République **A**lgérienne **D**émocratique et **P**opulaire

Ministère de l'**E**nseignement **S**upérieur et de la **R**echerche **S**cientifique

Université des **S**ciences et de la **T**echnologie d'**O**ran **M**ohamed **B**oudiaf

Faculté de Génie Mécanique

Département de Génie Maritime

Cours

Algorithme et Programmation

Licence première année

Cours et exercices

Présentée par:

Dr. MADANI FOUATIH Omar

Année Universitaire 2018 / 2019

AVANT-PROPOS

Ce cours d'informatique, destiné aux étudiants de première année de l'enseignement supérieur LMD sciences technologiques, constitue un traité général des principes du langage de programmation ***Fortran****, qui permettra à l'étudiant, de comprendre et d'assimiler toutes les notions de base de ce langage. De plus, l'étudiant disposera aussi d'une grande variété d'exercices qui auront été traité dans les séances des travaux dirigés.*

L'Algorithmique et Programmation *est un chapitre important puisque pour résoudre un problème, il est indispensable d'établir d'abord un algorithme, puis de passer à la programmation en Fortran. La polycopie est destinée à donner les bases informatiques soit du point de vue algorithmique que de celui des constituants matériels d'un ordinateur.*

Ce document est divisé en trois parties. La première partie présente l'ordinateur et les différentes tâches qu'il peut traiter. Une présentation très détaillée est donnée pour expliquer la fonction des différents composants constituant l'ordinateur.

La deuxième partie explique les principes de base du ***système de numérotation*** *et le codage tel que la base binaire et les différentes opérations arithmétiques et logiques. Plusieurs exercices sont donnés à titre d'exemples, pour montrer comment les entiers et les réels sont représentés en mémoire en forme d'octet.*

L'algorithmique et la programmation *sont traités dans la troisième partie. Apprendre l'algorithmique, c'est enseigner, à utiliser la structure logique d'un programme informatique. Les instructions de lecture et d'écriture et les formats de conversion des données sont décrits. Les structures de contrôle conditionnelles et itératives sont arrangées, de manière à fournir à l'étudiant, les bases de la programmation en Fortran, afin que ce dernier puisse maitriser les applications au calcul scientifique. Cette partie est clôturée par les notions sur les tableaux, avec l'introduction des notations matricielles et des fonctions essentielles manipulant les tableaux multidimensionnels.*

Sommaire

Avant- propos

I. Introduction à l'informatique

1. Définition de l'ordinateur et informatique: 1

2. Système informatique :..... 1

3. Types d'ordinateurs :..... 1

4. Critères de performance d'un ordinateur : 1

5. Principaux éléments d'un ordinateur : 2

5.1. Les périphériques d'entrée : 2

5.2. Les périphériques de sortie :..... 2

5.3. La carte mère :..... 2

5.4. Le microprocesseur ou l'unité centrale :..... 3

5.5. La mémoire centrale :..... 3

5.6. Les mémoires auxiliaires (ou de masse) : 3

5.7. Les interface : 4

5.8. Les unités périphériques internes (carte sons, carte vidéo, carte réseau...) 4

6. L'Unité Centrale (l'unité de traitement) : 4

6.1. Unité de Mémoires Centrales (dite parfois principales) : 4

6.2. L'unité centrale de traitement : 5

II. Codage 11

1. Introduction 7

2. Principes de base du système de numérotation 7

2.1 Le système binaire naturel :..... 8

3. Les unités de mesures :..... 8

4. Les bases de numérotations utilisées en informatique 9

5. Transcodage (ou conversion de base) 10
5.1 Conversion entre les systèmes octal et binaire: 10
5.2. Conversion entre systèmes hexadécimaux et binaires: 11
6. Opérations usuelles en binaire 11
7. Opérations logiques 12
8. Codage des entiers relatifs 13
Le binaire signé 13
Code complément à 1 14
Complément à deux 15
9. Codage des nombres réels 17
Travaux Dirigés 1- codage 19
III. Généralités sur l'algorithmique et la programmation
1. Introduction : 24
2. Cycle de développement d'un programme: 24
3. Historique : 25
4. Le langage informatique : 25
5. Programmation : 26
5.1. Généralités sur la programmation en Fortran : 26
5.2. Historique : 26
6. Notion d'algorithme : 26
7. Caractéristiques d'un algorithme : 27
8. Analyse d'un problème: 27
9. La démarche et analyse d'un problème : 27
10. La représentation d'un algorithme : 27
11. Organigramme : 28
12. Structure d'un algorithme : 28
12.1. L'en-tête d'un algorithme : 29
12.2. La partie déclarative : 29

12.2.1 Les données d'un algorithme : 29
12.2.2. Les types de données : 30
12.3. Le corps d'un algorithme: 31
12.3.1 Les instructions lectures/écritures 31
Définition du format 33
12. 3.2. Opérateurs élémentaires et Expressions : 33
13. Structures de contrôle d'un algorithme : 35
13.1. Structure séquentielle : 35
Travaux Dirigés 2- Introduction à l'Algorithmique 37
13.2 Les Structures conditionnelles : 38
13.2.1 Structures conditionnelles simples : 39
13.2.2 Les structures alternatives : 39
13.2.3. Structures conditionnelles imbriqués : 40
13.2.3 La structure de choix (choix multiples) 41
Travaux Dirigés 3- Les Structures conditionnelles 43
13.3. Structures itératives ou répétitives: 45
13.3.1. Les boucles Tant que : 45
13.3.2. Les boucles Pour : 46
Travaux Dirigés 4- Les Structures itératives 47
14. Les tableaux 49
14.1. Définition 49
14.2. Représentation de tableau 49
14.3. Terminologie des tableaux 50
14.4. Manipulation d'un tableau 51
14.5. Tri d'un tableau 52
Travaux Dirigés 5 - Les Tableaux 56
Références bibliographiques 59

1. Définition de l'ordinateur et informatique:

Le terme anglais computer signifiait au départ : calculateur numérique électronique. En effet, les premières machines étaient surtout utilisées pour effectuer des suites d'opérations arithmétiques. Le terme français ordinateur est mieux adapté à la réalité d'aujourd'hui car il s'éloigne de la connotation numérique. L'ordinateur se définit maintenant comme une machine de traitement de l'information.

2. Système informatique :

Un système informatique est l'ensemble des moyens logiciels (software) et matériel (hardware) nécessaire pour satisfaire les besoins informatiques des utilisateurs.

L'information (ou *données)* est introduite par un organe d'entrée (clavier, disques, souris, commandes vocales, manettes etc.) pour être traitée dans l'unité centrale par un *programme* qui fournit des *résultats* à un organe de sortie (écran, imprimante, disques, synthétiseur de voix, commandes d'un robot etc.).

3. Types d'ordinateurs :

Le développement des ordinateurs se poursuit de nos jours dans deux directions principales :

- Accroissement des performances
- Miniaturisation des composants

- Super ordinateurs & Ordinateurs (PTT en Algérie, Columbia de la NASA,..)
- Mini Ordinateur-station de travail (HP wilaya d'Oran, Unix et Vax dépt informatique USTO).
- Micro-ordinateurs (IBM, Compaq, Apple, Macintosh,…)
- Mini micro-ordinateurs.

4. Critères de performance d'un ordinateur :

L'ordinateur est une machine performante qui est caractérisé par :

- La rapidité d'exécution des instructions (quelques milliards d'opérations élémentaires par seconde),
- La fidélité, l'ordinateur ne se trompe pas,
- La précision de calcul, l'ordinateur peut manipuler des chiffres très petits ou très grands,
- Possède une grande capacité mémoire de stockage de l'information dans d'excellentes conditions de sécurité.

5. Principaux éléments d'un ordinateur :

La configuration d'un ordinateur correspond à l'organisation adoptée pour mettre ensemble et faire fonctionner les divers éléments matériels (processeur, mémoire, terminaux, imprimante, unités de disque, etc.)

Les composants (principaux) d'un ordinateur

sont :

Les périphériques d'entrées standards

Les périphériques de sorties standards.

La carte mère

L'unité de traitement

5.1. Les périphériques d'entrée :

Ces unités, permettent à l'utilisateur, d'entrer des données, des commandes et des programmes qui seront gérés par l'unité de traitement. Ces données, quelle que soit leur forme, sont traduites en configurations identifiables par l'ordinateur.

- **Le clavier :** L'organe d'entrée standard est le ***clavier***.
- **La souris :** C'est le périphérique qui a révolutionné l'utilisation du micro-ordinateur. Il a permis à ce que le clavier ne soit plus seul maître de l'entrée des informations.
- **Les périphériques d'entrée *auxiliaires* :** D'autres organes tels que le ***scanner***, la ***carte magnétique***, le ***crayon optique***, l'***écran tactile*** et les ***mémoires auxiliaires,*** permettent également d'entrer l'information

5.2. Les périphériques de sortie :

Ces unités permettent à l'utilisateur de recevoir, des résultats, des calculs ou des manipulations de données, effectués par l'ordinateur. L'organe de sortie standard est l'écran. L'imprimante est un autre périphérique de sortie. D'autres périphériques de sortie peuvent être connectés à l'ordinateur, comme les haut-parleurs, la table traçante et les mémoires auxiliaires qui permettent également de transmettre de l'information.

