Informatique

TELLI AbdelmoutiA
Maitre de Conférences Classe A
Département d'informatique
Université de Biskra

2022-2023

Domaine : Sciences et Technologies

Intitulé du LICENCE ACADEMIQUE (Filière) : Hydraulique

Spécialité : Hydraulique

Semestre : S1

Intitulé de l'UE : UEM 1.1

Intitulée de la matière : Informatique 1

Crédits : 4

Coefficients : 2

Objectif et recommandations : 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.

Connaissances préalables recommandées : Notions élémentaires de la technologie du Web.

Mode d'évaluation : Examen 60% + Contrôle Continu 40%

TP Informatique 1 : 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 d'initiation et de familiarisation avec la machine informatique d'un point de vue matériel et systèmes d'exploitation (exploration des différentes fonctionnalités des OS)
- TP d'initiation à l'utilisation d'un environnement de programmation (Edition, Assemblage, Compilation, etc.)
- TP d'application des techniques de programmation vues en cours.

Références :

- John Paul Mueller et Luca Massaron, Les algorithmes pour les Nuls grand format, 2017.

Table des matières

Chapitre 1

Introduction à l'informatique

> The hero is a remedy for the natural weakness of children, the relational wound of adults or the historic humiliation of a nation.
>
> Boris Cyrulnik.

Introduction

L'informatique est l'art, la technique ou la science qui consiste à manipuler des informations à l'aide d'un outil, l'ordinateur. L'informatique a pour objet de définir des algorithmes qui permettent de modifier la vision que l'on a d'un problème, ou d'extraire d'une grande quantité d'informations mal structurées, de nouvelles connaissances plus utiles.

Les outils de l'informatique sont les ordinateurs. Actuellement, on utilise presque exclusivement des ordinateurs digitaux et qui fonctionnent selon les principes de la machine de **Von Neumann**. Cette machine comprend deux parties, une unité logique et arithmétique banalisée et un magasin ou mémoire qui contient des programmes et des données. Un programme décrit les opérations logiques à réaliser sur les données.

1.1 Définition de l'informatique

En anglais : Informatics, Computer science, Computer engineering et Information Technology.

Le terme ***informatique*** désigne une discipline née avec l'ordinateur.

Informatique est un nom féminin signifie :

1. Science du traitement automatique et rationnel de l'information considérée comme le support des connaissances et des communications.

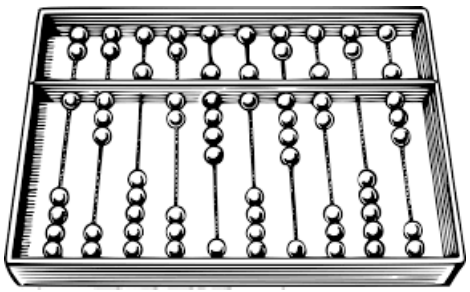
2. Ensemble des applications de cette science, mettant en œuvre des matériels et des logiciels.

L'informatique s'intéresse à la mise en œuvre de méthodes scientifiques pour traiter l'information au moyen d'ordinateurs. Il désigne l'ensemble des sciences et techniques en rapport avec le traitement automatique de l'information.

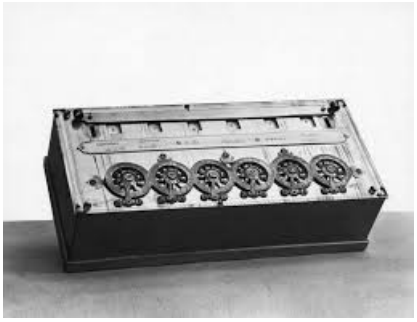
1.2 Evolution de l'informatique/ordinateurs

Avant de créer l'informatique, soit des machines capables de réaliser des calculs automatiquement, l'Homme a créé des objets qui l'ont aidé à réaliser des calculs. Ce sont les premiers calculateurs :

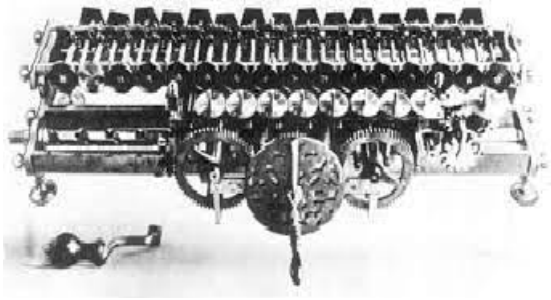
— **Le boulier (-2000)** : Le boulier ou abaque a été utilisé en Mésopotamie.



— **La Pascaline (1642) :** Blaise Pascal crée une machine capable d'additionner et de soustraire.

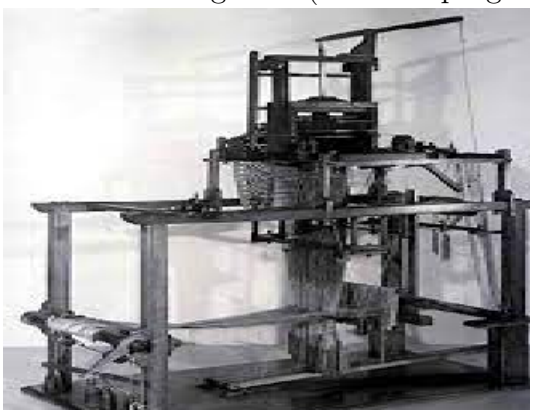


— **La machine à calculer REPLICA de Leibniz (1673)** : Leibniz qui s'est inspiré de la Pascaline invente une machine capable de multiplier et de diviser.

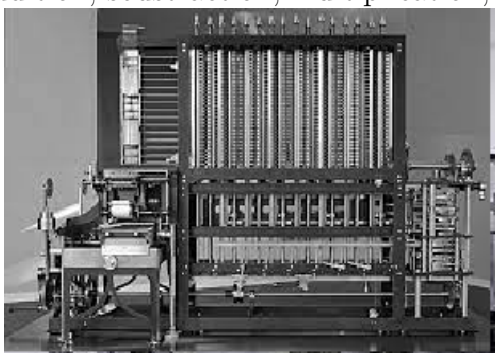


Les calculateurs sont passés du tout mécanique au tout électronique. Les capacités de calculs ont augmenté et la fiabilité des machines aussi :

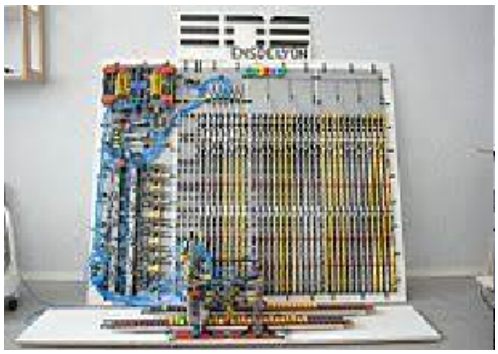
— **Le métier à tisser Jacquard (1800) :** Joseph-Marie Jacquard met au point un métier à tisser qui utilise des cartons perforés pour commander les mouvements des aiguilles (machine programmée).



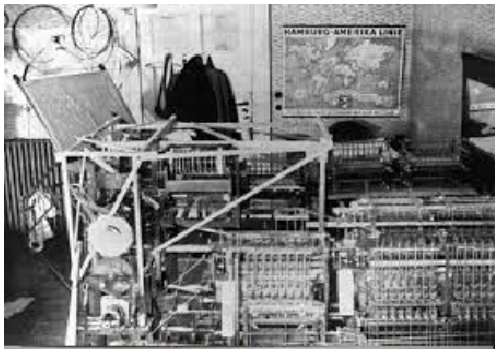
— **La machine à calculer de Charles Babbage (1834) :** s'inspire du métier à tisser de Jacquard, pour élaborer une machine qui évalue les fonctions : addition, soustraction, multiplication, et division.



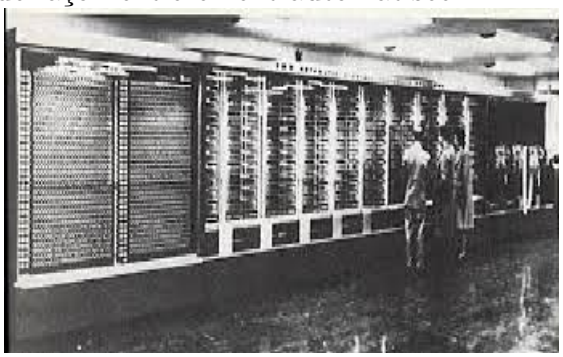
— **La machine de Alan Turing (1936) :** définissant les bases théoriques de la programmation sur des machines capables d'effectuer des calculs.



— **Les Z1, Z2 et Z3 de Konrad Zuse (1938-1941) :** Le Z1, créé par l'Allemand Konrad Zuse. C'est le premier calculateur programmable fonctionnel. Les informations sont stockées sur des cartes perforées.

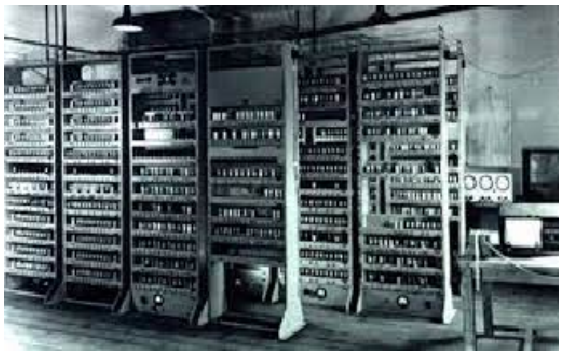


— **Le MARK1 (1943) :** créé par Howard Aiken en collaboration avec IBM à l'université d'Harvard. Cette machine calcule 5 fois plus vite que l'homme de façon entièrement automatisée.



— **L'ENIAC (1945) :** Créé par John William Mauchly et J. Presper Eckert, l'ENIAC devient le premier ordinateur ne comportant plus aucune pièce mécanique. Il est composé de 18 000 tubes à vide, s'étend sur plus 160 m2 et opère en décimal.

— **L'EDVAC (1946) :** Electronic Discrete Variable Automatic Computer est une évolution de l'ENIAC basée sur l'architecture de John von Neumann.



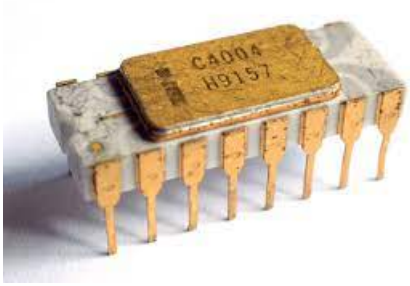
— **Le premier compilateur (1951) :** Grace Hopper développe le premier compilateur sur une évolution de l'EDVAC : l'UNIVAC. Un compilateur est un système qui traduit automatiquement le langage des informaticiens en langage machine.

— **La première souris (1963) :** par Douglas Engelbart au Stanford Research Institute. Elle est équipée de deux roues fixées sur deux capteurs et d'un bouton poussoir.

