AX,BX
	MOV [104h],AX
		var a,b,c : integer ;
		c := a + b ;
			c = a + b

II.6.2 L'instruction d'affectation

II.6.2.1 Syntaxe et signification

Abordons maintenant nos premières véritables manipulations d'algorithmique. Pas trop tôt, certes, mais pas moyen de faire autrement ! En fait, la variable (la boîte) n'est pas un outil bien sorcier à manipuler. On ne peut pas faire trente-six mille choses avec une variable, mais seulement une et une seule. Cette seule chose qu'on puisse faire avec une variable, c'est l'affecter, c'est-à-dire lui attribuer une valeur. Pour poursuivre la superbe métaphore filée déjà employée, on peut remplir la boîte.

En pseudo-code, l'instruction d'affectation se note avec le signe ←

Exemple :

Début
Riri ← "2018"
Fifi ← Riri
Fin

Dans l'exemple, Riri étant dépourvu de guillemets, n'est pas considéré comme une suite de caractères, mais comme un nom de variable. Le sens de la ligne devient donc

« affecte à la variable Fifi le contenu de la variable Riri ». A la fin de l'algorithme, la valeur de la variable Fifi est donc « 201 ». Ici, l'oubli des guillemets conduit certes à un résultat, mais à un résultat différent.

II.6.3 Les instructions de lecture et d'écriture

Tout bêtement, pour que l'utilisateur entre la (nouvelle) valeur de Titi, on mettra :
Lire Titi
Dès que le programme rencontre une instruction Lire, l'exécution s'interrompt, attendant la frappe d'une valeur au clavier
Dès lors, aussitôt que la touche Entrée (Enter) a été frappée, l'exécution reprend.
Dans le sens inverse, pour écrire quelque chose à l'écran, c'est aussi simple que :
Ecrire Toto
Avant de Lire une variable, il est très fortement conseillé d'écrire des libellés à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper (sinon, le pauvre utilisateur passe son temps à se demander ce que l'ordinateur attend de lui… et c'est très désagréable !):

Ecrire "Entrez votre nom : "
Lire NomFamille

Lecture et Ecriture sont des instructions algorithmiques qui ne présentent pas de difficultés particulières, une fois qu'on a bien assimilé ce problème du sens du dialogue (homme _ machine, ou machine _ homme).

II.6.4 Les tests : les structures alternatives

Le test est une instruction de contrôle du flux d'instructions qui permet d'exécuter une instruction sous condition préalable.

II.6.4.1 Structure d'un test

Il n'y a que deux formes possibles pour un test ; la forme de gauche est la forme complète, celle de droite la forme simple.

Si booléen Alors
Instructions 1
Sinon
Instructions 2
Finsi;

Si booléen Alors
Instructions
Finsi;

Un booléen est une expression dont la valeur est VRAI ou FAUX. Cela peut donc être (il n'y a que deux possibilités) :

- une variable de type booléen ;
- une condition.

Toujours est-il que la structure d'un test est relativement claire. Arrivé à la première ligne (Si…Alors) la machine examine la valeur du booléen. Si ce booléen a pour valeur VRAI, elle exécute la série d'instructions 1. Cette série d'instructions peut être très brève comme très longue, cela n'a aucune importance. A la fin de cette série d'instructions, au moment où elle arrive au mot « Sinon », la machine saute directement à la première instruction située après le « Finsi ».

De même, au cas où le booléen a comme valeur « Faux », la machine saute directement à la première ligne située après le « Sinon » et exécute l'ensemble des « instructions 2 ».

II.6.4.2 Qu'est ce qu'une condition ?

Une condition est une comparaison ; Cette définition est essentielle ! Elle signifie qu'une condition est composée de trois éléments :

- une valeur ;
- un opérateur de comparaison ;
- une autre valeur.

Les valeurs peuvent être a priori de n'importe quel type (numériques, caractères…). Mais si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du même type !
Les opérateurs de comparaison sont :

- = égal à… ;
- <> différent de… ;
- < strictement plus petit que… ;
- > strictement plus grand que… ;
- =< plus petit ou égal à… ;
- >= plus grand ou égal à…

On peut trouver aussi des conditions composées, reliées par ce qu'on appelle des opérateurs logiques: l'informatique met à notre disposition quatre opérateurs logiques:
ET, OU, NON, et **XOR.**

II.6.4.3 Tests imbriqués

Un SI ouvre donc deux voies, correspondant à deux traitements différents, mais il y a des tas de situations où deux voies ne suffisent pas. Par exemple, un programme devant donner l'état de l'eau selon sa température doit pouvoir choisir entre trois réponses possibles (solide, liquide ou gazeuse).
Une première solution serait la suivante :

```
Variable Temp: Entier;
Début
Ecrire ("Entrez la température de l'eau :");
Lire (Temp);
Si Temp =< 0 Alors
Ecrire ("C'est de la glace");
Finsi;
Si (Temp > 0) Et (Temp < 100) Alors
Ecrire ("C'est du liquide");
Finsi;
Si Temp > 100 Alors
Ecrire ("C'est de la vapeur");
```

Finsi;
Fin.

Vous constaterez que c'est un peu laborieux. Les conditions se ressemblent plus ou moins, et surtout on oblige la machine à examiner trois tests successifs alors que tous portent sur une même chose, la température (la valeur de la variable Temp). Il serait ainsi bien plus rationnel d'imbriquer les tests de cette manière :

Variable Temp: Entier;
Début
Ecrire ("Entrez la température de l'eau :");
Lire (Temp);
Si Temp =< 0 Alors
Ecrire ("C'est de la glace");
Sinon
Si Temp < 100 Alors
Ecrire ("C'est du liquide");
Sinon
Ecrire ("C'est de la vapeur");
Finsi;
Finsi;
Fin.