5.3. La carte mère :

La carte-mère représente l'élément caractéristique principal de l'ordinateur, c'est sur cette carte que sont branchés ou soudés l'ensemble des éléments nécessaires de l'ordinateur. Tous les échanges de données partent et terminent à la carte-mère. Les différents types de composants sont :

- les puces électroniques ou chips : puces ROM, puces CMOS, chipsets ou contrôleurs : carte son, carte pour mémoires auxiliaires (disque dur et disquettes), carte pour lecteur de CD-ROM, carte vidéo
- les prises et ports d'entrée et de sortie ou d'E/S

5.4. Le microprocesseur ou l'unité centrale :

C'est le cerveau de l'ordinateur, car il exécute les instructions des programmes grâce à un jeu d'instructions, appelé aussi l'unité centrale (CPU Central Processing Unit). Il a pour tâche principale :

- d'effectuer des calculs arithmétiques et logiques
- de transmettre des données.

Le volume d'informations qu'un processeur peut recevoir ou émettre (traitement externe) et celui qu'il peut traiter en un cycle de traitement (traitement interne) s'exprime en multiples de 8 bits. L'unité de mesure est le MHz (Mégahertz). Un hertz équivaut à un battement par seconde, 1GHz équivaut à un milliard de battements par seconde.

5.5. La mémoire centrale :

La mémoire centrale qui est l'endroit où toutes les informations et les programmes résident pour être utilisés par la machine. Elle se compose de deux parties :

- la ***ROM*** (Read Only Memory) ou ***MEM*** (Mémoire Morte) qui est une mémoire permanente contenant les premières instructions nécessaires à la machine pour démarrer. Elle ne se vide pas lorsque le courant est coupé.
- la ***RAM*** (Random Access Memory) ou ***MEV*** (MémoirE Vive) qui est une mémoire effaçable mise à notre disposition et totalement vide lorsque le courant n'est pas mis. Il faut donc la remplir pour pouvoir travailler et il faut sauver son contenu lors d'une fin de session de travail.

5.6. Les mémoires auxiliaires (ou de masse) :

Les mémoires secondaires, sont des mémoires permettant le stockage permanent d'un très grand nombre d'informations. Ils permettent de conserver des données, des résultats intermédiaires ou définitifs et des programmes sous une forme directement accessible à la machine. Un ordinateur possède également d'autres types de stockage qui conservent

l'information de manière quasi permanente comme les flashs disque (USB), les disques durs, ou les disques optiques (CD, DVD).

Remarque 3: Les organes d'exploitation (lecteurs de disques, lecteur de bande,…) de ce type de mémoires exécutent des ordres de **LECTURE** et /ou d'**ECRITURE**.

Remarque 4 : les organes d'entré + les organes de sortie + les mémoires de masse constituent ce qu'on appelle les ***périphériques externes*** de l'ordinateur.

5.7. Les interface :

Sont des circuits spécialisés destinés à assurer les échanges d'informations entre l'unité centrale et les organes périphériques d'E/S.

5.8. Les unités périphériques internes (carte sons, carte vidéo, carte réseau...)

6. L'Unité Centrale (l'unité de traitement) :

Il s'agit ici du cœur de la machine ; ce sont les organes capables de transformer l'information et de commander le fonctionnement de l'ensemble des autres organes.

L'unité centrale de l'ordinateur. L'ensemble, CPU+ Mémoire Centrale, est souvent appelé unité centrale. Généralement, on peut distinguer à l'intérieur de l'unité centrale deux blocs distincts :

- unité de mémoire centrale,
- unité centrale de traitement.

6.1. Unité de Mémoires Centrales (dite parfois principales) :

La mémoire centrale ou principale (*Main Memory*) contient les instructions et les données des programmes que l'on désire exécuter, ainsi qu'une partie du système d'exploitation nécessaire au bon fonctionnement de l'ordinateur. Elle se compose de:

a- **Mémoire vive ou RAM (R**andom **A**ccess **M**emory)** est une mémoire à accès aléatoire, le temps d'accès est indépendant de sa place. C'est une mémoire où on peut enregistrer et effacer des informations à volonté. Ce type de mémoire, généralement fabriquée à l'aide de circuits intégrés, est utilisé pour stocker, les données en attente de traitement, les résultats intermédiaires ou définitifs et les programmes d'applications.

Remarque : Toute machine de traitement de l'information comprend ce genre de mémoire !

b- <u>Mémoire morte ou ROM (R</u>ead <u>O</u>nly <u>M</u>emory) :

Ce type de mémoire permet notamment de conserver les données nécessaires au démarrage de l'ordinateur. Il s'agit de la ***ROM*** (*Read Only Memory*, dont la traduction est *mémoire en lecture seule*) appelée parfois *mémoire non volatile*, car elle ne s'efface pas, lors de la mise hors tension du système. En effet, ces informations ne peuvent être stockées sur le disque dur, étant donné que les paramètres du disque (essentiels à son initialisation) font partie de ces données vitales à l'amorçage. Différentes mémoires de type ROM contiennent des données indispensables au démarrage, c'est-à-dire :

- Le BIOS est un programme permettant de piloter les interfaces d'entrée-sortie principales du système.
- Le chargeur d'amorce: un programme permettant de charger le système d'exploitation en mémoire (vive) et de le lancer.
- Le Setup CMOS, c'est l'écran disponible à l'allumage de l'ordinateur permettant de modifier les paramètres du système (souvent appelé BIOS à tort...).
- Le Power-On Self Test (POST), programme exécuté automatiquement à l'amorçage du système permettant de faire un test du système.

c- La mémoire cache :

Le principe de la mémoire cache ou antémémoire apporte une solution au problème de la trop grande différence de vitesse entre le CPU et la mémoire centrale. La solution consiste à insérer entre les deux, une mémoire très rapide, mais pas trop grande, qui va contenir les informations, dont a besoin le CPU. Cette mémoire ne fait pas partie de la mémoire centrale. La mémoire cache contient les instructions et les données les plus fréquemment utilisées par le processeur lors de l'exécution d'un programme.

6.2. L'unité centrale de traitement :

L'unité centrale de traitement (**UCT**) ou processeur central (Central Processing Unit=**CPU**) est l'élément moteur de l'ordinateur qui interprète et exécute les instructions du programme. Véritable cerveau de l'ordinateur, le CPU intimement associé à la mémoire où sont stockées les instructions et les données à traiter. Il se présente sou forme d'un petit boîtier (quelques cm de long) muni de « pattes » qui sont des broches de connections pour les liaisons avec les autres organes.

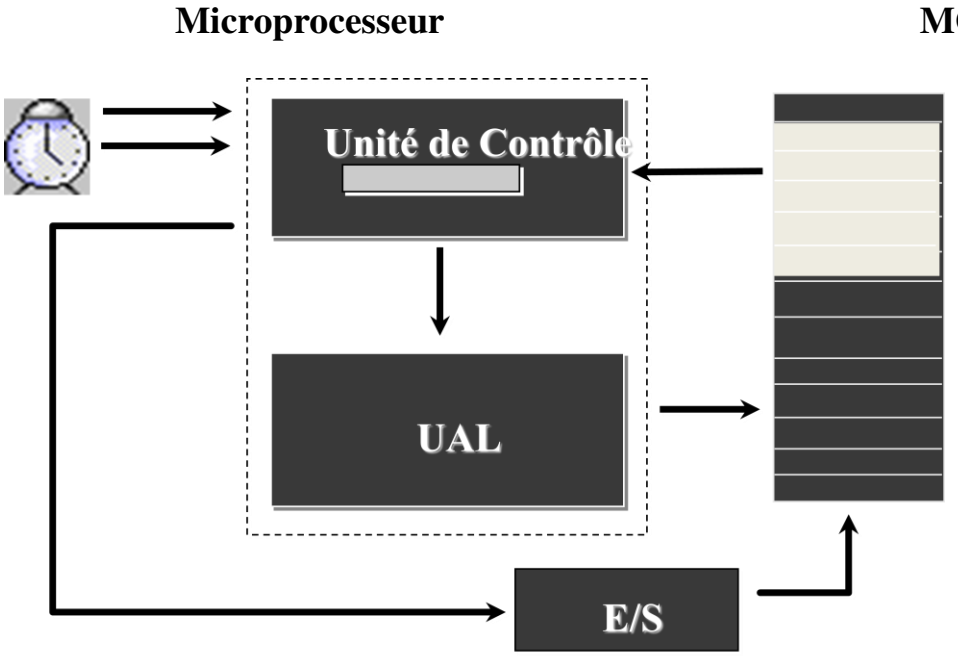
On définit souvent les micro-processeurs par la longueur du mot élémentaire qu'il peut traiter en un seul cycle (de 16, 32 64) actuellement les micro-processeurs sont à 32 bits ou encore à 64 bits.

L'unité centrale de traitement se compose de deux unités fonctionnellement séparées :

- L'unité arithmétique et logique (**UAL**)
- L'unité de contrôle ou de contrôle (**UCC**)

a- Unité arithmétique et logique (UAL) : est un organe capable de transformer les données en résultats en effectuant des calculs arithmétiques et logiques, grâce à un opérateur (circuit) implémentant le jeu d'instructions du micro processeur. La plus part des ordinateurs modernes ont des UAL capables de réaliser une grande variété d'opérations. Elle réalise les opérations arithmétiques (+, -, *, /) et logiques (ET, OU, etc.).

b- Unité de contrôle ou de commande (UCC) : Son rôle est d'extraire une instruction du programme en MC, de la faire exécuter par l'UAL ou un périphérique et de chercher l'instruction suivante. Elle décode les instructions et trouve les données pour l'UAL, et reliée à l'horloge système. Sa cadence de fonctionnement dépend de la fréquence



L'Horloge : Elle contrôle et synchronise le microprocesseur et les composants associés.