— **Programma 101 (1965) :** le Programma 101 de la société italienne Olivetti est le premier ordinateur personnel numérique et programmable. Il est inventé par l'ingénieur Pier Giorgio Perotto.

— **UNIX 1969 :** Ken Thompson développe la première version d'un système d'exploitation en ligne de commande (UNIX).

— **Premier microprocesseur (1971) :** La société Intel commercialise son premier microprocesseur 4004.



— Premier système d'exploitation graphique La société XEROX (1973) : sa station de travail Alto équipée d'un système d'exploitation appelé Alto OS. Il possède la première barre d'outils.

— **Premier ordinateur Apple (1976) :** Steve Jobs, Steve Wozniak et Ronald Wayne commercialisent le premier micro-ordinateur Apple 1.

— **Premier PC (1981) :** IBM lance le PC (Personal Computer).

— **Premier GUI de Macintosh (1984) :** les systèmes Macintosh d'Apple Computer sont les premiers à être dotés d'une interface graphique : Lisa OS. Au lieu d'avoir à taper des lignes de commandes au clavier, l'utilisateur peut maintenant se servir d'une souris et cliquer sur des icones.

— **Microsoft Windows 1.0 (1985) :** Après le succès de ses logiciels sur PC et de son système d'exploitation MS-DOS, Microsoft lance Microsoft Windows 1.0

— **Linux (1991) :** Linus Torvalds développe enfin le noyau "Linux" au projet GNU lancé en 1983 par Richard Stallman.

La véritable évolution informatique de la fin du $20^{ème}$ siècle est surtout la transformation de l'ordinateur vers de nouveaux objets (Console de jeux, smartphone, tablette tactile...).

1.3 Les systèmes de codage des informations

Bit : est l'information traitée par l'ordinateur représentée par des nombres. Est un la plus petite unité qui prend deux valeurs soit 0 ou 1.

Octet (Byte) : est une unité de mesure composée de 8 bits :

— 1Ko (Kilo octet) =1K byte=1024 octet=2^{10} octet

— 1Mo (Mega octet) =1M byte=2^{10} Ko =2^{20} octet

— 1GO (Gega octet) =1G byte=2^{10} Mo=2^{30} octet

— 1To (Tera octet) =1T byte=2^{10} Go=2^{40} octet

— 1Po (Peta octet) =1P byte=2^{10} To=2^{50} octet

— 1Eo (Exa octet) =1E byte=2^{10} Po=2^{60} octet

— 1Zo (Zetta octet) =1Z byte=2^{10} Eo=2^{70} octet

Bps (bit/seconde) : unité de mesure de la vitesse des communications.

Hertz : unité de mesure de fréquence (événements par seconde).

1.3.1 Système de Numération

Pour qu'une information numérique soit traitée par un circuit, elle doit être mise sous forme adaptée à celui-ci. Pour cela, il faut choisir un système de numération de base B. De nombreux systèmes de numération sont utilisés en technologie numérique. Les plus utilisés sont les systèmes : Décimal (base 10), Binaire (base 2), Tétral (base 4), Octal (base 8), Hexadécimal (base 16).

Tout nombre N peut se décomposer en fonction des puissances entières de la base de son système de numération. Cette décomposition s'appelle la forme polynomiale du nombre N et qui est donnée par :

$$N = a_n B^n + a_{n-1} B^{n-1} + a_{n-2} B^{n-2} + ... +_{a2} B^2 +_{a1} B^1 + a_0 B^0$$

B : Base du système de numération, a_i : un chiffre parmi les chiffres de la base du système de numération, et i : rang du chiffre a_i

La base de numération Décimal

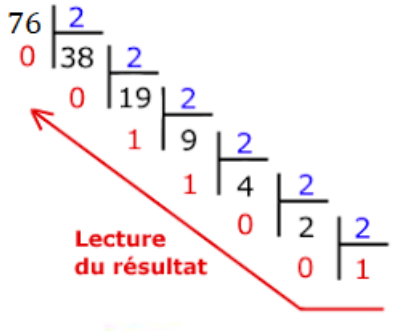
C'est le système de numération usuel dans la vie quotidienne. Dans ce système , tout nombre N est exprimé à partir des dix chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Par exemple : 1356 =10 +3*10 +5*10 +6*10.

Pour la conversion du système Décimal vers une base quelconque, il faut faire des divisions successives par B et retenir à chaque fois le reste jusqu'à l'obtention à un quotient inférieur à la base B, dans ce cas le nombre s'écrit de la gauche vers la droite en commençant par le dernier quotient allant jusqu'au premier reste.

La base de numération Binaire

Ce système de numération ne prennent que deux états 0 ou 1.

Pour passer du décimal vers binaire, il suffit de faire la division le nombre décimale par 2. Par exemple : $(76)_{10}$=$(1001100)_2$



Pour la conversion binaire vers décimal :

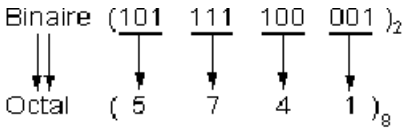
$$(1001100)_2 = 0*2^0 + 0*2^1 + 1*2^2 + 1*2^3 + 0*2^4 + 0*2^5 + 1*2^6 = (76)_{10}$$

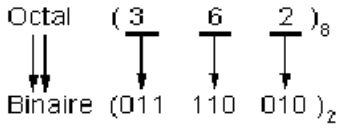
La base de numération Octal

Ce système est basé sur 8 symboles (0, 1, 2, 3, 4, 5, 6, 7).

Pour passer du décimal vers octal, il suffit de faire la division le nombre décimale par 8. Par exemple : $(262)_{10} = (406)_8$

Pour convertir un code binaire en octal il suffit de décomposer le code en groupes de 3 bits.

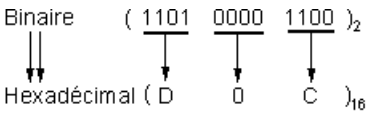


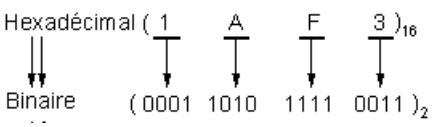


La base de numération hexadécimal

Ce système est représenté sur 16 symboles (0-9,A,B,C,D,E,F).

Pour passer du décimal vers hexadécimal, il suffit de faire la division le nombre décimale par 16. Par exemple : $(31)_{10} = (1F)_{16}$. Donc, Pour convertir un code binaire en hexadécimal, il suffit de décomposer le code binaire en groupes de 4 bits.





1.4 Fonctionnement d'un ordinateur

Les deux principaux constituants d'un ordinateur sont la mémoire principale et le processeur :

1.4.1 La mémoire Centrale

Elle contient les instructions du ou des programmes en cours d'exécution et les données associées à ce programme. Elle se décompose souvent en :

— **La mémoire morte (ROM = Read Only Memory :)** chargée de stocker le programme. C'est une mémoire à lecture seule.

— **La mémoire vive (RAM = Random Access Memory) :** chargée de stocker les données intermédiaires ou les résultats de calculs. On peut lire ou écrire des données dedans, ces données sont perdues à la mise hors tension.

Les disques durs, clés USB, CDROM, etc. sont des périphériques de stockage et sont considérés comme des mémoires secondaires.

1.4.2 Le processeur

Le processeur appelé ausi CPU (Central Processing Unit) ou encore MPU (Micro Processing Unit) pour les microprocesseurs. Un microprocesseur n'est rien d'autre qu'un processeur dont tous les constituants sont réunis sur la même puce électronique (pastille de silicium), afin de réduire les coûts de fabrication et d'augmenter la vitesse de traitement. Les microordinateurs sont tous équipés de microprocesseurs. L'architecture de base des processeurs équipant les gros ordinateurs est la même que celle des microprocesseurs.

Un ordinateur contient un circuit, le processeur, qui permet d'effectuer de petits traitements de base qu'on appelle instructions et qui sont la base de tout ce qu'on trouve sur un ordinateur. Ce traitement est mené à l'aide des instructions.

Une instruction informatique est une commande unique. Elle est l'opération élémentaire que le processeur peut accomplir. Les instructions sont stockées dans la mémoire principale, en vue d'être traitée par le processeur.

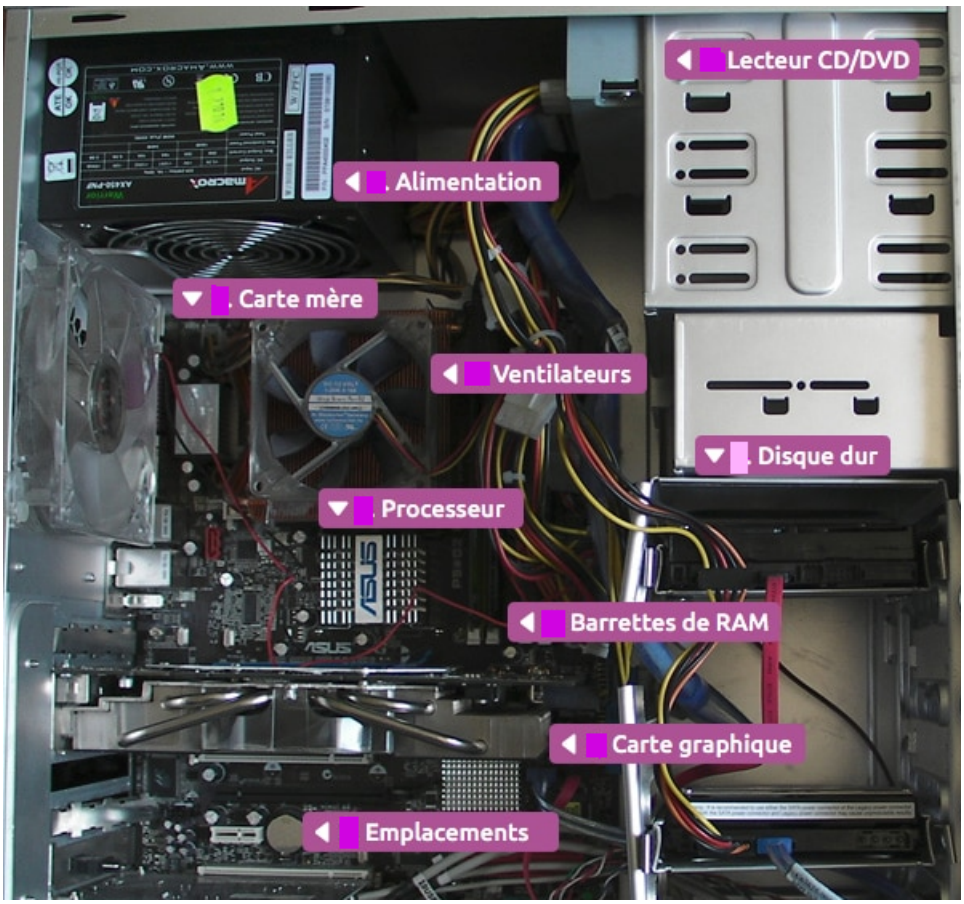
Le jeu d'instructions est l'ensemble des opérations élémentaires qu'un processeur peut accomplir. Le jeu d'instruction d'un processeur détermine ainsi son architecture. Le processeur travaille effectivement grâce à un nombre limité de fonctions, directement câblées sur les circuits électroniques. La plupart des opérations peuvent être réalisées à l'aide de fonctions basiques. Certaines architectures incluent néanmoins des fonctions évoluées courantes dans le processeur.