II.6.5 Les Boucles : les structures répétitives

On est arrivés, la voilà, la quatrième et dernière structure : ça est les boucles. Si vous voulez épater vos amis, vous pouvez également parler de structures répétitives, voire carrément de structures itératives. Ici, entre nous, on parlera de boucles. Toutes les structures itératives répètent l'exécution de traitement(s). Deux cas sont cependant à envisager, selon que :

- le nombre de répétitions est connu à l'avance : c'est le cas des boucles itératives
- le nombre de répétitions n'est pas connu ou est variable : c'est le cas des boucles conditionnelles

II.6.5.1 La structure POUR … DE … A …, FAIRE

Cette structure est une BOUCLE ITERATIVE ; elle consiste à répéter un certain traitement un nombre de fois fixé à l'avance.
En algorithmique, on traduit cette structure itérative à l'aide des instructions :
Pour i de 1 jusqu'à N (on répète un nombre connu de fois le même traitement ; ici, de 1 à N, donc N fois)
Faire traitement 1 (instructions à effectuer)
FinPour
Remarques:

- La variable est un compteur, dont la valeur augmente automatiquement de 1 à chaque tour. Cette variable permet en définitive de contrôler le

nombre entier de tours. Cette variable est en d'autres termes un variable de contrôle d'itération, caractérisée par sa valeur initiale, sa valeur finale et son pas de variation.

- La sortie de la boucle s'effectue lorsque le nombre souhaité d'itérations est atteint, c'est-à-dire lorsque i prend la valeur N.

II.6.5.2 La structure TANT QUE …, FAIRE

Parfois, pour réaliser une tâche, on doit effectuer plusieurs fois les mêmes instructions, sans que le nombre de fois soit déterminé à l'avance. On utilise alors une BOUCLE CONDITIONNELLE. Dans cette structure, le même traitement est effectué tant qu'une condition reste valide ; la boucle s'arrête quand celle-ci n'est plus remplie. Cette structure répétitive est ainsi formulée :

Tant que condition (on répète un nombre inconnu de fois le même traitement, autant de fois que la condition
est vérifiée)
Faire traitement (instructions à effectuer)
FinTant que

Remarques :

- Le nombre de répétitions dépendra de la condition.
- Si la condition n'est pas vérifiée au début, alors le traitement 1 ne sera pas exécuté du tout.
- Si la condition est vérifiée au début et si la condition n'est pas susceptible d'être modifiée lors du traitement 1, alors le traitement 1 sera indéfiniment exécuté et l'utilisateur sera contraint d'arrêter le programme. Dans ce cas, il s'agit d'une erreur majeure car un programme ne doit pas boucler indéfiniment mais au contraire s'arrêter automatiquement une fois que la condition cesse d'être vérifiée.

Exemple :

Dans tout programme, on met en place ce qu'on appelle un contrôle de saisie, afin de
vérifier que les données entrées au clavier correspondent bien à celles attendues par
l'algorithme. Voyons voir ce que ça donne :

```
Variable Rep en Caractère
Début
Ecrire("Voulez vous un café ? (O/N)" );
Lire (Rep) ;
TantQue (Rep <> "O") ET (Rep <> "N")
Lire (Rep) ;
FinTantQue ;
Fin.
```

II.6.5.3 La structure REPETER … JUSQU'A …

Une variante de structure répétitive avec BOUCLE CONDITIONNELLE consiste à répéter un traitement jusqu'à ce qu'une certaine condition soit vérifiée. On la traduit par l'instruction :

Répète (on répète un nombre inconnu de fois le même traitement, autant de fois que la condition est vérifiée)
traitement (instructions à effectuer)
Jusqu'à condition
Remarque :
Dans ce type d'instruction, le test est effectué à la fin de la boucle, si bien que le traitement est exécuté au moins une fois, que la condition soit ou non vérifiée au début.

II.6.5.4 Des boucles dans des boucles

De même qu'une structure SI … ALORS peut contenir d'autres structures SI … ALORS, une boucle peut tout à fait contenir d'autres boucles.

```
Variables Truc, Trac en Entier
Pour Truc = 1 à 15
Ecrire ("Il est passé par ici") ;
Pour Trac = 1 à 6
Ecrire ("Il repassera par là") ;
Trac Suivant
Truc Suivant
```

Dans cet exemple, le programme écrira une fois "il est passé par ici" puis six fois de suite "il repassera par là", et ceci quinze fois en tout. A la fin, il y aura donc eu 15 x 6 = 90 passages dans la deuxième boucle (celle du milieu), donc 90 écritures à l'écran du message « il repassera par là ». Notez la différence marquante avec cette structure :

```
Variables Truc, Trac : Entier ;
Pour Truc = 1 à 15
Ecrire ("Il est passé par ici") ;
Truc Suivant
Pour Trac = 1 à 6
Ecrire ("Il repassera par là") ;
Trac Suivant
```

Ici, il y aura quinze écritures consécutives de "il est passé par ici", puis six écritures consécutives de "il repassera par là", et ce sera tout.
Des boucles peuvent donc être imbriquées (cas n°1) ou successives (cas n°2).

II.7 Les instructions en Pascal

II.7.1 Instructions composées

Une instruction spécifie une opération ou un enchaînement d'opérations à exécuter sur des objets. Les instructions sont séparées par des ; et sont exécutées séquentiellement, c'est-à-dire l'une après l'autre, depuis le BEGIN jusqu'au END. final.
Instruction déjà vues

```
{ a := 5 affectation
{ writeln ('Bonjour') affichage
{ readln (x) lecture
```

Plus généralement : Soient I1, I2, etc, des instructions.