II. Codage

1. Introduction

Le système de numération utilisé généralement est le système décimal. Toute information qui transite dans un ordinateur est codée avec des bits (Binary digIT) ne prenant que les valeurs 0 et 1. L'information est donc toujours discrète Par exemple pour coder le nombre 13 en binaire, il faut les quatre chiffres binaires 1101. En effet 13 peut être décomposé comme une somme de puissances de 2.

2. Principes de base du système de numérotation

Un système de numérotation positionnel pondéré à base b défini sur un alphabet de b chiffres : $A= \{c_0, c_1, c_2, \ldots, cb_{-1}\}$

Exemple : dans le système décimal :

$$Z = \{0, 1, 2,\ldots, 9\}$$

- Une base (combien de chiffres); Exemple: $B= \left|Z\right| = 10$

• Un nombre est une séquence linéaire de chiffres.

• La valeur d'un chiffre à une position spécifique dépend de sa valeur signification "et sur sa position.

• La valeur d'un nombre est la somme de ces valeurs.

On peut écrire les nombres réels sous diverses bases. La base décimale est la plus usagée. La forme générale s'écrit : $N= A_{n-1}\ A_{n-2}\ \ldots\ A_1\ A_{0\,(b)}$

- A_i : est un chiffre de l'aphabet de poids i (position i) ;
- A_0 : chiffre de poids 0 appelé le chiffre de poids faible ;
- A_{n-1} : chiffre de poids n-1 appelé le chiffre de poids fort.

La valeur de **N** en base ***b*** est donnée par le formule suivante :

$$N_b=\sum_{i=0}^{n-1} A_i\ b^i = A_{n-1}\ b^{n-1} + A_{n-2}\ b^{n-2} + \ldots + A_0\ b^0$$

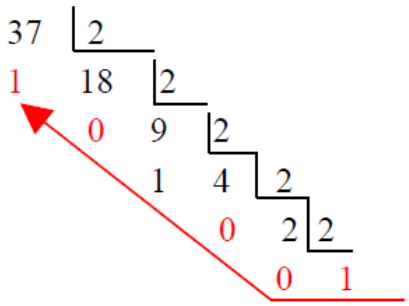
2.1 Le système binaire naturel :

Maintenant que nous avons vu comment est possible de compter dans des systèmes de numérotation autres que le système décimal. Toute information, que ce soient des nombres, du texte, des images, des sons, des relations, les circuits électroniques d'un ordinateur ne peuvent utiliser que des informations en binaire (Binary digIT). C'est le système binaire, qui utilise seulement les deux chiffres 0 et 1.

Nous pouvons compter dans le système binaire en utilisant le plan expliqué dans le point précédent pour compter dans d'autres systèmes.
Par exemple pour coder le nombre 14 en binaire, il faut les quatre chiffres binaires 1110. En effet 14 peut être décomposé comme une somme de puissances de 2

$14 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$ on ne conserve que les coefficients
$= 1\ 1\ 0\ 1$ en binaire

Exemple : pour convertir le nombre 37 (écrit en décimale) en binaire, nous pouvons utilisé la division Euclidienne comme suit :



$37 = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
Le nombre décimal 37 est représenté base 2 : 1 0 0 1 0 1

3. Les unités de mesures :

L'information traitée par l'ordinateur est représentée par des nombres appelés Bit. En système de numération binaire, le bit désigne en anglais (Binary Digits) est la plus petite unité qui prend deux valeurs soit 0 ou 1. Dans le coté électrique, l'information binaire est représentée par un signal électrique qui possède un certain seuil correspond à 1.
Donc : Avec un seul bit (O ou 1) on peut coder 2^1 Bits (2 bits).

Avec un deux bits on peut coder 2^2 Bits (4 bits).
Avec un trois bits on peut coder 2^3 Bits (8 bits).

L'octet : est une unité de mesure composée de 8 bits 1 octet =8 bits.
1ko (kilo octet) =1k byte=1024 octet=210 octet
1Mo (Mega octet) =1M byte=1024 Ko =220 octet
1GO (Gega octet) =1G byte=1024 Mo=230 octet
1To (Tera octet) =1T byte=1024 Go=240 octet

Parmi les unités de mesure on peut citer aussi : Bps (bit/seconde) : unité de mesure de modem (la vitesse des communications) Hertz : unité de mesure de fréquence (nombre d'événements par seconde), fréquence de bus, fréquence de processeur

4. Les bases de numérotations utilisées en informatique

Les bases de numérotations utilisées en informatique sont Binaire, Octale et Hexadécimale.

- **Système binaire (b=2)** : représente le système de fonctionnement des ordinateurs, utilise deux chiffres $\{0,1\}$
- **Système Octale (b=8)** : utilisé il y a un certain temps en informatique, il permet de coder *3 bits* par un seul symbole. Le décompte octal pourrait avoir été utilisé dans le passé à la place du décompte décimal. Le système octal utilise 8 chiffres $\{0, 1, 2, \dots, 7\}$
- **Système hexadécimal (b=16)** : est un système qui utilise 16 symboles, en général les dix premiers chiffres en chiffres arabes et les lettres *A* jusqu'à *F* pour les six suivants. Il utilise 16 chiffres : $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Le système hexadécimal est très utilisé notamment en électronique numérique et en informatique car il est particulièrement commode et permet un compromis entre le code binaire des machines et une base de numération pratique à utiliser pour les ingénieurs. Il permet de coder 4bits en un seul symbole
Nous verrons dans cette opération algorithmique une méthode permettant d'exprimer un nombre entier dans une base b.
Les exemples suivants enrichissent le concept :

- $57_{(10)} =? _{(2)}$

57 ÷ 2 = 28 reste **1**

28 ÷ 2 = 14 reste **0**

14 ÷ 2 = 7 reste **0**

7 ÷ 2 = 3 reste **1**

3 ÷ 2 = 1 reste **1**

1 ÷ 2 = 0 reste **1**

d'où $57_{(10)}$ = **111001**$_{(2)}$

$637_{(10)} = ?_{(16)}$

637 ÷ 16 = 39 reste **13** (représenté par **D** en hexadécimal)

39 ÷ 16 = 2 reste **7**

2 ÷ 16 = 0 reste **2**

d'où 637(10) = **27D**(16)

5. Transcodage (ou conversion de base)

Le transcodage (ou changement de base) 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.

En transcodage nous pouvons discerner les conversions suivants ;

- Binaire vers Décimale, Octale et Hexadécimale
- Décimale vers Binaire, Octale et Hexadécimale

5.1 Conversion entre le système octal et binaire:

Comme nous avons observé à cet instant, écrire et lire les nombres sous forme binaire naturelle est assez gênant à cause du grand nombre, de chiffres impliqués. Puisqu'il est facile de convertir des nombres binaires naturels en nombres octaux, il est pratique d'écrire les nombres binaires naturels sous forme octale pour faciliter la manipulation. Quelques exemples vous dévoileront comment les conversions sont effectuées.

Exemple **:** Convertissez le nombre binaire 1011010 en équivalent octal.

Solution **:** La première étape consiste à réécrire le numéro avec les chiffres groupés par trois:

001 011 010

1. Notez que deux zéros ont été placés devant le premier chiffre 1 afin de créer chaque groupe complet.
2. Ensuite, écrivez l'équivalent décimal sur chaque groupe de trois:

1 3 2

001 011 010

3. L'équivalent octal du nombre binaire 1011010 est 132.

5.2. Conversion entre systèmes hexadécimaux et binaires:

En tant qu'avoir sans doute observé à cette époque, écrire et lire les nombres en binaire naturel, la forme est assez gênante à cause du grand nombre de chiffres impliqués. Comme c'est facile de convertir des nombres binaires naturels en nombres hexadécimaux, il est pratique d'écrire ou d'utiliser une machine à imprimer des nombres binaires naturels sous forme de nombres hexadécimaux pour faciliter la manipulation.

Quelques exemples vous montreront comment les conversions sont effectuées.

Exemple : Convertissez le nombre binaire 1011010 en équivalent hexadécimal.

Solution : La première étape consiste à réécrire le numéro avec les chiffres regroupés en à quatre bits:

0101 1010

• Notez que le zéro a été placé devant le premier chiffre 1 afin de rendre les deux groupes complets.

• Ensuite, écrivez l'équivalent décimal sur chaque groupe:

5 A

0101 1010

• L'équivalent hexadécimal du fichier binaire 1011010 est 5A.