1.5 Partie matériel d'un ordinateur

En plus de les mémoires (RAM et ROM) et le processeur et son ventilateur, un ordinateur ou PC (Personal Computer) possède des matériels qui se trouve à l'intérieur de l'ordinateur, et des périphériques externes qui sont reliés par des câbles ou des moyens de communication sans fil.

1.5.1 Les composants internes

Ce sont des composants d'ordinateurs qui sont ici décrits le plus simplement possible pour vous permettre de vous familiariser avec le matériel informatique de votre PC :



1. **L'alimentation :** Elle transforme et fournit l'énergie nécessaire à la précieuse carte mère, sur laquelle est connecté un bon nombre d'éléments.
2. **La carte mère :** C'est le composant principal de l'ordinateur, car elle est vissée au boîtier de PC, de plus elle possède les connecteurs (slots) pour accueillir des dizaines de composants et périphériques en plus des éléments indispensables.
3. **Le disque dur :** Utilisé pour stocker du contenu et des données numériques sur les ordinateurs.
4. **Le lecteur/graveur CD/DVD :** Est un périphérique permettant de lire ou d'écrire sur un cédérom ou un DVD-Rom.
5. **La carte graphique :** Permet de produire une image affichable sur un écran.
6. **La carte sons :** (carte audio) Permet de gérer les entrées et les sorties des sons de l'ordinateur.
7. **La carte réseau :** Est l'interface entre votre ordinateur et le réseau. Elle reçoit les données émises par l'ordinateur et les transfère vers un autre appareil

Il y a des autres composants tel que :

— **Les périphériques d'entrée :** permettant à un utilisateur extérieur de fournir des informations (données/programmes) à la machine sous forme numérique omme : souris, clavier, scanner, joystick, appareil photo numérique, camescope numérique...

— **Les périphériques de sortie** : Permettant de visualiser ou de transmettre des données internes à l'extérieur comme : écran, imprimante, IPod, ...

— **L'unité Centrale :** Est le boitier contenant tout le matériel électronique permettant à l'ordinateur de fonctionner.

1.6 Partie système d'un ordinateur

Elle est appleé aussi la partie logicielle qui est constituée de l'ensemble des codes stockés dans sa mémoire, et plus particulièrement de ses programmes.

1.6.1 Langages d'assemblage

Ecrire un programme, c'est donc écrire une suite d'instructions élémentaires s'enchaînant les unes après les autres pour réaliser un traitement sur des données. Dans le disque dur et la mémoire centrale, ces programmes sont codés sous forme de bits à la façon du mini-programme dont l'exécution est détaillée. il y a ainsi presque autant de langages d'assemblage différents que de microprocesseurs.

1.6.2 Langages évolués

Il sont aussi appelés des programmes exécutables, puisqu'ils sont directement prêts à être exécutés. La programmation dans un tel langage nécessite de connaître l'architecture matérielle de l'ordinateur sur lequel il s'exécutera.

L'utilisation de compilateurs a permis la définition de langages de programmation de haut niveau. Dans de tels langages, on peut s'abstraire de la connaissance matérielle de l'ordinateur sur lequel s'exécutera le programme pour se concentrer sur sa seule logique.

1.6.3 Les systèmes d'exploitation

Parmi les logiciels les plus usuels, il en est un qui est devenu indispensable à tous les ordinateurs actuels : c'est le système d'exploitation ou système opératoire (Operating System).

Le système d'exploitation d'un ordinateur est en quelque sorte son gestionnaire central, son chef d'orchestre. Quand on allume un ordinateur, on provoque automatiquement la recopie du système d'exploitation du disque dur vers la mémoire centrale. Ses rôles principaux sont les suivants :

- Fournir une interface entre l'ordinateur et l'utilisateur pour permettre à ce dernier de donner des ordres à la machine.
- Gérer les ressources de l'ordinateur, les mémoires, et les périphériques.
- Permettent à plusieurs programmes de s'exécuter en même temps (multitâches).
- Indépendant du matériel, masquer les particularités de la machine en substituant aux ressources physiques des abstractions.

Le système d'exploitation est la couche logicielle de base qui s'intercale toujours entre l'utilisateur et le matériel. Les plus couramment installés sur les ordinateurs actuels sont :

1. **MS-DOS (MicroSoft Disk Operating System) :** est le système d'exploitation le plus connu, sa version la plus commercialisée est celle de Microsoft. MS-DOS a vu le jour en 1981 lors de son utilisation sur un IBM PC. Il constituait la base des systèmes Windows de Microsoft jusqu'à **Windows 3.1**. Les manipulations d'objets graphiques étaient en fait traduits en commandes MS-DOS.

2. **Windows 95, 98, XP,... :** sont les systèmes d'exploitation multitches de Microsoft ayant pris la place de MS-DOS. Il est au départ une interface graphique unifiée produite par Microsoft, qui est devenue ensuite une gamme de systèmes d'exploitation à part entière, principalement destinés aux ordinateurs compatibles PC.

3. **MacOS :** (Mac Intosh Operating System), est un système d'exploitation partiellement propriétaire développé et commercialisé par Apple depuis 1998, les outils d'interface graphiques (menus, fenêtres...). Les systèmes actuels de Mac sont en fait des variantes du système Linux.

4. **Linux/ GNU/Linux :** est un système d'exploitation, ou un ensemble de système d'exploitation, de type Unix. Sa particularité est d'être open source, développé de façon collaborative. Linux dispose de nombreuses distributions dont les plus connues sont : Ubuntu et Debian.

1.7 Conclusion

Dans ce chapitre. Nous avons détaillé plusieurs concepts informatiques. En particulier, son évolution et sa fonctionnalité, les systèmes de numérotation des données et le matériel et les logiciels du PC.

Bien que comportant beaucoup de détail soit nécessaire pour la bonne compréhension de l'algorithmique et de ses bases qui seront détaillées dans le chapitre suivant.

Références du chapitre

Brookshear, J. Glenn, Dennis Brylow, and S. Manasa. "Computer science : An overview." (2009).

Tanenbaum, Andrew S., and Albert S. Woodhull. Operating systems : design and implementation. Vol. 2. Englewood Cliffs : Prentice Hall, 1997.

Zhang, Jiajie, and Donald A. Norman. "A representational analysis of numeration systems." Cognition 57.3 (1995) : 271-295.

Chapitre 2

Notions d'algorithme et de programmes

La programmation peut être un plaisir ; de même que la cryptographie.
Toutefois, il faut éviter de combiner les deux. »

Kreitzberg et Sneidermann

Introduction

La résolution d'un problème avec un ordinateur nécessite l'utilisation d'un algorithme. Les ordinateurs ne peuvent exécuter que des ordres bien précis et sans ambiguïtés. Il faut trouver une méthode efficace de résolution d'un problème écrite en langage algorithmique. Don, on peut avoir plusieurs algorithmes pour la résolution du même problème.

Le but de ce chapitre est de présenter les éléments de base de l'algorithme et du langage C, sous leur forme la plus simple. Arrive au bout du chapitre, le lecteur sera capable d'écrire des programmes élémentaires.

2.1 Concept d'un algorithme

Le concept d'algorithme est le plus ancien, puisque 2500 ans avant notre ère, les comptables utilisaient déjà des algorithmes pour effectuer les quatre opérations, calculer des prêts, des héritages, ... et les arpenteurs pour calculer l'aire de surfaces agricoles.

La programmation structurées : décompose le problème complexe en sous problèmes et ces sous problèmes en d'autres sous problèmes jusqu'à obtenir des problèmes faciles a résoudre c'est-à-dire connus. On résout les sous problèmes simples sous forme d'algorithme puis on recompose les algorithmes pour obtenir l'algorithme global du problème de départ. Pour cela, on manipule des objets simples ou structurés.

Un objet va être caractérisé par :

— **Un identificateur :** ou un nom.

— **Un type :** entier, caractère, ... simple ou structuré. Un type détermine en particulier les valeurs possibles de l'objet et les opérations primitives applicable à l'objet.

— **Une valeur :** (contenu de l'objet) unique. Cette valeur peut varier au cours de l'algorithme ou d'une exécution à l'autre. Ces objets sont des variables. Dans les cas contraires (valeur fixe) ce sont des constantes.

Tous les objets manipulé par un algorithme doivent être clairement définis comme :

Les mots clefs

const : Pi=3.14 ;

var a, b : entier ;

x, y : caractère ;

Sachant que : a, b, x, y sont des identificateurs

2.2 Représentation en organigramme

Un algorithme est représenté graphiquement sous la forme d'un organigramme ou logigramme ou algorigramme. Cet organigramme ou logigramme ou algorigramme permet de décrire le fonctionnement du système automatisé plus facilement et plus clairement qu'avec du texte.

2.2.1 Définition d'organigrame

Un algorigramme, aussi appelé **organigramme** de programmation, est la représentation visuelle d'un algorithme. Il montre les enchaînements de décisions et d'opérations à faire pour un algorithme donné. Un algorithme est une suite de règles opératoires rigoureuses propre à un calcul. C'est l'étape précédant la programmation qui permettra de parler la même langue que l'ordinateur.

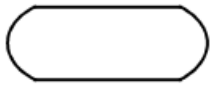
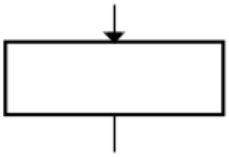
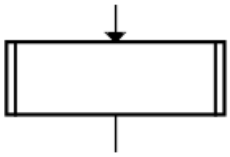
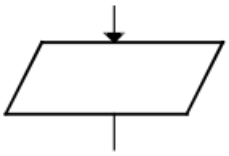
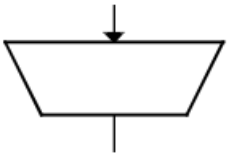
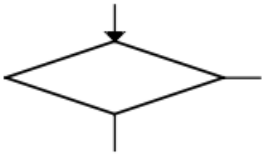
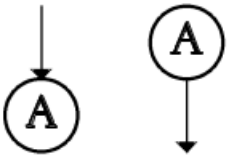
2.2.2 Les règle et les composants d'un organigramme

Il existe différentes règles pour construire un algorigramme :

— Centrer l'algorigramme au centre de la feuille.
— La lecture de l'algorigramme puisse se faire verticalement.
— Les lignes de liaisons entre les symboles ne doivent pas se couper.
— Une ligne de liaison doit toujours arriver sur le haut et au centre d'un symbole.
— Les commentaires sont à placer de préférence à droite et les renvois de branchement à gauche.
— On doit suivre les normes ISO 5807

Pour créer facilement des diagrammes, On peut utiliser des logiciels gratuits comme : **Diagram Designer** (`https://logicnet.dk/DiagramDesigner/`).