On fabrique dans la suite de nouvelles instructions :
{ if expr then I1
{ while test do I1
{ etc
Il est possible de regrouper l'enchaînement I1; I2; I3; en une instruction unique en l'encadrant entre un begin et un end :
begin I1; I2; I3; end
Intérêt On veut faire I1 puis I2 dans un if.

```
if B then I1; I2;
   if B then
   begin I1;
      I2;
   end;
I3;
```

Dans la 1ière forme, I2 ne fait pas partie du if, dans la 2nde oui.

II.7.2 Les branchements

Etant donné une expression et plusieurs instructions, la valeur de l'expression va déterminer laquelle de ces instructions sera exécutée.
En Pascal il y a 2 types de branchements, le if et le case.

II.7.2.1 Le test booléen if

L'instruction ci-dessous prend 2 formes, elle signifie si . . . alors . . . sinon.
Syntaxe
if B then I1;
if B then I1 else I2;
B est une expression booléenne, I1 et I2 sont des instructions. L'expression B est évaluée ; si elle est vraie, alors I1 est exécutée, sinon I2 est exécutée.
Remarque :
On peut se passer de else en n'employant que des if then, mais c'est moins efficace, et on peut facilement se tromper : l'exemple suivant ne donne pas les mêmes résultats !

```
a := 1;
 { sans else }                  { avec else }
if a = 1 then a := 2;           if a = 1 then a := 2
if a < > 1 then a := 3;         else a := 3;
```

On peut imbriquer des if then else de différentes manières :

```
{ forme 1 }                     { forme 2 }
if B1                           if B1
then I1                         then if B2
else if B2                         then Ia
    then I2                           else Ib
       else if B3                        else if B3
          then I3                           then Ic
             else Iautre;                         else Id;
```

Règles :

- Il n'y a jamais de ; avant le else .

- Le else se rapporte toujours au dernier then rencontré.

Problème :

Dans la deuxième forme, comment supprimer l'instruction Ib ? On ne peut pas simplement supprimer la ligne else Ib, car alors le else if B3 se rapporterait à then Ia.

On ne peut pas non plus rajouter un ; car il y a un else après.

La solution consiste à protéger if B2 then Ia; dans un begin end :

```
if B1
then begin
     if B2
     then Ia;
     end
else if B3
     then Ic
     else Id;
```

Remarque :

Il faut faire très attention aux tests multiples, imbriqués ou non, et être très rigoureux dans l'écriture.

II.7.2.2 Sélection de cas avec case

Syntaxe :

```
case E of
    C1 : Ia;
    C2 : Ib;
    C3, C4 : Ic; { liste }
    C5..C6 : Id; { intervalle }
    { ... }
    else Iautre; { en option }
end;
```

Cette instruction signifie un choix qui permet d'exécuter l'une des instructions Ix selon le cas E. E est une expression ordinale (dont le type est un entier, un caractère, un booléen, ou un énuméré, mais pas un réel ni une chaîne de caractères). Les Cx sont des constantes ordinales du même type que E.

Comment ça marche ? E est évalué. Ensuite, est recherchée parmi les valeurs possibles Cx, laquelle est égale à E. L'instruction correspondante Ix est alors exécutée.

Sinon, l'instruction après le else (s'il y en a un) est exécutée.

L'exemple ci-dessus est équivalent à une forme en if then else imbriqués.

```
V := E; { evalué une seule fois au debut }
if V = C1 then Ia
else if V = C2 then Ib
else if (V = C3) or (V = C4) then Ic
else if (V >= C5) and (V <= C6) then Id
else Iautre;
```

On préféré la forme avec le case, qui est plus lisible et plus efficace.

Exercice complet :

Ecrire un programme qui lit un caractère, puis classe ce caractère comme espace, lettre, digit ou autre.

```
PROGRAM caractere;
TYPE
 nat_t = (Espace, Lettre, Digit, Autre);
VAR
 nat : nat_t; { nature }
 c : char;
BEGIN
   write ('Rentrez un caractere :');
   readln(c);
   { analyse de c }
   case c of
     'a'..'z', 'A'..'Z', '_' : nat := Lettre;
     '0'..'9' : nat := Digit;
     ' ' : nat := Espace;
   else nat := Autre;
   end; { case c }
   { affichage de nat }
   case nat of
    Espace : writeln ('Espace');
    Lettre : writeln ('Lettre');
    Digit : writeln ('Digit');
    Autre : writeln ('Autre');
    else { case nat }
          writeln ('Erreur case nat : ', ord(nat), ' non prevu');
    end; { case nat }
END.
```

Exercice :

Réécrire l'exemple suivant, (Type énuméré) avec un case :

```
VAR
 feux : (Rouge, Orange, Vert, Clignotant);
BEGIN
   { ... }
   if feux = Rouge
   then Arreter
   else if feux = Orange
   then Ralentir
   else if feux = Vert
   { ... }
END.
```

Bonnes habitudes :

- Après le else et le end, marquer en commentaire qu'ils se rapportent au case.
- Faire afficher un message d'erreur après le else : aide à la mise au point du programme.

II.7.3 Les boucles

II.7.3.1 La boucle while

Cette instruction signifie tant que. Elle permet de répéter l'exécution d'une instruction de boucle I :

Syntaxe :

while B do I;

B est une expression booléenne.

(*) B est évaluée. Si B est vraie, alors I est exécutée, et on recommence depuis (*).

Remarques :

- Les variables de l'expression B doivent être initialisées avant le while, pour que au premier passage B puisse être évaluée.
- Le while continue de boucler tant que B n'est pas faux. Pour éviter une boucle infinie, qui < plante > le programme, il faut obligatoirement que dans I il y aie une sous-instruction rendant B faux à un moment donné.