6. Opérations usuelles en binaire

Les deux opérations binaires de base sont

– l'addition

– la multiplication

Le concept est similaire à l'addition décimale

• Exemple: Ajoutez les nombres binaires 1010 et 11

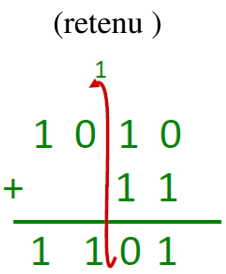


Table d'addition binaire	*Table de multiplication binaire*
0 + 0 = 0	0 x 0 = 0
0 + 1 = 1	0 x 1 = 0
1 + 0 = 1	1 x 0 = 0
1 + 1 = 0 avec une *retenue* de 1	1 x 1 = 1

Exercice : On utilisant la table d'adition binaire, calculer les aditions suivantes :

1101 + 10 =

111 + 111 =

101101+111000 =

7. Opérations logiques

En logique binaire une variable logique (booléenne) est VRAI (TRUE = 1) ou FAUSSE (FALSE= 0). Une expression logique est constituée de plusieurs variables logiques combinées par des connecteurs (opérateurs) logiques :

Les opérateurs logiques élémentaires sont :

- NON [*NOT*]
- ET [*AND*]
- OU [*OR*]

Table de vérité du OU	*Table de vérité du ET*	*Table de vérité du NON*
0 OU 0 = 0	0 ET 0 = 0	NON 0 = 1
0 OU 1 = 1	0 ET 1 = 0	NON 1 = 0
1 OU 0 = 1	1 ET 0 = 0	
1 OU 1 = 1	1 ET 1 = 1	

8. Codage des entiers relatifs

Il existe au moins trois méthodes pour coder :

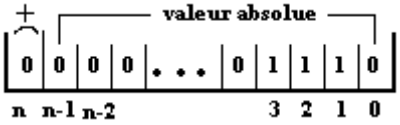
- Code binaire signé (par signe et valeur absolue)
- Code complément à 1
- Code complément à 2 (Utilise sur ordinateur)

Le binaire signé

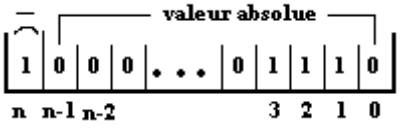
Ce codage permet la représentation des nombres entiers relatifs. Dans la représentation en binaire signé, le bit le plus significatif est utilisé pour représenter le signe du nombre :

- si le bit le plus fort = 1 alors nombre négatif
- si le bit le plus fort = 0 alors nombre positif

Les **n** autres bits représentent la valeur absolue du nombre en binaire pur.



+14 est représenté par **0**000...01110



-14 est représenté par **1**000...01110

Exemple : Sur 8 bits, codage des nombres -24 et -128 en (bs)

-24 est code en binaire signe par : 1 0 0 1 1 0 0 0(bs)

-128 hors limite nécessite 9 bits au minimum

Binaire signé :

Exercice :

1. Coder 100 et -100 en binaire signe sur 8 bits
2. Décoder en décimal $(11000111)_{(bs)}$ et $(00001111)_{(bs)}$

Solutions :

$100_{(10)} = (01100100)_{(bs)}$

$-100_{(10)} = (11100100)_{(bs)}$

$(11000111)_{(bs)} = -71_{(10)}$

$(00001111)_{(bs)} = 15_{(10)}$

Etendu de codage : pour une mémoire à n+1 bits (n>1), tous les entiers naturels de l'intervalle $[-(2^n - 1), (2^n - 1)]$ seront représentés.

- *soit pour n+1=8 bits, tous les entiers de l'intervalle [-127, 127]*
- *soit pour n+1=16 bits, tous les entiers de l'intervalle [-32767, 32767].*

Le nombre zéro est représenté dans cette convention (dites du zéro positif) par : **0**000...00000

Limitations du binaire signe:

- Deux représentations du zéro : + 0 et – 0, Il reste une configuration mémoire inutilisée : **1**000...00000
- Sur 4 bits : +0 = 0000(bs), -0 = 1000(bs)
- Multiplication et l'addition sont moins évidentes.

Code complément à 1

Le complément à 1 d'un nombre binaire, aussi appelé Complément Logique (CL) ou Complément Restreint (CR).

- les nombres positifs sont codés de la même façon qu'en binaire pure
- un nombre négatif est codé en inversant chaque bit de la représentation de sa valeur absolue
- Le bit le plus significatif est utilisé pour représenter le signe du nombre :

_ Si le bit le plus fort = 1 alors nombre négatif

_ Si le bit le plus fort = 0 alors nombre positif

Exemple : -24 en complément à 1 sur 8 bits

|-24|en binaire pur _ 0 0 0 1 1 0 0 0 $_{(2)}$ puis

On inverse les bits _ 1 1 1 0 0 1 1 1 $_{(cà1)}$

Limitation :

- deux codages différents pour 0 (+0 et -0), sur 8 bits :

  +0=0 0 0 0 0 0 0 $0_{(cà1)}$ et -0=1 1 1 1 1 1 1 $1_{(cà1)}$
- Multiplication et l'addition sont moins évidentes.

Exercices :

Coder 100 et -100 par complément à 1 (c à1) sur 8 bits

- $100_{(10)}$ = (01100100) $_{(c\ à1)}$
- $-100_{(10)}$ = (10011011) $_{(c\ à1)}$
- Décoder en décimal (11000111) $_{(c\ à1)}$ et (00001111) $_{(c\ à1)}$
- (11000111) $_{(c\ à1)}$= $-56_{(10)}$
- (00001111) $_{(c\ à1)}$= $15_{(10)}$

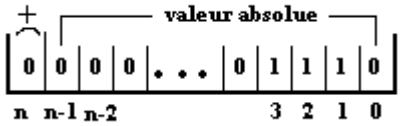
Complément à deux

Ce codage, purement conventionnel, aussi appelé Complément Vrai (CV) : et très utilisé de nos jours, est dérivé du binaire signé ; il sert à représenter en mémoire les entiers relatifs.

- Les nombres positifs sont codés de la même manière qu'en binaire pure.
- Un nombre négatif est codé en ajoutant la valeur 1 à son complément à 1.
- Le bit le plus significatif est utilisé pour représenter le signe du nombre, le reste

Supposons que la mémoire soit à n+1 bits, soit **x** un entier relatif à représenter.

- si **x > 0,** alors c'est la convention en *binaire signé* qui s'applique (le bit de signe vaut 0, les n bits restants codent le nombre), soit pour le nombre +14 :

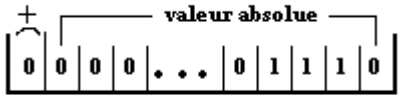


*+14 est représenté par **0**000...01110*

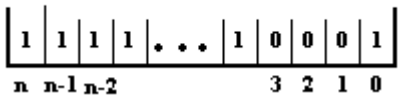
- si x < 0, alors

- on code **|x|** en binaire signé.
- puis l'on complémente tous les bits de la mémoire (complément à 1 ou complément restreint). Cette opération est un non logique effectué sur chaque bit de la mémoire.
- Enfin l'on additionne **1** au nombre binaire de la mémoire (addition binaire).

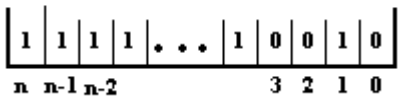
Exemple : soit à représenter le nombre -14

- 

codage de |-14|= 14

- 

complément à 1

- 

addition de 1

Le nombre -14 s'écrit donc en complément à 2 : 1111..10010.

Un des intérêts majeurs de ce codage est d'intégrer la soustraction dans l'opération de codage et de ne faire effectuer que des opérations simples et rapides (non logique, addition de 1).Un seul codage pour 0. Par exemple sur 8 bits :

+0 est code par $00000000_{(c\ à\ 2)}$

-0 est code par $11111111_{(cà1)}$

Donc -0 sera représenté par $00000000_{(c\ à\ 2)}$

Etendu de codage :

Avec n bits, on peut coder de -(2n-1) a (2n-1-1)

Sur 1 octet (8 bits), codage des nombres de -128 a 127

+0 = 00000000

-0=00000000

+1 = 00000001

-1=111111111

+127= 01111111

-128=10000000

Exercices-

Coder 100(10) et -100(10) par complément a 2 sur 8 bits

100(10) = 01100100(Ca2)

-100(10) = 10011010(Ca2)

Décoder en décimal $11001001_{(Cà2)}$ et $01101101_{(Cà2)}$

$11001001_{(Cà2)} = -55_{(10)}$

$01101101_{(Cà2)} = 109_{(10)}$

9. Codage des nombres réels

• Les formats de représentations des nombres réels sont :

Format virgule fixe : une représentation d'un nombre en virgule fixe est un type de donnée correspondant à un nombre qui possède (en base deux ou en base dix) un nombre fixe de chiffres après la virgule. Les nombres en virgule fixe sont utiles pour représenter des quantités fractionnaires utilisées par les premières machines.

Ils possèdent une partie entière et une partie décimale séparées par une virgule. Les bits à gauche de la virgule représentent la partie entière du nombre. Chaque bit à droite de la virgule, correspond à l'inverse d'une puissance de 2. Ainsi la première décimale binaire est ½, la seconde est ¼, la troisième est 1/8 et ainsi de suite. virgule.

La position de la virgule est fixe d'ou le nom.

•Exemple : $54{,}25_{(10)}$; $10{,}001_{(2)}$; $A1{,}F0B_{(16)}$

Etant donne une base b

Un nombre x est représente par :

$$x = a^{n-1}a^{n-2}\ldots a^{1}a^{0}, a^{-1}a^{-2}\ldots a^{-p}{}_{(b)}$$

- a^{n-1} est le chiffre de poids fort
- a^{-p} est le chiffre de poids faible
- n est le nombre de chiffre avant la virgule
- p est le nombre de chiffre après la virgule

La valeur de x en base 10 est : $x = \sum_{-p}^{n-1} a_i b^i$

Exemple :

$101,01_{(2)} = 1.2^2+0.2^1+1.2^0+0.2^{-1}+1.2^{-2} = 5,25_{(10)}$

Changement de base 10 ⟶ 2

Le passage de la base 10 a la base 2 est défini par :

- Partie entière, elle est codée sur p bits (division successive par 2)
- Partie décimale, elle est codée sur q bits en multipliant par 2 successivement jusqu'à ce que la partie décimale soit nulle ou le nombre de bits q est atteint.