Avant de programmer, on analyse donc toutes les étapes de fonctionnement du système à partir d'un organigramme. Il doit respecter des règles d'écriture simples. Il se construit à partir des symboles normalisés et de textes. L'organigramme est une représentation graphique d'un algorithme. Il contient des formes normalisées. Chaque forme décrit une opération bien déterminée :

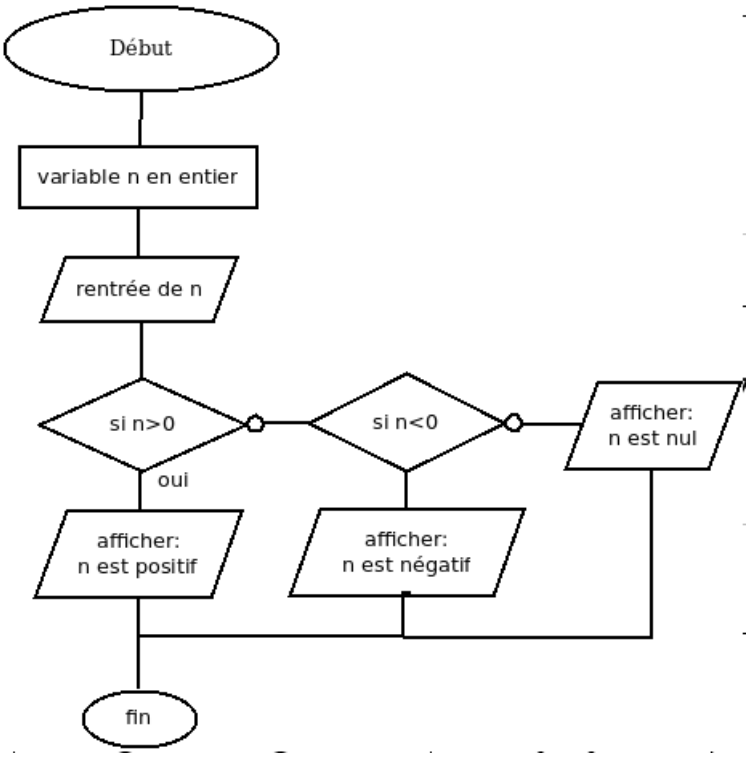
		
Début ou fin	Liaison orientée	Traitement
		
Processus prédéterminé (sous-programme)	Entrée / Sortie	Opération manuelle
		
Branchement conditionnel	Renvoi	Commentaire

2.2.3 Les avantages d'un algorigramme

Quand un programmeur écrit un algorithme compliqué il peut parfois se perdre dans ses idées. Pour éviter toutes erreurs, il fait un schéma de son algorithme qui permet de :

- Visualiser facilement les blocs du programme, les boucles, les tests et les erreurs.
- Revenir dessus et utiliser pour comprendre et changer l'algorithme.
- Éviter à chaque programmeur de marcher sur le territoire des autres (cas ou plusieurs programmes se font en équipe).
- Voir très facilement si les algorithmes vont s'annuler entre eux.
- Plus compréhensible peu importe sont langage de programmation (un algorigramme, tout le monde le comprend).

Par exemple, on désire de réaliser un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif.



2.3 Structure d'un programme

Un algorithme est l'énoncé dans un langage bien défini d'une suite d'opérations permettant de répondre au problème. Un algorithme doit donc être :

- **1. Lisible :** l'algorithme doit être compréhensible même par un non-informaticien
- **2. Haut niveau d'abstraction :** l'algorithme doit pouvoir être traduit en n'importe quel langage de programmation, il ne doit donc pas faire appel à des notions techniques relatives à un programme particulier ou bien à un système d'exploitation donné.
- **3. Précis :** chaque élément de l'algorithme ne doit pas porter à confusion, il est donc important de lever toute ambiguïté.
- **4. Concis :** il est décomposer le problème en plusieurs sous-problèmes.
- **5. Structuré :** un algorithme doit être composé de différentes parties facilement identifiables à savoir : une entête où apparaissent le nom de l'algorithme (utilisant le mot **Algorithme**) ainsi que les déclarations des objets manipulés puis le corps de l'algorithme qui commence par "**Début**" et se termine par "**Fin**. Entre ces deux mots clés se trouvent les actions ordonnées à exécuter par la machine.

```
Algorithme Nom_de_l'algorithme;          ┐
Variables                                ├ Entête
  Liste des variables;                   ┘
Début                                    ┐
  Action 1;                              │
  Action 2;                              │
  Action 3;                              ├ Corps
  ….                                     │
  Action n;                              │
Fin.                                     ┘
```

2.3.1 Caractéristiques et Conditions

1. Un algorithme = ensemble d'instructions à suivre pour aboutir à un résultat.
2. Les données dont on dispose au début, les résultats que l'on doit obtenir à la fin.
3. Chaque mot clé est souligné.
4. Une marque de terminaison (**;**) est utilisée après chaque action.
5. Un programme est suite ordonnée d'instructions et de commandes.
6. Un programme représente la traduction informatique d'un algorithme dans le langage de programmation informatique choisi.
7. Instruction = une opération élémentaire dans un programme informatique

2.3.2 Le paradigme de programmation

Le paradigme de programmation est ce choix de structuration. C'est à l'aide du paradigme que vous allez formuler la manière dont le programme s'exécutera. Le choix dépendra de ce que vous souhaitez développer, des problèmes auxquels le résultat que vous souhaiter obtenir vous confronte et de la finalité de chaque action. Parmi les types de paradigmes de programmation :

1. **Impérative :** consiste à écrire une suite d'instructions qui sert à modifier l'état d'un programme. Il existe deux sous-ensembles de ce paradigme :
   — La programmation procédurale
   — La programmation structurée
2. **Déclarative :** un type de programmation qui vise à supprimer tout effet de bord.

2.4 La démarche et analyse d'un problème

Le mot Algorithme est un dérivé du nom de l'illustre savant musulman ***Muhammad Ibn Musa Al Khawarizmi*** qui vécut au $9^{ème}$ siècle ($2^{ème}$ Hidjri), sous le règne du ***Calife abbasside Al-Mamun***.

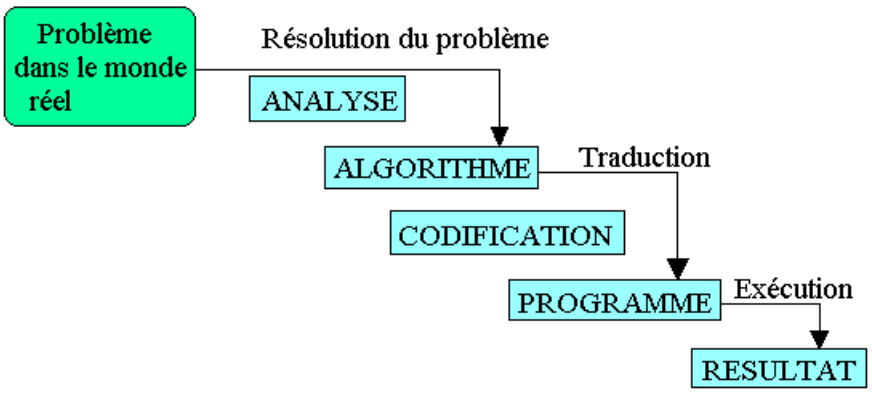
Al Khawarizmi a exposé les méthodes de base pour l'addition, la multiplication, la division, l'extraction de racines carrées ainsi que le calcul des décimales de π. Ces méthodes sont précises, sans ambiguïté, mécaniques, efficaces, et correctes. Ces méthodes sont des algorithmes.

Généralement, les étapes de résolution d'un problème quelconque sont :

- Comprendre l'énoncé du problème.
- Décomposer le problème en sous-problèmes plus simples à résoudre.
- Associer à chaque sous problème, une spécification :
  - Les données nécessaires ;
  - Les données résultantes ;
  - La démarche à suivre pour arriver au résultat ;
- Élaboration d'un plan d'actions (algorithme). Informatiser une application, facturation de la consommation d'eau par exemple, c'est faire réaliser par un ordinateur, une tâche qui était réalisée par l'homme. Pour cela, il faut abord détailler suffisamment les étapes de résolution du problème, pour qu'elles soient exécutables par l'homme.
- Transférer la résolution en une suite d'étapes simples et à exécuter.

La programmation est structurée en 05 étapes :

1. Définition et analyse du problème : Il s'agit de définir les données qu'on dispose et les objectifs qu'on souhaite atteindre, et prévoir des réponses à tous les cas envisageables.



2. Écriture de l'algorithme : C'est la phase la plus difficile et importante, elle fournit la méthode et la démarche que l'ordinateur va suivre pour résoudre le problème posé.

3. Programmation : Il s'agit d'exprimer l'algorithme dans un langage connu par l'ordinateur. Il faut donc choisir un langage de programmation et ensuite traduire l'algorithme sous forme d'un programme exprimé dans ce langage.

4. Compilation du programme : Il s'agit d'exprimer l'algorithme dans un langage connu par l'ordinateur.

5. Exécution et test du programme : Il s'agit de s'assurer que le programme donne un résultat correct dans tous les cas et pour toutes les éventualités.

Effectuer plusieurs jeux de tests correspondant aux différents cas et vérifier la validité des résultats.

2.5 Structure des données

En programmation informatique, une structure de données peut être sélectionnée ou conçue pour stocker des données de manière à pouvoir manipuler ces dernières à l'aide de plusieurs algorithmes.

2.5.1 Les variables

Une variable est un emplacement mémoire capable de contenir des valeurs de type défini au préalable. Elle peut être définie comme une boite qui admet un nom, une taille, un contenu et une adresse.

Le nom de la variable s'appelle identificateur de la variable. La taille dépend du type de la variable (exemple : 2 octets pour un entier, 1 octet pour un caractère, 4 octets pour un réel...). L'adresse désigne le numéro du premier octet occupé par cette variable en mémoire centrale.

Une variable est un objet dont le contenu peut être modifié par une action. Dans un algorithme, on peut avoir 0 à plusieurs variables.

Dans un algorithme, les variables sont déclarées comme suit :

VAR
Liste des variables suivies par des virgules : type 1 ;
Liste des variables suivies par des virgules : type 2 ;
....
...
Liste des variables suivies par des virgules : type i ;
...
...
Liste des variables suivies par des virgules : type n ;

Les types de contenu des variables les plus courants sont :

1. **Booléen :** représentant une valeur logique binaire oui ou non, ouvert ou fermé, vrai ou faux.
2. **Entier :** Un nombre entier s'écrit comme une série de chiffres, éventuellement précédée par un signe (+ ou -). Par exemple : 8, -10, 3 sont des entiers.
3. **Réel :** Permettent de représenter les nombres réels (parfois appelés flottants en informatique). Par exemple : -4.6, 3.45 E2.
4. **Caractère :** Il s'agit du domaine constitué des caractères alphabétiques et numériques . Une variable de ce type ne peut contenir qu'un seul et unique caractère. Par exemple : 'a', 'b', '5'.