Exemple :

Programme calculant la somme des nombres de 1 à 100.

```
PROGRAM Somme;
VAR
    s, k : integer;
BEGIN
  s := 0; k := 1;
  while k <= 100 do
  begin
    s := s + k;
    k := k + 1;
  end;
  writeln (s);
END.
```

On se sert souvent d'un booléen dans une boucle while :

```
continuer := true;
while (k <= 100) and continuer do
begin
    { ... }
    if ( ... ) then continuer := false;
end;
```

II.7.3.2 La boucle repeat

Cette instruction signifie répéter . . . jusqu'à. Elle permet comme le while de répéter l'exécution d'une instruction de boucle I :

Syntaxe :

```
repeat
  I;
until B;
```

B est une expression booléenne.

(*) I est exécutée, puis B est évaluée. Si B est vraie, alors on s'arrête, sinon on recommence depuis (*).

Différences avec while :

- L'instruction I est exécutée au moins une fois.
- Le test B étant évalué après I, B peut être affecté dans I. Pour le while il faut avoir initialisé B avant.
- Pas besoin d'encadrer un groupe d'instructions par un begin end, le repeat until joue déjà ce rôle.

Exemple :

Le while de Somme s'écrit avec un repeat :

```
s := 0; k := 1;
repeat s := s + k; k := k + 1; until k > 100;
```

Traduction d'une boucle while B do I; avec un repeat :

```
if B then
  repeat
    I;
  until not B;
```

II.7.3.3 La boucle for

Cette instruction signifie pour. Elle permet de répéter l'exécution d'une instruction de boucle I :

Syntaxe :

For k := E1 to E2 do I;

k est le compteur de boucle, E1 et E2 sont les bornes inférieures et supérieures.
E1 et E2 sont des expressions ordinales, du même type que la variable k.
E1 et E2 sont d'abord évaluées, puis k prend la valeur E1. (*) Si k <= E2, alors I est exécutée, puis k est incrémenté de 1, et on recommence depuis (*).

Pour avoir **une boucle décroissante**, on écrit :

for k := E2 downto E1 do I;

On peut écrire une boucle for k := E1 to E2 do I; avec un while :

```
k := E1; { init de k }
m := E2; { on evalue E2 une fois pour toutes }
while k <= m do
begin
   I;
   k := k+1;
end;
```

On en déduit l'écriture d'une boucle for k := E1 to E2 do I; avec un repeat :

```
k := E1; { init de k }
m := E2; { on evalue E2 une fois pour toutes }
if k <= m then
  repeat
     I;
     k := k+1;
  until k > m;
```

Remarques :

- L'instruction de boucle I n'est pas exécutée du tout si E1 > E2.
- Modifier pendant la boucle la valeur de E1 ou E2 n'a pas d'effet.
- Il est totalement interdit de modifier la valeur du compteur k dans le corps de la boucle.
- L'incrément de 1 n'est pas modifiable (contrairement au Basic avec step).
- A la fin de l'exécution de la boucle, la variable k redevient indéterminée : elle a une valeur qui dépend du compilateur. Par exemple sous Delphi, elle vaut E2+1, et sous Turbo Pascal 7.0, elle vaut E2.

Exemple :

On peut aussi imbriquer les boucles.

```
PROGRAM table_multiplication;
VAR
   i, j : integer;
BEGIN
   for i := 1 to 10 do
   begin
      for j := 1 to 10 do write (i*j : 3);
      writeln;
  end;
END.
```

II.7.3.4 Choix de la boucle

La règle est simple (l'apprendre par cœur) :
Si le nombre d'itérations est connu a priori, alors on utilise un For.
Sinon : on utilise le Repeat (quand il y a toujours au moins une itération), ou le While (quand le nombre d'itérations peut être nul).

II.8 Comment réparer les erreurs dans les programmes ?

Les programmeurs sont familiers avec les erreurs dans les programmes.
Alternativement appelée « bogues », les erreurs de programmation surface souvent tout au long des phases de développement de la conception d'un logiciel. Il existe trois types d'erreurs de programme communément rencontrées et corrigé par un programmeur. Les erreurs de syntaxe résultent de l'écriture des codes qui ne sont pas conformes à la grammaire langue de programmation. Ils sont faciles à repérer et à corriger depuis le logiciel de développement permettra d'identifier les cours de compilation. Les erreurs logiques ou sémantiques se produisent quand un programmeur écrit un algorithme tort ou formule et le résultat attendu est incorrect.
Les erreurs d'exécution de temps montrent généralement pendant l'exécution du programme lorsque des données entrées par l'utilisateur qui sont impossibles à exécuter.

Fin du chapitre.

Chapitre 3 :
Les variables Indicées

III.1. Introduction :

Jusque-là, nous n'avons que très peu de moyens de représenter les données d'un programme. Une donnée en mémoire est soit une variable de type Integer, soit une variable de type Double, String ou Boolean. Pour chaque donnée du programme, nous avons donc ainsi une unique variable.
En réalité, comme nous allons le voir dans ce chapitre (cours concernant les types structurés), une même variable peut contenir un ensemble de données. En particulier, lorsque toutes ces données sont de même type, il est possible de les stocker dans une variable de type tableau.