Exemple : $4,25_{(10)} = ?_{(2)}$ format virgule fixe

$4_{(10)} = 100_{(2)}$

0,25 x 2= 0,5 ⟶ 0

0,5 x 2 = 1,0 ⟶ 1

Donc $4,25_{(10)} = 100,01_{(2)}$

Travaux Dirigés 1 — Codage de l'information —

Exercice 1 Changement de base

Q. 1.1: Convertir en nombres décimaux les nombres binaires suivants :

- 110, 1100, 100101110.

Correction

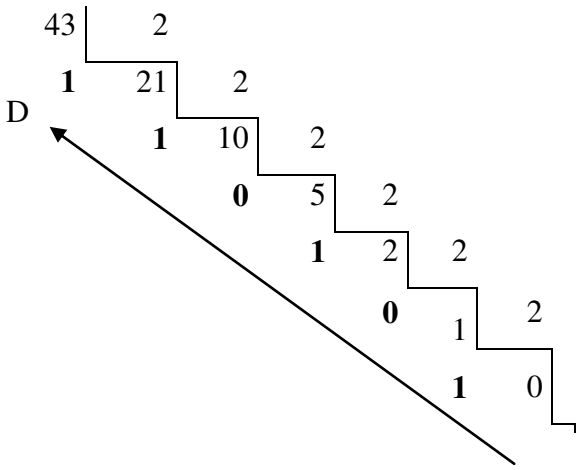
- $(110)_2 = 2^2 + 2^1 = 6$
- $(1100)_2 = 2^3 + 2^2 + = 12$
- $(100101111)_2 = 2^8 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0 = 303$

Q. 1.2 : Convertir en nombres binaires les nombres décimaux suivants :

43, 51, 128, 131, 202.

Correction

On va appliquer la méthode donnée dans le cours qui repose sur des divisions successives:



- $43 = (101011)_2$
- $55 = (110111)_2$
- $128 = (10000000)_2$
- $202 = (11001010)_2$

Q. 1.3 : Convertir en nombres binaires puis en nombres décimaux les nombres hexadécimaux suivants : 12, DADA et 5F3.

Correction

Pour la conversion hexadécimal→binaire :

- $(12)_{16} = (0001\ 0010)_2$
- $(DADA)_{16} = (1101\ 1010\ 1101\ 1010)_2$
- 5F316 = 0101 1111 00112

Pour la conversion hexadécimal→décimal :

- $(12)_{16} = 1\times161 + 2\times160 = 18$
- $(DADA)_{16} = 13\times16^3 + 10\times16^2 + 13\times161 + 10\times160 = 56\ 026$
- $(5F3)_{16} = 5\times16^2 + 15\times16^1 + 3\times16^0 = 1\ 523$

Exercice 2 Opérations binaires

Dans cet exercice, on va travailler sur des entiers naturels codés sur un octet.

Q. 2.1 : Quels sont les entiers naturels que l'on peut représenter sur un octet ?

Correction

Avec un octet (8 bits) on peut coder 2^8 valeurs : tous les entiers naturels de 0 (codé 0000 0000 en binaire) à $2^8 - 1 = 255$ (codé 1111 1111 en binaire).

Q. 2.2 : Réaliser les opérations suivantes en binaire puis en décimal (coder sur 1 octet).

(a) 0001 01012 +1011 01112

(b) 0100 01112 +1101 10012

Correction

(a) Calcul de 0001 01012 +1011 01112

```
        1 1 1 1 1    retenues
    0 0 0 1 0 1 0 1
+   1 0 1 1 0 1 1 1
-----------------------
    1 1 0 0 1 1 0 0      = 2^7 +2^6 +2^3 +2^2 = (204)10
```

(b) Calcul de $0100\ 0111_2 + 1101\ 1001_2$

```
      1 0 1 111 1 1      retenues
    0 1 0 0 0 1 1 1
+   1 1 0 1 1 0 0 1
-----------------------
  1 0 0 1 0 0 0 0 0      = 2^5 = (32)10
```

On observe ici un dépassement de capacité : le résultat est codé sur 9 bits alors qu'on ne dispose que de 8 bits. Le résultat de l'opération est donc erroné.

Exercice 3 : codage d'entiers (naturels ou relatifs) :

1- Coder sur **1 octet** les nombres suivants en : bs, cà1, cà2 :

-33, +117,-59, -138

2- Décoder en décimal les nombres binaires suivants :

$(11001101)_{cà2}$,($10001101)_{cà1}$, $(00110101)_{cà2}$,($10110101)_{bs}$

Correction

Décimal	+117	-33	-59	-138
Binaire	+1110101	-100001	-111011	-10001010
Valeur absolue	|+117|=01110101 Sur 1 octet	|-33|=00100001 Sur 1 octet	|-59|=00111011 Sur 1 octet	|-138|=0000000010001010 Sur 2 octets
Binaire signé	**0**1110101 signe positif	**1**0100001 signe négatif	**1**0111011 signe négatif	**1**000000010001010 signe négatif
Cà1	01110101 identique au bs puisque c'est positif	11011110 inverser les bits de la valeur absolue mise sur 1 octet	11000100 inverser les bits de la valeur absolue mise sur 1 octet	1111111101110101 inverser les bits de la valeur absolue mise sur 2 octets
Cà2	01110101 identique au bs puisque c'est positif	11011111 cà1+1	11000101 cà1+1	1111111101110110 cà1+1

1-

1. $(\mathbf{1}1001101)_{cà2}$ le nombre est **négatif**

   $(11001101)_{cà2}$ -1 = $(11001100)_{cà1}$ inversement (00110011) = 51 en valeur absolue

   Le nombre final = -51

   ($\mathbf{1}0001101)_{cà1}$ le nombre est **négatif**

   ($10001101)_{cà1}$ inversement (01110010) = 114 en valeur absolue

   Le nombre final = -114

   $(00110101)_{cà2}$ le nombre est **positif**

   Le cà2 d'un nombre positif =cà1 = bs = valeur binaire du nombre

   $(00110101)_{cà2}$ =$(00110101)_{cà1}$ = $(00110101)_{bs}$ = $(00110101)_2$ =$(53)_{10}$

Exercice 4 : arithmétique binaire :

Calculer en **cà2** le résultat des opérations suivantes (décodé en fin en décimal):

20-15 ; 23-46

Correction

20-15 = 20 + -15
20 = +10100 = 00010100 = $(00010100)_{cà1}$ = $(00010100)_{cà2}$
-15 = -1111 = -00001111 = $(11110000)_{cà1}$ = $(11110001)_{cà2}$
20-15 = $(00010100)_{cà2}$ + $(11110001)_{cà2}$ =$(00000101)_{cà2}$ normalement 100000101 mais sur 8 bits donnera 00000101 et le 1er 1 superflu éliminé.

Résultat décodé :(**0**0000101) $_{cà2}$: le nombre est **positif** décodage directe du binaire au décimal 00000101= 101=5 d'où 20-15=5

23-46 = 23 + -46
23= +10111 = 00010111=$(00010111)_{cà1}$ = $(00010111)_{cà2}$
-46= -101110= -00101110 = $(11010001)_{cà1}$ = $(11010010)_{cà2}$
23-46 =(00010111) $_{cà2}$ + $(11010010)_{cà2}$ = (11101001) $_{cà2}$

Résultat décodé :(**1**1101001) $_{cà2}$: le nombre est **négatif**
11101001 – 1 = (11101000) $_{cà1}$
Inversement = $(00010111)_2$
$(00010111)_2 = |23|$
=-23

Exercice 5 : codage des nombres réels

1. Coder en virgule fixe (sur 32 bits : 1 bit signe, 15 bits partie entière, 16 bits partie flottante) les nombres suivants : -125. 296875 ; 8.625
   Codage de -125.296875
2. En binaire : -125.296875 = $(-1111101.010011)_2$

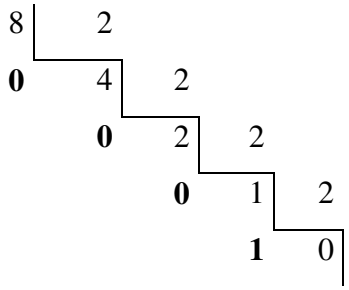
125	2	
1	62	2
	0 31	2
	1 15	2
	1 7	2
	1 3	2
	1 1	2
	1 0	

3. 0.296875 x2 = **0**.59375
4. 0.59375 x2= **1**.1875
5. 0. 1875 x2= **0**.375
6. 0.375 x2= **0**.75
7. 0.75 x2= **1**.5
8. 0.5 x2= **1**.0

Virgule fixe : **1** 000000001111101**.**0100110000000000

Codage de 8.625

En binaire virgule fixe : **0** 000000000000**1000.101**0000000000000



0.625 x2 = **1.**25

0.25 x2= **0**.5

0. 5 x2= **1**.0

III. Généralités sur l'algorithmique et la programmation :

1. Introduction :

L'informatique traite de deux aspects complémentaires : les programmes immatériels (logiciel, software) qui d´écrivent un traitement à réaliser et les machines (matériel, hardware) qui exécutent ce traitement. Le matériel est donc l'ensemble des éléments physiques utilisés pour traiter les données.