   Exemple :

   VAR
   X, Y : entier ;
   A : réel ;
   text : caractere ;
   VF : Booléen ;

2.5.2 Les constantes

Une constante est un objet dont l'état reste inchangé durant toute l'exécution d'un programme. On ne peut jamais modifier sa valeur et celle-ci doit donc être précisée lors de la définition de l'objet.

Syntaxe :

CONST
Nom const = val ;

CONST
PI=3.14 ;

2.6 Les Opérateur

Un opérateur est un symbole d'opération qui permet d'agir sur des variables ou de faire des calculs.

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

Une expression est une combinaison d'opérateur(s) et d'opérande(s), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type.

2.6.1 Les opérateur d'affectation

L'opérateur d'affectation renvoie généralement une référence à l'objet afin d'être utilisé. L'affectation permet de donner une valeur à une variable. Cette action permet de ranger une nouvelle valeur dans une variable

Syntaxe

Identificateur ⟵ <expression> ; Expression peut être : une variable, une constante, une expression arithmétique ou logique.

- Une constante ne peut jamais figurer à gauche d'une affectation.
- Après une affectation, l'ancien contenu est perdu pour être substitué par le nouveau contenu.
- Une action d'affectation doit se faire entre deux types compatibles.

Par exemple :

$X \longleftarrow 81$; signifié X reçoit 81. Si X avait une valeur auparavant, cette valeur disparaît : elle est écrasé par 81.

2.6.2 Les Opérateurs Relationnels (Comparaison)

Les opérateurs relationnels désignent une action sur une relation. Ils permettent d'interroger la base de données grâce à la création de requêtes. Souvent associés aux opérateurs logiques, ils permettent de comparer des expressions pour effectuer des tests, fréquents en informatique.

Les opérateurs sont le fondement même du traitement des opérations mathématique et comparatif de n'importe quel langage, l'algorithmie ne fait pas exception. Il est a noter qu'il n'existe pas de véritable standard sur la forme d'écriture des opérateurs algorimiques, mais seulement une tendance très prononces pour les symboles mentionnés.

Opérateur	**Description**
=	Comparaison d'un égalité
<>	Comparaison d'une différence
>	Comparaison de plus grand que
>=	Comparaison de plus grand ou égal que
<	Comparaison de plus petit que
<=	Comparaison de plus petit ou égal que

Exemples :

L'expression 1+2=2+1. Le calcul de cette expression donne comme résultat **Vrai**, car effectivement 1+2 et 2+1 sont égaux.

L'expression 1+2=100 donnera, au moment du calcul, la valeur **Faux**, car 1+2 et 100 ne sont pas égaux.

L'expression x >100 donne **Vrai** si la valeur de la variable est supérieure à 100. Elle retourne Faux sinon.

2.6.3 Les opérateurs logiques

Pour exprimer les fonctions logiques, on peut utiliser des combinaisons d'opérations logiques simples (les opérations logiques unaires et binaires).

Les opérations binaires mettent en jeu deux opérandes entourant un symbole appelé l'opérateur : [**opérande**] <**opérateur**> [**opérande**].

Les opérations unaires mettent en jeu une seule opérande et un opérateur. Dans les deux cas, les opérations logiques produisent une valeur logique (Vrai/Fau) :

- **L'opération ET logique :** Quelques soient les valeurs logiques x et y = Vrai si les opérandes x ET y valent Vrai tous les deux. Par contre, x ET y = Faux dans tous les autres cas.
- **L'opération OU logique :** Quelques soient les valeurs logiques x et y, l'opération x OU y = Vrai si au moins l'une des opérandes vaut Vrai. D'ailleurs, x OU y = Faux dans tous les autres cas.
- **L'opération NON logique :** Quelque soit la valeur logique x, l'opération $\neg x$ se lit "non x". $\neg x$ = Vrai si x = Faux, $\neg x$ = Faux si x = Vrai.

Une table de vérité est une table mathématique utilisée en logique pour représenter la valeur d'une fonction relativement à ses arguments. Elle est composée d'une colonne pour chaque variable (x et y) et d'une colonne où sont inscrits les résultats de l'opération logique représentée.

x	x	x ET y	x OU y	¬ x
Vrai	Vrai	Vrai	Vrai	Faux
Faux	Vrai	Faux	Vrai	Vari
Faux	Faux	Faux	Faux	Vrai
Vari	Faux	Faux	Vrai	Faux

2.6.4 Les opérateurs arithmétiques

Les opérateurs arithmétiques utilisables sur les variables sont :

1. **Addition (+),**
2. **Soustraction (-),**
3. **Produit (*) :** Multiplication
4. **Division réelle (/) :** la division réelle donne le quotient réel de la division réelle.
5. **La division entière (DIV) :** n DIV p donne la partie entière du quotient de la division entière de n par p.
   Par exemple : 6 Div 8 = 0 alors que 6 / 8 = 0.75
6. **Le modulo (MOD) :** donne le reste de la division entière.
   n MOD p donne le reste de la division entière de n par p.
   Par exemple : 10 Mod 4 = 2.

Noter que, en algorithmique les expressions s'écrivent sous forme linéaire (sur une ligne). Par exemple : $B^2 - 4AC$ s'écrit en algorithmique comme : $(B * B) - (4 * A * C)$

2.6.5 Les priorités dans les opérations

La priorité des opérations arithmétiques par ordre décroissant est comme suite : les parenthèses sont les plus prioritaires, ensuite viennent les opérations *, /, DIV et MOD et enfin les opérations + et -. Si l'ordre entre les opérateurs dans une expression est le même, on évalue l'expression de gauche à droite. Par exemples :

17 MOD 10 DIV 3=(17 MOD 10) DIV 3=7 DIV 3=2.

2.7 Les opérations d'entrée/sortie

Elles permettent de récupérer une valeur venant de l'extérieur (lecture) ou de transmettre une valeur à l'extérieur (écriture).

1. **L'instruction de lecture : Lire**(var_1, var_2, ..., var_n) ; lit les variables : var_1, var_2, ..., var_n. La saisie se fait uniquement dans des variables. Ce sont les cases (cellules) qui pourront accueillir les données correspondantes. La donnée à introduire doit être de même type que la variable réceptrice.

   Cette instruction récupère la valeur rentrée au clavier et l'affecte à la variable. Physiquement, la valeur rentrée au clavier est mise dans l'emplacement mémoire de la variable.
2. **L'instruction d'écriture : Ecrire**(<expression>) ; Affiche l'expression sur écran. Cette action permet de communiquer un résultat ou un message sur écran ou sur imprimante pour l'utilisateur.

Par Exemples :

Ecrire(12,55) ; {Affiche sur écran la valeur 12,55}

Ecrire("Esalem alikom") ; {Affiche sur écran le message : Esalem alikom}

Si X variable entière contient 5 et Y variable entière contient 2.

Ecrire(X) ; {Affiche sur écran 5}

Ecrire(Y + 3) ; {Affiche sur écran 5}

Ecrire("La Valeur de X = ", X, " et la Valeur de Y = ", Y) ;

{Affiche sur écran : La Valeur de X = 5 et la Valeur de Y = 2}

{} est un commentaire (un texte ajouté au programme pour décrire le code source, facilitant sa compréhension par les humains.

2.8 Les structures de contrôle

Toutes les actions ne sont pas réalisables à chaque fois. En effet, s'il pleut l'action prendre le parapluie a un sens sinon elle est annulée ou remplacée par une autre comme prendre les lunettes de soleil.

En programmation, on est souvent confronté à des situations où on a besoin de choisir entre deux ou plusieurs traitements selon la réalisation ou non d'une certaine condition. C'est la notion de traitement conditionnel. On distingue deux structures de traitement conditionnel à savoir :

2.8.1 Les structures de contrôle simple

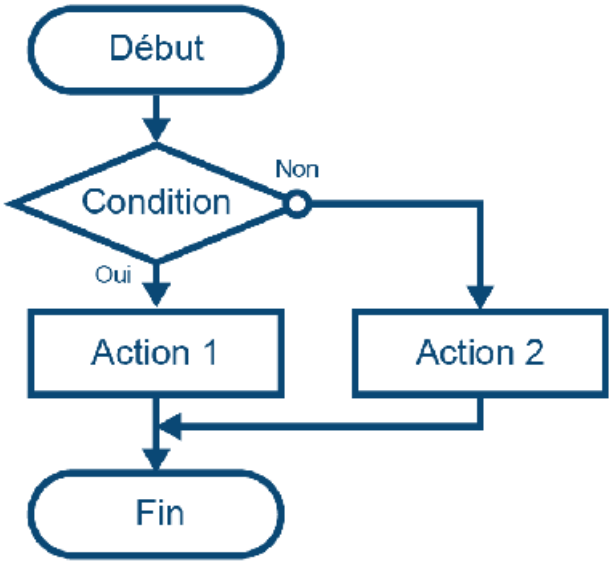
La structure conditionnelle simple qui consiste à évaluer une condition (expression logique à valeur vrai ou faux) et d'effectuer le traitement relatif à la valeur de vérité trouvée selon le syntax suivant :
SI Condition **Alors**

Instruction 1 ;

Instruction i ;

Instruction n ;
FinSi ;

Il s'agit dans ce cas de conditionner un groupe d'actions (qui peut contenir d'une à plusieurs instructions) selon un certain test. Si la condition est vérifiée, alors le groupe d'action est déclenché puis les actions suivantes sont exécutées. Dans le cas où la condition est fausse (non vérifiée) un saut est effectué vers les actions qui suivent le groupe conditionné. L'exécution de cette instruction se déroule selon l'organigramme suivant :



Par exemple, un algorithme qui permet de saisir un entier et d'afficher impossible d'être diviseur si cet entier est égal à 0.

Algorithme Diviseur ;
VER
a : Entier ;
Debut
Lir(a) ;
SI a=0 **Alors**

Ecrire ("impossible d'être diviseur") ;
FinSi ;
Fin.

L'exécution de cet algorithme pour la valeur a = 0 fait que les instructions Lire (a) et Écrire ("impossible d'être diviseur") sont les seules à être exécutées car l'évaluation de la condition a= 0 est vrai.

L'exécution de cet algorithme pour la valeur x = 7 fait que l'instructions Lire (a) seulement car la condition a = 0 est évaluée à faux.