III.1. Les tableaux à une dimension

III.1.1. Exemple introductif

Supposons qu'on ait le problème suivant : on voudrait écrire une procédure permettant de multiplier dix nombres entiers par 3.
Les dix entiers sont des variables globales. Il faut donc tout d'abord déclarer dix variables globales :
Var N1, N2, ..., N10 : Integer;

Et la procédure de multiplication devrait comporter dix instructions :

```
Procedure MultiplierPar3 ();
Begin
N1 := N1 * 3;
N2 := N2 * 3;
.
.
N10 := N10 * 3;
End;
```

Pour dix entiers, cette manière de procéder est envisageable. Mais imaginez qu'on ait le même problème avec un million d'entiers !
Ne croyez pas, d'ailleurs, que le traitement d'un million d'entiers soit un problème totalement irréaliste, car les ordinateurs ont souvent à traiter des millions de nombres en même temps. En informatique, ce type de problème est tout à fait courant.
Malheureusement, on ne peut pas utiliser de boucle pour faire ceci, car on n'a aucun moyen de faire varier le nom d'une variable !
On aurait envie d'écrire quelque chose du genre :

```
Var i : Integer;
For i:=1 To 10 Do Var Ni : Integer;
Procedure MultiplierPar3 ();
Var i : Integer;
Begin
For i := 1 To 10 Do Ni := Ni * 3;
End;
```

en pensant que Ni serait successivement remplacé par N1, N2, … , N10.
Malheureusement, ce mécanisme de génération automatique de nom de variable n'est pas possible !
Alors, comment faire ?
La solution est d'utiliser un tableau.
Pour utiliser un tableau, il faut tout d'abord déclarer une variable de type tableau.
Avec l'exemple précédent, ce serait :

Var N : array [1 .. 10] Of Integer;

Cette écriture revient à déclarer dix entiers N1, N2, … , N10, mais elle est beaucoup plus concise. Lorsqu'un interpréteur ou un compilateur rencontre une déclaration de tableau dans un programme, il alloue de la mémoire pour l'ensemble du tableau. Dans notre exemple, il y aurait donc une allocation mémoire de 20 octets en supposant des entiers 16 bits :

Figure III.1 : Allocation mémoire d'un tableau

D'autre part, il devient à présent possible de multiplier les dix entiers du tableau par 3 avec une boucle For, comme ceci :

```
For i := 1 to 10 do
   N[i]=N[i]*3 ;
```

Contenu du tableau avant la boucle :

N[1]	N[2]	N[3]	N[4]	N[5]	N[6]	N[7]	N[8]	N[9]	N[10]
2	-3	4	-5	4	2	5	-1	7	11

Après la boucle :

Instruction	i	N[1]	N[2]	N[3]	N[4]	N[5]	N[6]	N[7]	N[8]	N[9]	N[10]
N[1] := N[1] * 3	1	6	-3	4	-5	4	2	5	-1	7	11
N[2] := N[2] * 3	2	6	-9	4	-5	4	2	5	-1	7	11
N[3] := N[3] * 3	3	6	-9	12	-5	4	2	5	-1	7	11
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
N[10] := N[10] * 3	10	6	-9	12	-15	12	6	15	-3	21	3

Et on écrit :

```
Procedure MultiplierPar3 ();
Var i : Integer;
Begin
For i := 1 To 10 Do
N[i] := N[i] * 3;
End;
```

L'écriture N[i] désigne le ième élément du tableau, autrement dit l'élément d'indice i du tableau N.
Vous constaterez qu'avec un million d'entiers, le code du programme n'est pas plus long :

```
Var N : array [1 .. 1000000] of Integer;
Procedure MultiplierPar3 ();
Var i : Integer;
Begin
For i := 1 To 1000000 Do N[i] := N[i] * 3;
End;
```

III.1.2. Le cas général

En Pascal, la déclaration d'un tableau s'écrit :

Var nom du tableau : array [indice min .. indice max] Of type des éléments;

où indice min et indice max sont deux entiers qui représentent respectivement le plus petit indice et le plus grand indice des éléments du tableau.

Le type des éléments peut varier d'un tableau à l'autre, mais tous les éléments d'un même tableau sont forcément du même type.

Voici, par exemple, la déclaration d'un tableau de chaînes de caractères :

Var Prenom : array [1..1000] Of String;

On pourra ensuite affecter une valeur à un élément du tableau, tout comme on affecte une valeur à une variable.

Par exemple l'instruction :

Prenom[3] := 'Jeanne';

affecte la chaîne de caractères 'Jeanne' à l'élément d'indice 3 du tableau **Prenom**.

III.1.3. Utilisation de constantes pour définir la dimension d'un tableau

Supposons que vous ayez à modifier un programme très long (plusieurs milliers de lignes, par exemple), qui utilise un tableau de T de dix entiers déclaré par :

Var T : array [1 .. 10] Of Integer;

On vous demande de changer la dimension du tableau T de 10 à 20.

Supposons que dans ce programme figurent des centaines d'instructions faisant référence à la dimension du tableau T, c'est-à-dire au nombre 10.

Dans toutes ces instructions, il va donc falloir remplacer le nombre 10 par le nombre 20.

Les constantes servent en particulier à éviter ce genre de manipulation longue et fastidieuse.

Supposons à présent que le programme initial ait été écrit en utilisant une constante DIMENSION, qui représente le nombre d'éléments du tableau T, déclarée comme suit:

Const DIMENSION = 10;

et que le tableau soit déclaré comme suit :

Var T : array [1..DIMENSION] of Integer;

Si le développeur qui a écrit le programme a bien fait les choses, il aura fait référence à la constante DIMENSION (et non pas 10 !) dans toutes les instructions utilisant la dimension du tableau T.

Un programme écrit de cette manière est beaucoup plus facile à mettre à jour. En effet, pour changer la dimension du tableau de 10 à 20, il vous suffit à présent de modifier une seule ligne de programme : la déclaration de la constante DIMENSION.

C’est-à-dire que vous allez la remplacer par :

Const DIMENSION = 20;

et tout le reste fonctionnera tout seul !