- Pourquoi apprendre l'algorithmique pour apprendre à programmer ?
- En quoi a-t-on besoin d'un langage spécial, distinct des langages de programmation compréhensibles par les ordinateurs ?

Parce que l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage. Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique. Cette dimension est présente quelle que soit le langage de programmation ; mais lorsqu'on programme dans un langage on doit en plus se colleter les problèmes de syntaxe, ou de types d'instructions, propres à ce langage. Apprendre l'algorithmique de manière séparée, c'est donc sérier les difficultés pour mieux les vaincre.

2. Cycle de développement d'un programme:

Le cycle de développement d'un "programme informatique " peut se résumer ainsi :

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

- **Problème**
- **Analyse :** phase de réflexion qui permet d'identifier les caractéristiques du Problème à traiter
- **Algorithme :** Il s'agit d'une description compréhensible par un être humain de la suite des opérations à effectuer pour résoudre le Problème.
- **Programme :** fichier texte des instructions et de leurs enchaînements dans un langage informatique.
- **Compilation :** transformation en langage machine d'un programme écrit en langage évolué
- **Exécution :** l'ordinateur exécute les instructions d'un programme en langage "binaire".

3. Historique :

En informatique, l'algorithmique désigne l'ensemble des règles et des techniques qui sont impliquées dans la définition et la conception des algorithmes. Le mot « algorithme » vient du nom du mathématicien Al Khawarizmi, qui, au 9ème siècle écrivit le premier ouvrage systématique sur la solution des équations linéaires et quadratiques.

La notion d'algorithme est donc historiquement liée aux manipulations numériques, mais elle s'est progressivement développée pour porter sur des objets de plus en plus complexes : des textes, des images, des formules logiques, des objets physiques, etc. L'ordinateur est une machine électronique, qui permet de traiter des informations selon des séquences d'instructions prédéfinies (un programme). L'objectif est apprendre à écrire des programmes.

Avec quelles conventions écrit-on un algorithme ?

- Historiquement, plusieurs types de notations ont représenté des algorithmes. Il y a eu notamment une représentation graphique, avec des carrés, des losanges, etc. qu'on appelait **des organigrammes**.

- Aujourd'hui, cette représentation est quasiment abandonnée, pour deux raisons.

  - D'abord, parce que dès que l'algorithme commence à grossir un peu, ce n'est plus pratique du tout.

  - Ensuite parce que cette représentation favorise le glissement vers un certain type de programmation, dite non structurée, que l'on tente au contraire d'éviter.

- C'est pourquoi on utilise généralement une série de conventions appelée « **pseudo-code** », qui ressemble à un langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe.

4. Le langage informatique :

Un langage informatique est un **code de communication**, permettant à un être humain de dialoguer avec une machine en lui soumettant des instructions et en analysant les données matérielles fournies par le système.

- Le langage informatique est l'intermédiaire entre le programmeur et la machine. Il permet d'écrire des programmes destinés à effectuer une tache donnée.

Exemple : un programme de résolution d'une équation du second degré.

5. Programmation :

La programmation est un ensemble des activités orientées vers la conception, la réalisation, le test et la maintenance de programmes. L'exécution d'un programme en langage d'assemblage reste une tache fastidieuse et de tels programmes dépendent de la machine pour laquelle ils ont été conçus. Les langages de programmation tels que **Fortran, Cobol, Pascal, Basic, C,**

Exemples : Langages procéduraux : **…**

Langages orientés objets : **C++, , C#, SmallTalk, …**

5.1. Généralités sur la programmation en Fortran :

Le Fortran est un ensemble de règles syntaxiques et grammaticales et de mots permettant d'écrire des opérations mathématiques, de communiquer des informations entre l'ordinateur (mémoire), les fichiers (disque dur) et l'utilisateur (clavier/écran).

5.2. Historique :

La première version du Fortran "FORmula TRANslator" née en 1954 chez IBM. Différentes versions/améliorations successives (Fortran77, Fortran90, Fortran95, Fortran2003, Fortran2008 ...). Par conséquent, tout ce qui est dit ici devrait rester valable avec les normes futures, dont les plus utilisées sont Fortran77 et Fortran90.

6. Notion d'algorithme :

Un algorithme est une suite organisée d'opérations élémentaires (actions ou instructions) qui doivent être exécutées dans un ordre bien déterminé pour résoudre un problème.

Exemples : recette de cuisine, montrer le chemin à un touriste, programmer un magnétoscope…

Exemple : Mode d'emploi d'un télécopieur :

Extrait du mode d'emploi d'un télécopieur concernant l'envoi d'un document.

1. Insérez le document dans le chargeur automatique.
2. Composez le numéro de fax du destinataire à l'aide du pavé numérique.
3. Enfoncez la touche envoi pour lancer l'émission.

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

7. Caractéristiques d'un algorithme :

- La réalisation d'un algorithme est un acte créatif basé sur la logique.
- Un algorithme doit être fini et doit se terminer après un nombre fini d'opérations.
- Un même problème peut être résolu au moyen de plusieurs algorithmes.

8. Analyse d'un problème:

Lorsqu'on cherche un algorithme permettant d'automatiser un traitement donné, il faut suivre les étapes suivantes :

Etape 1 : Lire et comprendre bien l'énoncé du problème à résoudre.

Etape 2 : Définir les résultats du problème (les sorties), Définir les données du problème (les entrées), Définir le traitement (les relations permettant d'obtenir les résultats à partir des données).

Etape 3 : Ecrire l'algorithme on respectant la structure pseudo-code.

9. La démarche et analyse d'un problème :

- L'élaboration d'un algorithme est une démarche de résolution de problème exigeante.
- La rédaction d'un algorithme est un exercice de réflexion qui se fait sur papier.

Figure 3.1 : Du problème au programme

Exemples de problèmes :

- Addition de deux nombres ;
- Calcul de la factorielle d'un nombre ;
- Dessiner la courbe d'une fonction ;
- Calculer la moyenne des notes des étudiants ; …

10. La représentation d'un algorithme :

On peut représenter un algorithme à l'aide d'un pseudo-code ou d'un organigramme

Organigramme (algorigramme ou Ordinogramme) :

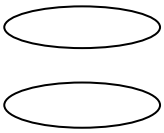
11. Organigramme :

L'organigramme est une représentation graphique de l'algorithme. Pour le construire, on utilise des symboles normalisés. Les organigrammes sont constitués de symboles généralement normalisés qui donnent une représentation de l'algorithme plus facile à suivre et à contrôler qu'une représentation littérale

Symbole général Opération ou groupe d'opérations sur des données, des instructions,…	**Sous-programme** Portion de programme considérée comme une simple opération	**Branchement** Exploitation de conditions variables impliquant un choix parmi plusieurs	**Renvoi**

Un algorithme doit avoir

- Un et un seul point d'entrée:
- Un et un seul point de sortie:



Ils sont appelés symboles terminaux

Les symboles sont reliés par des lignes de flux

Chaque symbole d'opération dispose de

- Un et un seul flux d'entrée
- Un et un seul flux de sortie

12. Structure d'un algorithme :

Lorsqu'on écrit un algorithme, il faut respecter la syntaxe (règles d'écriture d'un langage donné), un algorithme écrit en pseudo-code est composé de trois parties suivantes :

L'en-tête, la partie déclarative et **le corps.**

Un algorithme prend généralement la structure suivante :