2.8.2 Les structures de contrôle répétitives

La structure conditionnelle répétitives qui consiste à évaluer une expression qui n'est pas nécessairement à valeur booléenne (elle peut avoir plus de deux valeurs) et selon la valeur trouvée, effectue un traitement. Le syntaxe de cette instruction est :
SI Condition **Alors**

Instruction 1 ;

Instruction 2 ;

....

Instruction n ;
Sinon

Instruction 1 ;

Instruction 2 ;

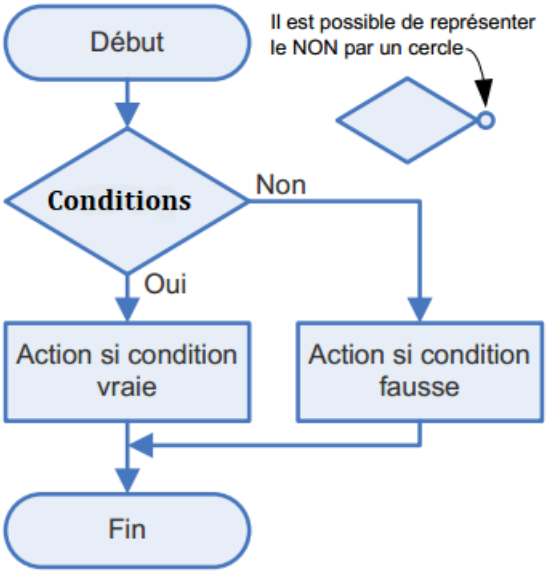
....

Instruction m ;
FinSi ;

Dans cette forme, la condition est évaluée. Si elle vaut vrai alors c'est la séquence d'**Instruction n** qui sera exécutée sinon c'est la séquence d'**Instruction m** qui sera exécutée.

Donc, Il y a obligatoirement une exécution de l'un ou l'autre des groupes d'action mais que si l'un est exécuté l'autre ne l'est pas.

L'exécution de cette instruction se déroule selon l'organigramme suivant :



Par exemple, l'algorithme suivant calcule et affiche la nature (positive ou négative) d'un nombre entier. Le zéro est considéré comme valeur positive.

Algorithme Nature ;
VAR

X : **Entier** ;
Debut
Lire (X) ;
Si X $<$ 0 **Alors**

Ecrire ('Le nombre X est négati') ;
Sinon

Ecrire ('Le nombre X est positif') ;
FinSi ;
Fin.

L'exécution de cet algorithme pour la valeur x = 69 nous conduit à suivre le chemin vers l'affichage 'Le nombre est positif'. Tandis que, l'exécution avec x = -44 conduit vers l'affichage 'Le nombre est positif'.

2.8.3 Les structures de contrôle imbriquée

Dans certains cas il existe plus d'une alternative à une condition. Il s'avère alors que le test sur une valeur peut avoir plus de deux chemins possibles et des tests sont inclus dans d'autres.

Cette forme est celle des tests imbriqués selon le syntaxe suivant :
SI condition 1 **Alors**

Action(s)1 ;
Sinon

SI condition 2 **Alors**

Action(s)2 ;

Sinon

SI condition N-1 **Alors**

Action(s)N-1 ;

Sinon

Action(s)N ;
Finsi

Si la condition 1 est vraie alors la séquence d'actions 1 sera exécutée sinon on évalue la condition 2 si elle est vraie alors la séquence d'actions 2 sera exécutée. Enfin, si aucune des N-1 conditions est vraie alors on exécute la séquence d'actions N.

Par exemple, l'algorithme suivant permet de déterminer la nature d'un nombre entier et que l'on considère que le zéro est une valeur qui n'est ni positive ni négative mais qu'il est déclaré nul :

Algorithme Nature2 ;
VAR
X : **Entier** ;
Debut
Lire (X) ;
Si X $<$ 0 **Alors**

Ecrire ('Le nombre X est négatif') ;

Sinon

Si X $>$ 0 **Alors** ;

Ecrire ('Le nombre est positif') ;

Sinon

Ecrire ('Le nombre X est une valeur nulle') ;

FinSi ;

FinSi ;
Fin.

A l'exécution de cet algorithme un et un seul des trois chemins est suivi selon que la première condition est vraie :

Si la première condition est fausse : Lire (x) ; Ecrire ('Le X nombre est négatif') ;

Si la deuxième conditions est vraie : Lire (x) ; Ecrire ('Le nombre X est positif') ;

Si les deux premières conditions sont fausses : Lire (x) ; Ecrire ('Le nombre X est une valeur nulle') ;

2.8.4 Les structures conditionnelles à choix multiples

Dans le cas où les conditions des tests imbriqués portent sur des égalités par rapport à des valeurs précises ou des intervalles, ces tests peuvent être modélisés avec le constructeur Selon qui modélise le choix multiple comme suite :

Selon <Sélecteur> **Faire**
<liste de valeurs1> : <traitement 1> ;
<liste de valeurs2> : <traitement 2> ;
...
<liste de valeursN> : <traitement i> ;
.....
<liste de valeursN> : <traitement N> ;

Fin Selon

Cette structure conditionnelle est appelée aussi à choix multiple ou sélective car elle sélectionne entre plusieurs choix à la fois, et non entre deux choix alternatifs (le cas de la structure SI).

Le sélecteur peut être une variable de type scalaire ou une expression arithmétique ou logique.

- Le sélecteur est un identificateur.
- <traitement i> est une séquence d'actions.
- <liste de valeurs i> peut être une constante ou un intervalle de constantes de même type que le sélecteur.
- La partie sinon est facultative. Elle est exécutée si aucune des valeurs n'est égale au sélecteur.

Dans le cas ou il existe un groupe d'instructions à exécuter en dernier recours, une ligne est rajoutée en fin pour le cas autre selon la syntaxe :

Selon variable **Fair**
Valeur1 : groupe instructions 1 ;
Valeur2 : groupe instructions 2 ;
...
Valeurn : groupe instructions n ;
Autre : groupe instructions n+1 ;
FinSelon ;

Par exemple l'algorithme suivant affiche l'action équivalente à la lumière active dans des feux de circulations :

Algorithme Feux ;
VAR
Feux : Chaine de caractère ;
Debut

Lire (Feux) ;

Selon Feux

'Orange' : **Ecrire** ('Attention !') ;

'Geen' : **Ecrire** ('Walk') ;

'Red' : **Ecrire** ('Stop') ;

Finselon ;
Fin.

2.8.5 Les conditions composées

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple désignée ci-dessus. Les opérateurs Logiques (et, ou, non) servent alors à composer les conditions selon la syntaxe :

Si (Cond. 1) op. logique 1 (Cond. 2) op. logique 2 ... (Cond. n) **Alors**

Il est donc possible de lier deux conditions par un mot clé appelé « opérateur logique ». Ces opérateurs se comportent alors exactement comme des portes logiques. C'est-à-dire que les entrées de l'opérateur logique sont les deux expressions booléennes, et la sortie sera la valeur testée par l'ordinateur pour décider quelle branche exécuter.

Par exemple, la condition suivante ne sera validée que si le prix est à la fois strictement plus grand que 15, et à la fois strictement plus petit que 30 :

prix $>$ 15 **ET** prix $<$ 30.

La condition suivante ne sera validée que si le prénom est Ahmed, et en même temps que l'âge est strictement inférieur à 18 ans :

Prenom = "Ahmed" **ET** age $<$ 18

NB : En algorithmique, l'écriture 15 $<$ age $<$ 30 n'est pas valable. Il faut écrire 15 $<$ age **ET** age $<$ 30.

La condition suivante est validée dès lors que l'âge est plus petit ou égal à 26 ans, ou que l'âge est plus grand ou égal à 60 ans :
age $<=$ 26 **OU** age $>=$ 60.

La condition suivante est validée dès lors que le prix est strictement plus petit que 50, ou que la variable enPromotion vaut vrai :
prix $<$ **50** OU enPromotion.

2.9 Le langage C/C++

Le langage C a été conçu en 1972 par Dennis Richie et Ken Thompson, chercheurs aux Bell Labs, afin de développer un système d'exploitation UNIX sur un DEC PDP-11. En 1978, Brian Kernighan et Dennis Richie publient la définition classique du C dans le livre The C Programming language. Le C devenant de plus en plus populaire dans les années 80.

En 1983, l'ANSI (American National Standards Institute) décida de normaliser le langage ; ce travail s'acheva en 1989 par la définition de la norme ANSI C. Celle-ci fut reprise telle quelle par l'ISO (International Standards Organization) en 1990. C'est ce standard, ANSI C, qui est décrit dans le présent document.

2.9.1 Composantes d'un programme en langage C

Un programme en langage C est constitué de composants élémentaires suivants :

- **Les identificateurs :** qui donne un nom à une entité du programme
- **Les mots-clefs :** sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs : **int, float, while, ...**
- **Les constantes :** une valeur qui apparaît littéralement dans le code source d'un programme, le type de la constante étant déterminé par la façon dont la constante est écrite. Les constantes peuvent être de 4 types : entier, flottant (nombre réel), caractère, énumération. Ces constantes vont être utilisées, par exemple, pour initialiser une variable.
- **Les opérateurs :** L'affectation, Les opérateurs arithmétiques, Les opérateurs relationnels, Les opérateurs logiques booléens.
- **Les commentaires :** sont enlevés par le préprocesseur.

Généralement, un programme en C doit contenir la fonction principale appelé **main** :

```
# include <stdio.h>
main()
{
/*déclaration des variables */
instructions ;
}
```

2.9.2 Types de base

On peut les classer en trois catégories :

1. **Le type char :** C'est le type caractère. Il représente un nombre entier codé sur un octet (huit bits).
2. **Le type int :** C'est le type entier. Il est généralement codé sur 4 octets (32 bits) et permet de représenter. La plage des valeurs acceptables pour ce type est donc (-2147483648 ; 2147483647)
3. **Les types réels (float, double et long double) :** Ces types servent à représenter les nombres réels. **float :** C'est le type de base des nombres réels. **double :** Ce type permet de représenter des valeurs ayant une partie décimale avec une plus grande précision que le type float. **long double :** Ce type est récent. Il permet de représenter des nombres avec parties décimales qui nécessitent une très grande précision.

Le tableau suivant représente les instructions de langage C et l'équivalence avec l'algorithme :

Description	Algorithmique	Langage C
En tête	Algorithme	# include<stdio.h> main()
Début de programme	Debut	{
Variable constant	const	const
Variable de type entier	Entier	int
Variable de type réa l	Réal	float
Variable de type caractère	caractère	char
Sturcture conditionnel simple	Si	if
Sturcture conditionnel répétitive	Sinon	else
Fin de programme	Fin	}
Commentaire	{*commentaire*}	/* commentaire */ ou //
Affectation	←	=
Division entière	DIV	/
Reste de la division entière	MOD	%
Et logique	ET	&&
Négation logique	¬ ou non	!
Ou logique	OU	||
Test d'égalité	=	==
Test de non-égalité	<>	!=
Addition, soustraction, multiplication	+ - *	+ - *
Acquérir des données	Lire	scanf
Communiquer un résultat	Ecrire	printf
Conditionnelles à choix multiples	Selon	switch
Acquérir un varibale entier	Lire (x)	sanf("%*d*", &x)
Acquérir un variable réal	Lire (x)	sanf("%*f*", &x)
Acquérir un variable caractère	Lire (x)	sanf("%*c*", &x)
Afficher un texte	Ecrire ('Texte')	printf("Texte")

2.9.3 Premier programme en C

Voici donc notre premier programme en C qui affiche sur l'écran la phrase : EssalemAlikom.

```
# include <stdio.h>
main()
{
printf("EssalemAlikom");
}
```

2.9.4 Les séquences d'échappement

Il existe en C plusieurs couples de symboles qui contrôlent l'affichage ou l'impression du texte.