Vous vous demandez peut-être pourquoi on n'aurait pas pu utiliser une variable à la place de la constante. C’est à- dire déclarer :

Var DIMENSION : Integer;

T : array [1..DIMENSION] of Integer;

puis faire DIMENSION := 10; au début du programme.

Ceci ne fonctionne pas !

Pourquoi ?

Parce que la dimension d'un tableau ne peut pas être modifiée pendant l'exécution d'un programme.

En effet, un tableau est un objet statique, c'est-à-dire que l'espace mémoire qui lui est alloué ne peut être modifié en cours d'exécution.

Cela présente un certain nombre d'inconvénients. En particulier, si vous ne connaissez pas a priori la taille de tableau qu'il vous faudra pour résoudre un problème particulier, la seule solution que vous avez est de prévoir large, de manière à ce que vous ne risquiez pas d'avoir un dépassement de capacité (taille de tableau insuffisante pour enregistrer toutes les données).

Nous verrons ultérieurement qu'il existe d'autres manières de représenter les informations en mémoire (on parle de structures de données), qui permettent d'éviter ce problème.

III.1.4. Tableaux remplis partiellement

III.1.4.1. Représentation

Nous avons vu qu'un tableau est un objet statique, c'est-à-dire que sa dimension ne peut pas être modifiée pendant l'exécution d'un programme.

Il faut donc toujours prévoir large, c'est-à-dire déclarer un tableau de taille suffisante pour tous les cas de figure.

Cela signifie que l'on va, en général, remplir les tableaux partiellement jusqu'à un certain indice définissant la fin des données.

Pour représenter un tableau rempli partiellement, il faut donc nécessairement une variable entière qui contiendra à tout moment l'indice de fin des données.

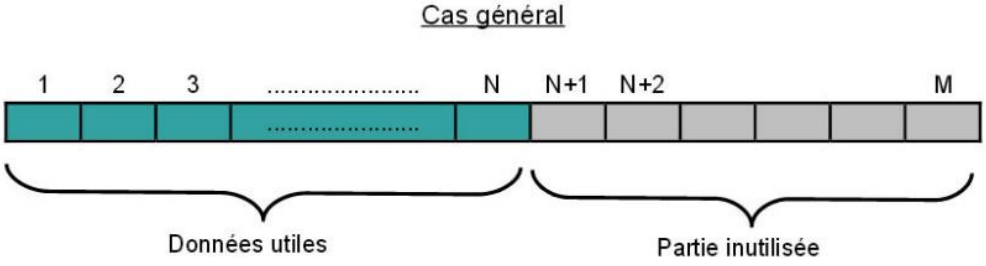
Pour fixer les idées, prenons une variable T pour le tableau et une variable N pour l'indice de fin. On aura donc les déclarations suivantes :

Const M = Nombre maximum d'éléments;

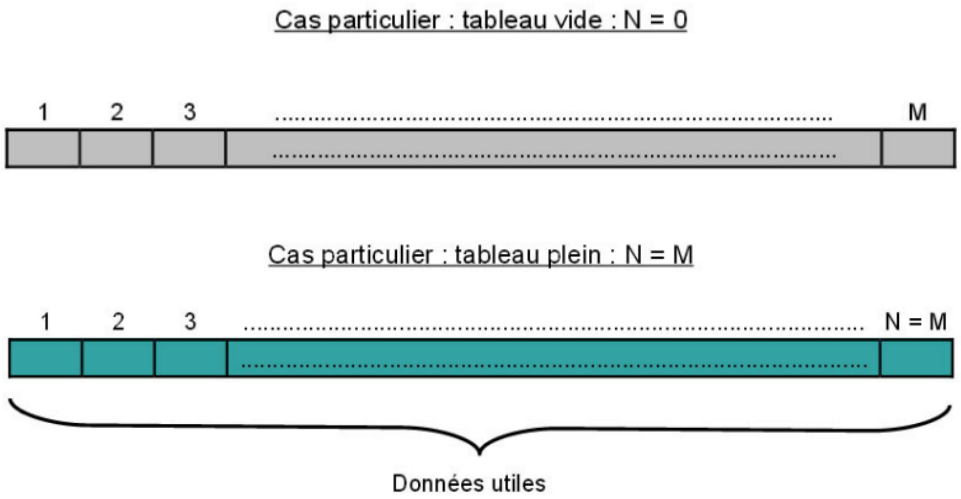
Var N : Integer;

T : array [1..M] of Type;

Les données utiles sont stockées dans les N premiers éléments du tableau :



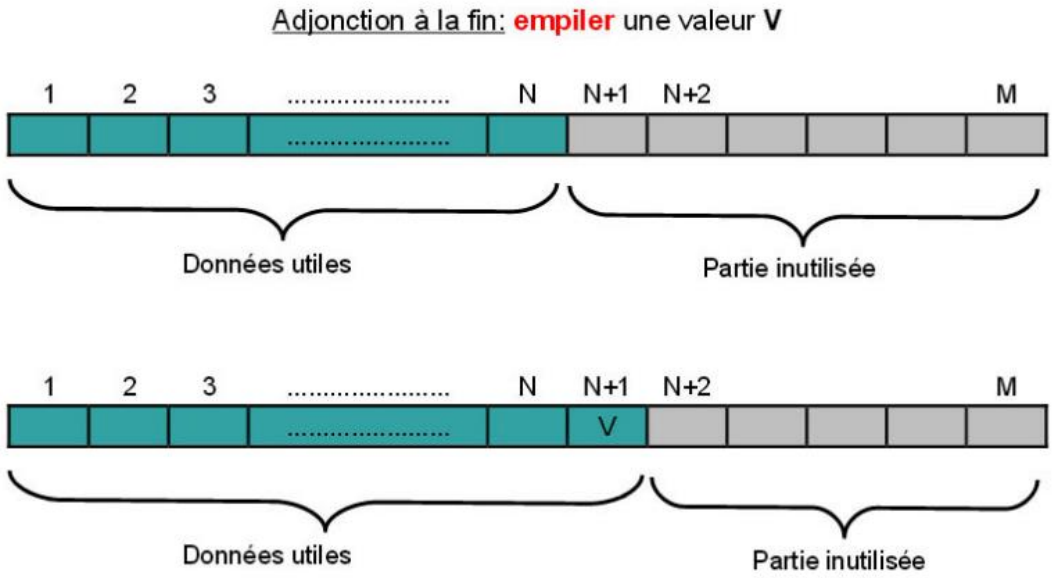
Le tableau est « vide » lorsque N vaut 0 et plein lorsque N est égal à M :