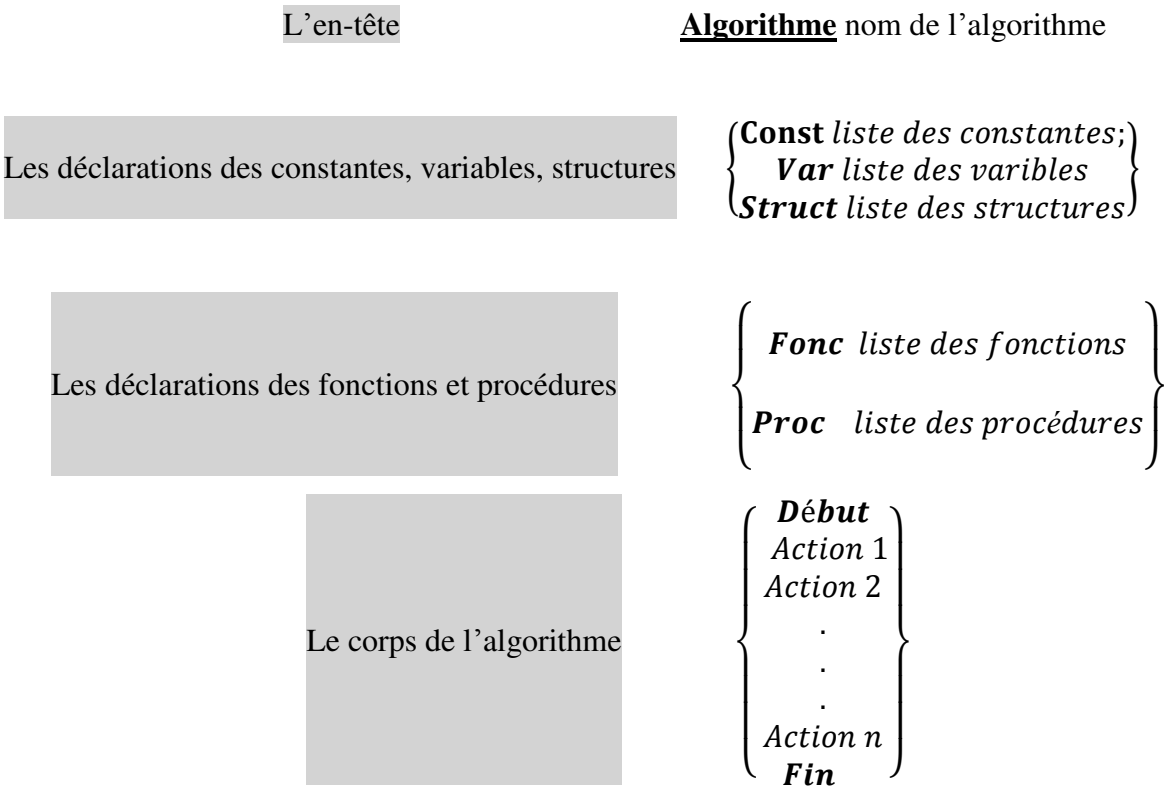


Figure 1 : Structure générale d'un algorithme

12.1. L'en-tête d'un algorithme :

L'en-tête d'un algorithme est constitué du mot Algorithme, suivi d'un nom identifiant l'algorithme.

12.2. La partie déclarative :

Comprend une liste des variables et des constantes utilisés et manipulés dans le corps de l'algorithme.

12.2.1 Les données d'un algorithme :

Les données (objets) sont des informations nécessaires au déroulement d'un algorithme. On distingue deux catégories **: les constantes** et **les variables.**

a) **Les variables :** Une variable sert à stocker la valeur d'une donnée dans un langage de programmation, dont le contenu peut être modifié par une action durant l'exécution d'un algorithme. Une variable est caractérisée par son nom, sa valeur et son type.

Syntaxe : Type Nom_Variable

Exemple : Variable nb élèves : entier ;

Variable note : réel ;

Variable Prénom : chaines de caractères ;

b) **Les constantes :** Une constante est une donnée fixe qui ne varie pas durant l'exécution d'un algorithme. Une constante est caractérisée par son nom et sa valeur (fixe)

Syntaxe **:** Const Nom _Constante = valeur

Exemple **:** Constante Pi =3,14

Constante A = 8

12.2.2. Les types de données :

Un programme peut être amené à manipuler différents types de données :

a) **entiers** : valeurs numériques entières pouvant être signées ou non signées ;

Syntaxe : Entiers : integer i, ou integer*2 i (simple précision) ; integer*4 i (double précision)

sur n bits, les nombres représentables sont :

$-2n-1 \leq i \leq 2n-1-1$

D'où leur domaine d'utilisation est :

-128 à 127 (1 octet) pour INTEGER*1 (pour un entier coder sur 1 octet).

-32 768 à 32 767 (2 octets) pour INTEGER*2 (pour un entier coder sur 2 octets).

-2 147 483 648 à 2 147 483 647 (4 octets) pour INTEGER*4 (pour un entier coder sur 3 octets).

b) **réels :** valeurs numériques codées avec une mantisse et un exposant ;

real a, ou real*4 a (simple précision) les nombres sont représentés sur 32 bits. Ce type de réel, appelé réel simple précision;

real*8 a (double précision) Ce type de représentation permet de représenter les nombres sur 64 bits

c) **complexes** : complex z, ou complex*8 z (simple précision) ; complex*16 z (double précision)

d) **booléen :** valeur pouvant être soit Vraie, soit Fausse ;

e) **caractère** : octet correspondant à un code ASCII ; (A, B, a, b, &…), c'est le type qui permet de gérer les chaînes de caractères. Les chaînes de caractères sont écrites indifféremment entre guillemets ou entre apostrophes. Type caractère tels que : ′a′, ′B′, ′*′, ′9′, ′@′, ′ ′,…

f) **chaîne de caractères** : ensemble de caractères ; (mot)

Syntaxe : character nom*n

N.B.: n peut être le nombre de caractères de la chaine (ex. : character nom*4 pour nom=toto), ou bien n=1 si la chaîne n'est pas spécifiée.

Le nombre après '*' représente le nombre de byte (octets) occupés en mémoire par une variable. Un caractère occupe, quant à lui, 1B, voilà pourquoi il faut spécifier la longueur de la chaine de caractères.

- **tableau de données** : ensemble de données de même type (tableau d'entiers, tableau de réels).

NB/Toutes ces données sont codées sous forme d'octets en mémoire.

12.3. Le corps d'un algorithme:

Le corps d'un algorithme est une suite d'instructions ou des tâches à exécuter.

12.3.1 Les instructions lectures/écritures

Les instructions d'entrée/sortie permettent de transférer des informations entre la mémoire centrale et les unités périphériques (terminal, clavier, disque, imprimante,…). Les opérations d'entrée/sortie engendrent des conversions caractères alphanumériques/binaires.

On distingue deux types d'instructions de lectures/écritures :

1. **Formatées**

   C'est à dire organisée en lignes de caractères. C'est le cas des lectures clavier et des écritures écran, des lectures/écritures de fichiers texte et de chaînes.

2. **non-formatées**

   Qui signifie transfert octet par octet. C'est le cas des lectures/écritures sur fichiers binaires.

a) L'instruction d'entrée (Lire)

Un des moyens d'affecter une donnée à une variable est l'utilisation d'une instruction d'entrée des données appelée instruction de lecture (Fonction d'entrée) : Instruction qui permet d'entrer des données tapées au clavier.

Syntaxe : Lire (variable1),

Lire (variable2),

Lire (variable1, variable2),

Exemple : Lire (a, b)

Cette instruction affectera les valeurs 2 et 7.6 respectivement à a et b.

On constate que a est une entier et b un réel.

Lecture formatée

La lecture formatée s'applique au clavier et aux fichiers texte

Syntaxe générale

- Read (périphérique, format [, options]) liste
- Read format, listePour lire : READ(*,*) a

La machine met la valeur entrée au clavier dans la zone mémoire.

Remarques : Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la saisie de l'entrée attendue par le clavier et de la touche Entrée (cette touche signale la fin de l'entrée) ;

Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper.

b) L'instruction de sortie (Ecrire)

Instruction qui permet d'afficher le contenu d'une variables ou/et un message sur l'écran.

Syntaxe : Ecrire (variable),

Ecrire ('message'),

Ecrire ('message', variable).

Formats d'écriture

Un format est une série de codes, chaque code définissant le format d'écriture d'un élément d'une donnée.

Format standard :

Pour écrire : WRITE(*,*) 'Entrer la valeur de a'

Définition du format

Deux solutions :

- Directement dans l'instruction **write** avec une chaîne de caractères :

```
write(*,'format') liste
```

- Dans une ligne libellée contenant l'instruction **format** :

```
nn format(définition du format)
write(*,nn)
```

- Cette solution permet d'utiliser le même format dans plusieurs instructions **write**.

c) L'affectation : est une opération qui consiste à attribuer une valeur à une variable. Elle est représentée par une flèche orientée à gauche ⟵ .

Syntaxe **:** Variable ⟵ Valeur ou expression

Remarque :

1. L'instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche. La valeur de la partie droite doit obligatoirement être du type de la variable dont la valeur est modifiée.
2. En algorithmique, le signe de l'affectation est le signe ←. Mais en pratique, la plupart des langages de programmation emploient le signe égal. Et là, la confusion avec les maths est également facile.
3. En maths, A = B et B = A sont deux propositions strictement équivalentes. En informatique, absolument pas, puisque cela revient à écrire A ← B et B ← A, deux choses bien différentes.

12. 3.2. Opérateurs élémentaires et Expressions :

Opérateur : C'est un symbole d'opération qui permet d'agir sur des variables ou de faire des "calculs",

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

Expression : C'est une combinaison d'opérateur(s) et d'opérande(s), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur et un type.

Expressions arithmétiques :

Une expression arithmétique est une combinaison de constante, de variables indicées ou non, de noms de fonctions, d'opérations arithmétiques et de parenthèses. Les opérandes ne doivent pas être de type logique. Rappelons que les opérateurs arithmétiques sont:

- + addition,
- - soustraction,
- * multiplication,
- / division
- entiers div, mod :

  a div b : quotient euclidien de a par b, exemple : 11 div 2 vaut 5

  a mod b : reste de la division euclidienne de a par b, exemple : 11 mod 2 vaut 1

Une expression arithmétique peut contenir soit :

Le type d'une expression est le type de ses opérandes s'ils sont de même type. Dans une expression de type mixte. Une expression arithmétique peut contenir :

- Une constante numérique
- Une variable numérique
- Une combinaison des précédentes avec des opérateurs et des parenthèses

Exemple : (a+b)*c

Expressions booléennes :

Une expression logique est composée d'éléments logiques reliés par des opérateurs logiques et à la valeur « vrai » ou « faux » ;

Exemple - une constante booléenne : (vraie ou fausse) ;

- une variable booléenne ;
- des expressions arithmétiques combinées à des opérateurs relationnels ;
- des expressions booléennes combinées à des opérateurs booléens.