Les séquences d'échappement sont toujours précédées par le caractère d'échappement "\" :

Symbole	**Description**
$\backslash t$	Tabulation
$\backslash n$	Nouvelle ligne
$\backslash b$	Batch (curseur arrière)
$\backslash r$	Return (retour au début de ligne, sans saut de ligne)
$\backslash a$	Attention (signal acoustique)

2.9.5 Incrémentation et Décrémentation

Les opérateurs ++ et −− sont utilisés comme suite :

$i = i + 1$ s'écrit : $i++$ ou $++i$; $i = i - 1$ s'écrit : $i--$ ou $--i$

- X = i++ ; passe d'abord la valeur de i à X, puis incrémente i (le ++ est après i, on l'incrémente après).
- X = $i--$; passe d'abord la valeur de i à X, puis décrémente i
- X = ++i ; incrémente d'abord i puis passe la valeur de i incrémentée à X (le ++ est avant i, on l'incrémente avant).
- X = $--i$; décrémente d'abord i puis passe la valeur de i décrémentée à X. Par exemple : N = 8 ; X = N++ ; Résultat : X = 8 et N = 9

2.9.6 Ecriture et Lecture d'un caractère

La fonction **putchar (c)** transfère le caractère "c" vers le fichier de sortie standard stdout (l'écran), les arguments de putchar sont de type char. Par exemples : putchar('M') ; elle affiche M. Par contre putchar(M) ; affiche 77 (77 est le code ASCII de M).

La fonction **getchar()** retourne les valeurs des caractères. Le type du résultat de getchar est int. Par exemple : C=getchar() ; elle lit les données de clavier et fournit les données seulement après confirmation par "Enter". La bibliothèque **<conio.h>** fournit la prochain caractère entré au clavier.

2.9.7 L'opérateur Conditionnel (?)

L'opérateur (?) est utilisé comme suite : result = expr1 ? expr2 : expr3 ;

Si expr1 est vraie : result=expr2, sinon : result=expr3. Par exemple :

MAX=(A>B) ?A :B ; // Si A> B Donc, MAX=A. Sinon MAX=B.

2.10 Conclusion

Dans ce chapitre, nous avons s'initier à la syntaxe de l'algorithme et du langage C qui est un langage de programmation impérative : les instructions sont exécutées pour transformer l'état actuel du programme.

Le dernier chapitre du cours est consacré pour résoudre les séries d'exercices et de travaux pratiques.

Références du chapitre

Marc Guyomard, Patrick Bosc, Laurent Miclet - Collection Algorithmes. Conception d'algorithmes (3eme édition). 2021.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2010). Algorithmique : cours avec 957 exercices et 158 problèmes. Dunod.

Bosc, P., Guyomard, M., & Miclet, L. Conception d'algorithmes Principes et 150 exercices non corrigés.

Fuchs, L., Arnould, A., Peltier, S., & Lienhardt, P. (2021). Bases en algorithmique et en programmation : Cours et 120 exercices corrigés. Bases en algorithmique et en programmation, 1-450.

Mansour, M. E. D. E. D. J. E. L. (2020). Initiation à l'Algorithmique Cours et exercices corrigés.

Chapitre 3

Exercices et Travaux Pratiques

Somewhere in that database my name sat in its own little niche, the name of a reject, undisciplined and worthless. Just the way I liked it.
Ilona Andrews.

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

- TP d'initiation et de familiarisation avec la machine informatique d'un point de vue matériel et systèmes d'exploitation (exploration des différentes fonctionnalités des OS)
- TP d'initiation à l'utilisation d'un environnement de programmation (Edition, Assemblage, Compilation, etc.)
- TP d'application des techniques de programmation vues en cours.

De plus, ce document propose un ensemble d'exercices avec leurs solutions en langage C à partir de la base algorithmique.

Travaux Pratique I

Exercice 1 : Choisir la bonne réponse :

a) Quelle est la configuration la plus performante parmi les cas suivants :

1. RAM : 256 Mo, 1GO, 8GO
2. Disque dur : 80 Go, 1600 Mo, To
3. Processeur : 1,5GHZ, 4,5 GHZ, 800 MHZ

b) Quelle est la mémoire le plus stockage parmi les choix suivants :

1. 30 Go, 10 Go, 16000 Mo
2. 128 Ko, 128 Go, 128 Mo
3. 512 Mo, 64 Mo, 1 Go

Exercice 2 : Compléter les transformation suivantes :

1. 32 Go= ..... MO= Ko = ... Bits
2. 127Mo= ..... Octets
3. 1024 MO= ..... Octets=..... Bits
4. 2048 Octets= ..... Ko= ..... Mo

Exercice 3 : La disquette 3 pouce à une taille de stockage de 1,44 Mo.

1. Quel est le nombre de caractère qui peut contenir un fichier dont la taille est égale à la taille de la disquette ?
2. Combien de disquette peut contenir un CD-ROM dont la taille est 750Mo.

Exercice 4 : Classer les composants d'un PC selon :

1. Fonctionnalité (Entrer/Sortir).
2. Positon (Interne/Externe).
3. Nature (Hardware/ Software).

Solutions de Travaux Pratique I

Exercice 1 :

1. RAM : 8 Go
2. Disque dur : 16000 Mo
3. Processeur : 4,5 GHZ

Exercice 2 :

1. 32 Go= 2^5*2^{10}=2^{15} Mo= 2^{15} * 2^{10} = 2^{25}Ko = 2^{25}*2^{13}=2^{38} Bits
2. 127 Mo= 127*2^{20} Octets
3. 1024 Mo= 2^{10} * 2^{20}=2^{30} Octets= 2^{30}*2^3=2^{33} Bits
4. 2048 Octets= 2^{11} * 2^{-10}=2 Ko= 2*2^{-10}=2^{-9} Mo

Exercice 3 :

1. La taille de fichier= taille de la disquette=1.44 Mo
   1.44 Mo =1.44*220 octet, Le caractère est codée sur 1 octet donc la disquette contient dans ce cas 1.44*220 Caractère.
2. La taille de CD-ROM=750 Mo, La taille de disquette=1.44 Mo
   Pour obtenir le nombre de disquette on divise 750MO/1.44 Mo=520 disquette.

Exercice 4 :

— Entrer : Clavier, Souris, ...
— Sortir : Écran, Imprimante, ...
— Interne : RAM, Carte mère, ...
— Externe : Écran, Clavier, Souris, ...
— Hardware : Écran, RAM, ...
— Software : Système d'exploitation, Driver, ...

Travaux Pratique II

Exercice 1

Convertir les nombres suivants en nombres des bases indiquées :

1) $(101011101)_2 = (.....)_{10}$
2) $(1011111110)_2 = (.....)_8$
3) $(10110000)_2 = (.....)_{16}$
4) $(AC9)_{16} = (.....)_{10}$
5) $(BD3)_{16} = ( )_8$
6) $(AA0)_{16} = (...)_2$
7) $(70)_8 = (...)_2$
8) $(55)_8 = (...)_{10}$
9) $(100)_8 = (....)_{16}$
10) $(1010111)_2 = (.....)_{10} = (.....)_8 = (.....)_{16}$
11) $(177)_8 = (.....)_{10} = (.....)_2 = (.....)_{16}$
12) $(B8)_{16} = (.....)_{10} = (.....)_2 = (.....)_8$
13) $(101)_{10} = (.....)_2$
14) $(777)_{10} = (.....)_8$
15) $(100)_{10} = (.....)_{16}$
16) $(99)_{10} = (.....)_2 = (.....)_8 = (.....)_{16}$

Exercice 2

On s'intéresse à des jeux de 52 cartes réparties en 4 couleurs (pique, cœur, carreau et trèfle) de 13 cartes désignées par leurs rangs (As, 2, 3, . . . , 10, Valet, Dame et Roi).

1. Proposer un schéma de codage binaire des cartes du jeu.

2. Donner, dans ce schéma, la représentation binaire du valet de trèfle.

Solutions de Travaux Pratique II

Exercice 1 :

1) $(101011101)_2 = (349)_{10}$
2) $(1011111110)_2 = (1376)_8$
3) $(10110000)_2 = (B0)_{16}$
4) $(AC9)_{16} = (2761)_{10}$
5) $(BD3)_{16} = (57230)_8$
6) $(AA0)_{16} = (101010100000)_2$
7) $(70)_8 = (111000)_2$
8) $(55)_8 = (45)_{10}$
9) $(100)_8 = (40)_{16}$
10) $(1010111)_2 = (87)_{10} = (127)_8 = (57)_{16}$
11) $(177)_8 = (127)_{10} = (1111111)_2 = (7F)_{16}$
12) $(B8)_{16} = (184)_{10} = (10111000)_2 = (270)_8$
13) $(101)_{10} = (1100101)_2$
14) $(777)_{10} = (261)_8$
15) $(100)_{10} = (64)_{16}$
16) $(99)_{10} = (1100011)_2 = (143)_8 = (63)_{16}$

Exercice 2 :

1. Pour coder les quatre couleurs on a besoin de 2 bits ($2^2 \geq 4$). On peut utiliser le codage suivant : 00 :pique, 01 :cœur, 10 :carreau et 11 :trèfle. Pour coder les 13 rangs on a besoin de 4 bits ($2^4 = 16$). On peut utiliser le codage suivant : 0000 : As, 0001 : 2, 0010 : 3,..., 1100 :Roi.Une carte peut être codée avec 6 bits $c_1; c_0; r_3; r_2; r_1; r_0$, les bits c_i indique la couleur et r_i indique le rang.

2. Avec la représentation choisie, on a $c_1c_0 = 11$ pour la couleur trèfle et $r_3r_2r_1r_0 = 1010$ pour le valet. Donc, 111010 est la représentation binaire du valet de trèfle.

Travaux Pratique III

Exercice 1

Écrire un algorithme qui calcule et affiche la surface d'un cercle.

Exercice 2

Écrire un algorithme qui calcule et affiche la valeur absolue d'un nombre réel.

Exercice 3

Écrire un algorithme qui calcule la solution de l'équation a*x+b=0 dans R.