III.1.5. Adjonction d'une valeur

III.1.5.1. Adjonction à la fin

Pour ajouter une valeur V à un tableau rempli partiellement, le plus simple est de l'ajouter à la fin. On appelle cette opération **empiler**. L'élément qui suit le dernier prend la valeur V et l'indice de fin est incrémenté :

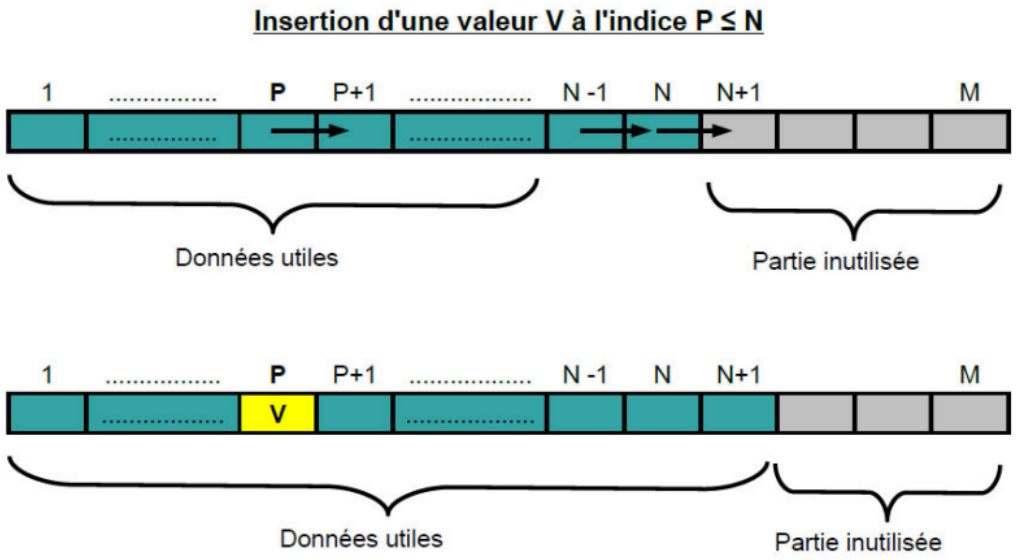


T [N+1] := V;
N := N+1;

III.1.5.2. Insertion

Insérer une valeur dans un tableau est une opération plus complexe. Supposons que l'on souhaite insérer une valeur V à l'indice P (P ≤ N) du tableau. Il faut d'abord « faire de la place » pour cette nouvelle valeur, c'est-à-dire décaler toutes les valeurs entre les indices P et N, d'un cran vers la droite. La valeur V peut

ensuite être affectée à l'élément d'indice P. Puis, comme on a ajouté une valeur, l'indice de fin augmente de 1 :

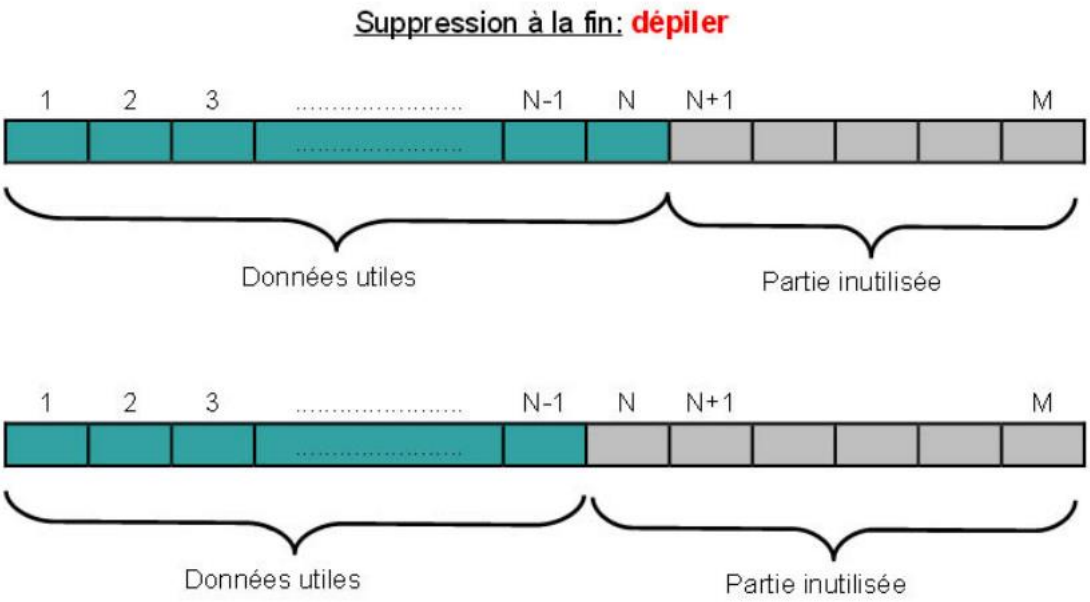


For i := N DownTo P Do T[i+1] := T[i];
T[P] := V;
N := N+1;

III.1.6. Suppression d'un élément

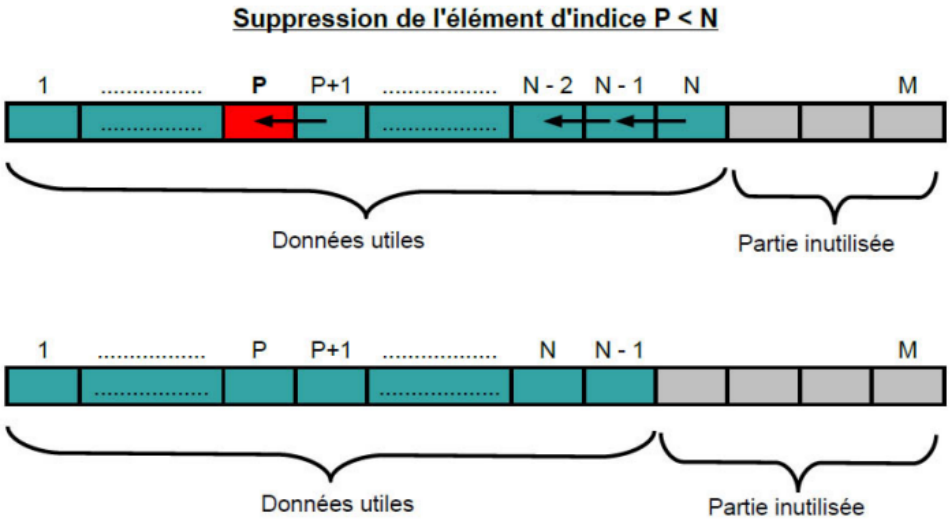
III.1.6.1. Suppression du dernier élément

Le dernier élément d'un tableau est le plus simple à supprimer ; pour cela, il suffit de dépiler, c'est-à-dire de décrémenter l'indice de fin de 1 :



III.6.1.2. Suppression d'un élément quelconque (différent du dernier)

Supprimer l'élément d'indice P (P < N) signifie décaler toutes les valeurs qui suivent P d'un cran vers la gauche, puis décrémenter l'indice de fin :



For i := P To N-1 Do T[i] := T[i+1];

N := N-1;

III.1.7. Notion de pile

Un tableau rempli partiellement sur lequel on ne fait que des opérations empiler et dépiler est une structure de données fondamentale de l'informatique que l'on appelle une pile (stack en anglais).
En particulier, la gestion mémoire des variables locales et des paramètres des sous-programmes utilise une pile : les valeurs des variables sont empilées lorsque l'on rentre dans le sous-programme et dépilées à la sortie.

III.2. Les tableaux à deux dimensions

III.2.1. Déclaration en Pascal

var T : array [1..3,1..4] of integer;

T [i , j] est l'élément de T se trouvant à la ligne i et à la colonne j.

Exemple : T [2, 3] := 10;

	1	2	3	4
1				
2				
3				

III.2.2. Traitement de tous les éléments

Principe :

```
for i := 1 to 3 do
begin
for j := 1 to 4 do
begin
Traitement de l'élément T[ i, j ]
end
end
```

Exemple :

```
for i := 1 to 3 do
begin
for j := 1 to 4 do
begin
T [ i , j ] := j + 3 * (i - 1) ;
end
end
```

	1	2	3	4
1	**1**	**2**	**3**	**4**
2	**5**	**6**	**7**	**8**
3	**9**	**10**	**11**	**12**

III.2.3. Traitement d'une ligne

Principe : traitement de la ligne i.

```
for j := 1 to 4 do
begin
Traitement de l'élément T[ i, j ]
end
```

Exemple : traitement de la ligne 2 :

```
for j:= 1 to 4 do
begin
T [2, j ] := 2 * j ;
End
```

	1	2	3	4
1				
2	**2**	**4**	**6**	**8**
3				

III.2.4. Traitement d'une colonne

Principe : traitement de la colonne j.

```
for i := 1 to 3 do
begin
Traitement de l'élément T[ i, j ]
end
```

Exemple : traitement de la colonne 3 :

```
for i := 1 to 3 do
begin
T [ i, 3 ] := 4 - i ;
End
```

	1	2	3	4
1			**3**	
2			**2**	
3			**1**	

Fin du chapitre.

Conclusion générale

Conclusion générale :

Au total, l'informatique a pris son envol avec l'avènement des réseaux sociaux qui ont révolutionné le monde. Nous sommes actuellement à l'ère numérique et l'informatique joue un rôle important tel qu'il existe dans notre vie quotidienne. Elle est imposée dans nos foyers par de nombreux appareils informatiques comme les téléphones, les télévisions, les appareils électroménagers et bien d'autres, et dans nos écoles, donnant aux enseignants de plus en plus de moyens pour améliorer leur enseignement, mais l'informatique est également présente dans les entreprises, leur permettant de grandir plus rapidement et à moindre coût.

L'informatique, comme toute technologie, présente des avantages et des inconvénients. C'est pourquoi beaucoup de recherche et développement sont faits dans ce domaine pour réduire les inconvénients.

En conclusion, l'informatique fait partie intégrante de la vie moderne. Cela a changé notre façon de vivre, de travailler et de communiquer. Les éléments constitutifs d'un ordinateur sont le matériel et les logiciels, et une encyclopédie est un outil utile pour gérer l'information. Charles Babbage est considéré comme le père de l'informatique et Louis Fein a inventé le terme "technologie de l'information" en 1959. À mesure que la technologie progresse, l'informatique jouera sans aucun doute un rôle encore plus important dans la formation de notre monde.

Bonne lecture.