Exercice 4

Écrire un algorithme qui lit deux réal et affiche leurs valeurs après permutation.

Exercice 5

Écrire un algorithme qui lit trois entiers et qui calcule et affiche leur somme, leur produit et leur moyenne.

Exercice 6

Écrire un algorithme qui calcul le prix à payer pour une marchandise à partir d'un prix initial et d'une remise sur ce prix.

La formule à appliquer est PaP = PI – PI*R/100.

Exercice 7

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

Exercice 8

Écrire un algorithme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant.

Solutions de Travaux Pratique III

Exercice 1 :

Algorithme surface ;
const Pi = 3,14
Var R,S : **Réel**
Début
Lire (R) ;
S ← Pi*R*R ;
Ecrire ("La surface = ",S) ;
Fin.

Exercice 2 :

Algorithme absolue ;
Var X, Abs : **Entier** ;
Début
Lire(X) ;
Si X<0 **Alors**
Abs← -X ;
Sinon
Abs ← X ;
Finsi ;
Ecrire (Abs) ;
Fin.

Exercice 3 :

Algorithme équation ;
Var a, b, x : Réel ;
Début
Lire (a,b) ; x ←-b/a ;
Ecrire (x) ;
Finsi ; **Fin**.

Exercice 4 :

Algorithme permutation 1 ;
Var a, b, c : **Réel** ;
Début
Lire (a,b) ;
c ← a ;
a ← b ;
b ← c ;
Ecrire (a,b) ;
Fin.
Algorithme permutation 2 ;
Var a, b : **Réel** ;
Début
Lire (a,b)
a ← a+b ;
b ← a-b ;
a ← a-b ;
Ecrire (a,b) ;
Fin.

Exercice 5 :

Algorithme calcul ;
Var a, b, c : **Eniter** ;
s,p,m : **Réal** ;
Début
Lire (a,b,c) ;
s ← a+b+c ;
p ← a*b*c ;
m ← (a+b+c)/3 ;
Ecrire (s,p,m) ;
Fin.

Exercice 6 :

Algorithme payer ;
var PI, R, PaP : **Réel** ;
Début
Écrire ('Donnez SVP le prix initial et la remise') ;
Lire (PI, R) ;
PaP←PI – PI* R / 100 ;
Écrire ('Le prix à payer =', PaP) ;
Fin.

Exercice 7 :

Algorithme carrer ;
Var nb, carr : **Entier** ;
Début
Ecrire ('Entrez un nombre :') ;
Lire (nb) ;
carr ⟵ nb * nb ;
Ecrire ('Son carré est : ', carr) ;
Fin.

Exercice 8 :

Algorithme parix ;
Var nb, pht, ttva, pttc : **Réel** ;
Début
Ecrire ('Entrez le prix hors taxes :') ;
Lire (pht) ;
Ecrire ('Entrez le nombre d'articles :') ;
Lire (nb) ;
Ecrire ('Entrez le taux de TVA :') ;
Lire (ttva) ;
pttc ⟵ nb * pht * (1 + ttva) ;
Ecrire ('Le prix toutes taxes est : ', pttc) ; **Fin.**

Travaux Pratique V

Exercice 1

Écrire un algorithme qui permet de saisir un entier et d'afficher pair si cet entier est pair ou impair si cet entier est impair.

Exercice 2

Écrire l'algorithme qui affiche dans l'ordre croissant trois nombres saisis dans un ordre quelconque.

Exercice 3

Écrire l'algorithme qui demande deux nombres entiers à l'utilisateur et, sans calculer le produit, l'informe ensuite si le produit est négatif, positif ou nul.

Exercice 4

Ecrire un algorithme permettant de résoudre une équation du second degré : $a * x^2 + b * x + c = 0$.

Exercice 5

Ecrire un algorithme qui saisit un jour de la semaine sous forme d'un nombre entier (0 pour dimanche, 1 pour lundi...) et affiche en clair le jour de la semaine pour un jour travaille et Week-end pour le samedi ou le dimanche. Dans tous les autres cas, il affiche Numéro de jour non valide.

Solutions de Travaux Pratique V

Exercice 1 :

Algorithme pair ;
Var x : **Entier** ;
Début
Si (x MOD 2 =0) **Alors**
Ecrire (x, 'est pair') ;
Sinon
Ecrire (x, 'est impair') ;
Finsi ;
Fin.

Exercice 2 :

Algorithme Tri ;
Var x, y, z : **Entier** ;
Début
Si (x > y **ET** y> z) **Alors**
Ecrire (x, y, z) ;
Si (x > y **ET** z> y) **Alors**
Ecrire (x, z, y) ;
Si (y > x **ET** x> z) **Alors**
Ecrire (y, x, z) ;
Si (y> x **ET** z> x) **Alors**
Ecrire (y, z, x) ;
Si (z > x **ET** x> y) **Alors**
Ecrire (z, x, y) ;
Si (z > x **ET** y> x) **Alors**
Ecrire (z, y, x) ;
Fin.

Exercice 3 :

Algorithme Composition ;
Var a, b : **Entier** ;
Debut
Lire (a, b) ;
Si ((a >= 0) et (b >= 0)) ou ((a<0) et (b<0)) **Alors**
Ecrire (‘Le produit est positif’) ;
Sinon
Ecrire (‘Le produit est négatif’) ;
FinSi ;
Fin.

Exercice 4 :

Algorithme eq2 ;
Var a, b, c, delta : **Reel** ;
Debut
Ecrire (’saisissez les valeurs de a, b et c’) ;
Lire (a, b, c) ;
Si (a=0) **Alors**
Ecrire(’C’est une équation du premier degré ’) ;
Si (b<>0) **Alors**
Ecrire (’Solution est’, -c/b) ;
Sinon
Ecrire (’Pas de solution’) ; **Sinon**
delta ← b*b-4*a*c ;
Si (delta = 0) **Alors**
Ecrire (’La solution unique est :’, -b/(2a)) ;
Si (delta > 0) **Alors**
Ecrire (’Les deux solutions sont :’, (-b-racin(delta))2*a, (-b+racin(delta))2*a) ;
Si (delta < 0) **Alors** **Ecrire** (Ppas de solution réelle’) ;
Finsi.
Fin.

Exercice 5 :

Algorithme jours ;
Var jr : **Entier** ;
Debut
Ecrire ("Numero du jour ? ") ;
Lire (jr) ;
Selon (jr) **Fair**

1 : **Ecrire** ("Lundi") ;
2 : **Ecrire** ("Mardi") ;
3 : **Ecrire** ("Mercredi") ;
4 : **Ecrire** ("Jeudi") ;
5 : **Ecrire** ("Vendredi") ;
6 : **Ecrire** ("Samdi") ;
7 : **Ecrire** ("Week-end") ;
Autre : **Ecrire** ("Numero de jour non valide") ;
FinSelon ;

Fin.

Travaux Pratique IV

Exercice 1

Écrivez un programme C qui demander à l'utilisateur deux nombres, les mémoriser dans deux variables, multiplier leurs valeurs en affectant le résultat à une troisième variable, puis l'afficher.

Exercice 2

Écrivez un programme C qui demande le prix unitaire HT et le nombre d'exemplaires. Puis calculer et afficher le prix total HT, la TVA et le prix total (TTC) à payer.

Exercice 3

Écrivez un programme C qui demande le rayon d'un cercle à l'utilisateur et trouve l'aire et le périmètre du cercle.

Exercice 4

Écrire un programme en C permettant de résoudre une équation du second degré :

$$a * x^2 + b * x + c = 0$$

Exercice 5

Écrivez un programme en C qui calcule la distance entre deux points $X1(x1, y1)$ et $X2(x2, y2)$ d'un plan avec la formule :

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

Solutions de Travaux Pratique IV

Exercice 1 :

```
# include <stdio.h>
main() {
int nb1, nb2, resultat ; // Entiers : nb1, nb2, resultat
printf("Produit de deux entiers") ;
printf("Entrez un nombre entier : ") ;
scanf("%d", &nb1) ; // Entrer : nb1
printf("Entrez un deuxieme nombre entier : ") ;
scanf("%d", &nb2) ; /*Entrer : nb2*/
resultat = nb1 * nb2 ; // resultat <- nb1 * nb2 ;
printf("%d × %d = %d", nb1, nb2, resultat) ;
}
```

Exercice 2 :

```
# include <stdio.h>
main()
{
float TAUXTVA ;
float prixUnitaireHT ;
int nb ;
printf("Prix TTC\n") ;
printf("Saisir le prix HT : ") ;
scanf("%f", & prixUnitaireHT) ;
printf("Saisir le nombre d'exemplaires : ") ;
scanf("%d", & nb) ;
printf("Prix Total HT : % f", prixUnitaireHT * nb) ;
printf("Prix TVA : %f", prixUnitaireHT * nb * TAUXTVA) ;
printf("Prix Total TTC : %f", prixUnitaireHT * nb * (1 + TAUXTVA)) ;
}
```

Exercice 3 :

```
# include <stdio.h>
main() {
float r, area, perimeter ;
printf("Entrez le rayon du cercle : ") ; // Demander le rayon du cercle
scanf("% f", &r) ;
perimeter = 2 * 3.14 * r ; //Trouver l'aire et le périmètre du cercle
area = 3.14 * (r * r) ;
printf("Le périmètre du cercle = % f unités", perimeter) ;
printf("L'aire du cercle = % f unités", area) ;
}
```

Exercice 4 :

```
# include <stdio.h>
main() {
float A, B, C, D ;
printf("Introduisez les valeurs pour a, b, et c : ") ;
scanf("%f %f %f", &A, &B, &C) ;
D = pow(B,2) - 4.0*A*C ;
if (A==0 && B==0 && C==0)
printf("Tout réel est une solution de cette équation.") ;
else if (A==0 && B==0)
printf("Cette équation ne possède pas de solutions.") ;
else if (A==0) {
printf("La solution de cette équation du premier degré est :") ;
printf(" x = %f", -C/B) ; }
else if (D<0)
printf("Pas de solutions dans R") ;
else if (D==0)
printf(" x = %f", -B/(2*A)) ;
else { printf(" x1 = %f", (-B+sqrt(D))/(2*A)) ;
printf(" x2 = %f", (-B-sqrt(D))/(2*A)) ; } }
```

Exercice 3 : # include <stdio.h>

 # include <math.h>

main()

{

float x1 = 0.0 ;

float y1 = 0.0 ;

float x2 = 0.0 ;

float y2 = 0.0 ;

float D = 0.0 ;

float Dx, Dy ;

printf("Entrez les coordonnees du point X1 :") ;

scanf("% f % f", & x_1, & y_1) ;

printf("Entrez les coordonnees du point X2") ;

scanf("% f% f", & x_2, & y_2) ;

Dx = x1 - x2 ;

Dy = y1 - y2 ;

D = **sqrt**(Dx*Dx + Dy*Dy) ;

printf("D(X1, X2) = ", D) ;

}