Opérateurs de comparaison : <, >, ≤, ≥, =, ≠

Opérateurs logiques (booléens): et, ou, non, non et, non ou..

Exemples : a, b des entiers

c, d des booléens

a=b : expression booléenne dont la valeur est vraie si a=b et faux si a<>b

p et q

p et non q

(a=b) ou q

Tout comme en arithmétique les opérateurs ont des priorités Par exemple * et / sont prioritaires sur + et -

Pour les booléens, la priorité des opérateurs est non, et, ou Exclusif et ou ;

Pour clarifier les choses on peut utiliser des parenthèses.

13. Structures de contrôle d'un algorithme :

En programmation procédurale comme en algorithmique, l'ordre des instructions est primordial. Le processeur exécute les instructions dans l'ordre dans lequel elles apparaissent dans le programme. On dit que l'exécution est **séquentielle.** Une fois que le programme a fini une instruction, il passe à la suivante. Tant qu'une instruction n'est pas terminée, il attend avant de continuer. Les structures algorithmiques sont réparties en 3 catégories :

- Succession linéaire d'opérations,
- Structures conditionnelles,
- structures répétitive.

13.1. Structure séquentielle:

C'est une suite d'instructions, les instructions sont exécutées dans l'ordre l'une après l'autre. Une instruction se termine toujours par ;

Syntaxe :

Début
Instruction 1 ;
Instruction 2 ;
.
.
Instruction n ;

Fin

Exemple : Ecrire un algorithme qui calcul la surface d'un rectangle

Algorithme Surfaces
Variable longueur, largeur : Entier {déclaration réservation d'espace mémoire}

Début

Ecrire (donner les cotes du rectangle) {préparation du traitement}
Lire (longueur, largeur)
Surface ⟵ Langueur* largeur

Ecrire (Surface)

Fin

Exercice : Ecrire un algorithme qui demande la valeur du rayon pour calculer la surface d'un cercle.

Etape 1 : on veut calculer la surface d'un cercle

Etape 2 : Résultat : La surface du cercle Surf,

Données : Le rayon R, 3,14 Pi,

Traitement Surf = Pi*R*R

Etape 3 : Syntaxe

```
Algorithme Surface_cercle
Constante Pi = 3,14
Variable R, Surf : Réel
Début
    Ecrire (' Donnez la valeur de rayon: ')
    Lire (R) ;
    Surf ←——Pi*R^2
    Ecrire (' La surface de cercle est : ', Surf)
Fin
```

Exercice : Ecrire une programme qui permute deux entiers

```
program permutation
implicit none
integer*4 a,b,temp
write(*,*) 'entrer la valeur de a'
read(*,*) a
write(*,*) 'entrer la valeur de b'
read(*,*) b
temp=a
a=b
b=temp
write(*,*) 'La nouvelle valeur de a est :', a,
& 'La nouvelle valeur de b est :', b
END
```

Exercice : Ecrire un algorithme qui permet de calculer le produit de 2 nombres puis le traduire en programme Fortran

```
Algorithme  Produit ;
Réel : a, b, P
Début
   Ecrire (' Donnez le nombre a: ') ;
   Lire (a) ;
   Ecrire (' Donnez le nombre b: ') ;
    Lire (b) ;
     P ← a*b ;
   Ecrire (' Le produit est : ', P) ;
Fin
```

```
Program produit
Real     a, b, p

write (*,*)  ' Donner le nombre a '
Read (*,*) a
Write (*,*)' Donner le nombre b '
Read(*,*) b
          p=a*b
Write (*,*)' Le produit est :',p
 End
```

Travaux Dirigés 2- Introduction à l'Algorithmique

Exercice 1:

Considérons les algorithmes ci-dessous. Quel sera le contenu des variables a, b et éventuellement c après leur exécution ?

a)

Algo A1
Entier a, b, c :
Debut
a ← 1
b ← 2
b ← a + b
c ← a + b
Fin

Correction
a)
a = 1
b = 3
c = 4

b)

Algo A2

Caractère a, b

Début

a ← '1'

b ← '2'

a ← a + b

Fin

Correction
b)
Ne marche pas. On ne peut pas additionner des caractères

Exercice 2:

A- Quel est l'ordre de priorité des différents opérateurs de l'expression suivante :

a + b ** c / d / e ; a ** b ** c * d + e ; ((3 * a) – x ^ 2) – (((c – d) / (a / b)) / d)

Correction

A a+ b ** c / d / e
4 1 2 3
a ** b ** c * d + e
2 1 3 4
((3 * a) – x ^ 2) – (((c – d) / (a / b)) / d)
1 3 2 8 4 6 5 7

B- Que valent les réels suivants? *Correction*

R1=12./3./2.	2
R2=12./3.*2.	8
R3=2/3	0
R4=R1+24.*3./(-2.)**3+2.*(5/2+1)	-1
R5=12/ (3/2)	12

Exercice 3

Sachant que a = 4, b = 5, c = -1 et d = 0, évaluer les expressions logiques suivantes :

1- (a < b) ET (c >= d) 2- NON (a < b) OU (c # b) 3- NON ((a # b ^ 2) OU (a*c < d))	*Correction* 1- Faux 2- Vrai 3- Faux

NB / *Le résultat d'une expression logique est toujours Vrai ou Faux*

13.2 Les Structures conditionnelles :

Les structures conditionnelles, ne permettent d'exécuter certaines instructions, que sous certaines conditions. Une condition (expression conditionnelle ou logique) est évaluée, c'est à dire qu'elle est jugée vraie ou fausse. Si elle est vraie, un traitement (une ou plusieurs instructions) est réalisé; si la condition est fausse, une autre instruction va être exécutée, et ensuite le programme va continuer normalement.

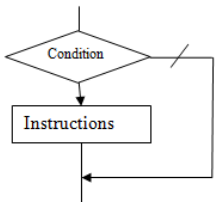
13.2.1 Structures conditionnelles simples :

Syntaxe **Format Organigramme**

Si <condition> **Alors**

Instructions 1

Finsi



13.2.2 Les structures alternatives :

Dans le déroulement d'un algorithme, on doit souvent choisir entre deux actions, suivant une condition concernant la valeur de certaines données. La structure alternative va permettre d'effectuer des choix.

Syntaxe: **Si** <condition> A**lors**
<traitement1>
Sinon
<traitement2>
Finsi

Ce type d'instructions incorpore deux blocs d'instructions dont un et un seul bloc sera exécuté, de manière alternative, en fonction de la condition.

Exemple : Affiche la valeur absolue d'un entier.

```
Algorithme Valeur Absolue I ;
Réel : x, y
Début
    Ecrire (" Entrez un réel : ")
    Lire (x)
    Si (x < 0) alors
        Ecrire ("la valeur absolue de ", x, "est:",-x)
    Sinon
        Ecrire ("la valeur absolue de ", x, "est:",x)
    Finsi
Fin
```

En Fortran les structures conditionnelles sont écrites dans les formes suivantes

– IF / ELSE / END

If (condition) **then**

instructions1

Else

instructions2

End if

Remarque : les identificateurs logiques sont déterminés comme suit :

.eq. (=), .ne. (6=), .gt. (>), .ge. (>), .le. (<), .lt. , .and. , .or. , .not.

Syntaxe

Si <condition> **Alors**

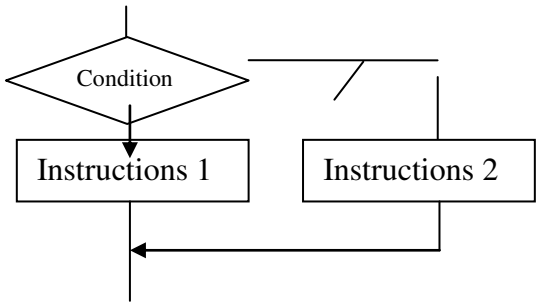
Instructions 1

Sinon

Instructions 2

Finsi

Format Organigramme



program division

real*4 a,b,c

write(*,*) ’entrer la valeur de a et b’

real(*,*) a,b

if (b .eq. 0) **then**

write(*,*) ’Division impossible !’

else

c=a/b

write(*,*) ’Le resultat est :’, c

endif

end

13.2.3. Structures conditionnelles imbriquées :

Un **test** est une instruction qui permet d'effectuer un traitement différent selon qu'une condition est vérifiée ou non. Le **test imbriqué** est une généralisation de la structure de contrôle conditionnelle, lorsque le nombre de traitements différents est plus grand que deux.

Par exemple, un programme devant donner l'état de l'eau selon sa température doit pouvoir choisir entre trois réponses possibles (solide, liquide ou gazeux).

L'algorithmique actuelle propose une syntaxe basée sur l'emploi de plusieurs conditions imbriquées.

Syntaxe: Si <condition 1> A**lors**

```
    Si <condition 2> alors
        <instructions A>
    Sinon
        <Instructions B>
    Finsi
Sinon
    Si <condition 3> alors
        <Instructions C>
    Finsi
Finsi
```

Exemple : Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif ou nul. Attention toutefois : on ne doit pas calculer le produit des deux nombres.

13.2.3 La structure de choix (choix multiples)

La structure de choix permet, en fonction de plusieurs conditions de type booléen, d'effectuer des actions différentes suivant les valeurs d'une seule variable.

Syntaxe :

Suivant (valeur ou expression) **faire**

Cas valeur1 : action1 ;

Cas valeur2 : action2 ;

.

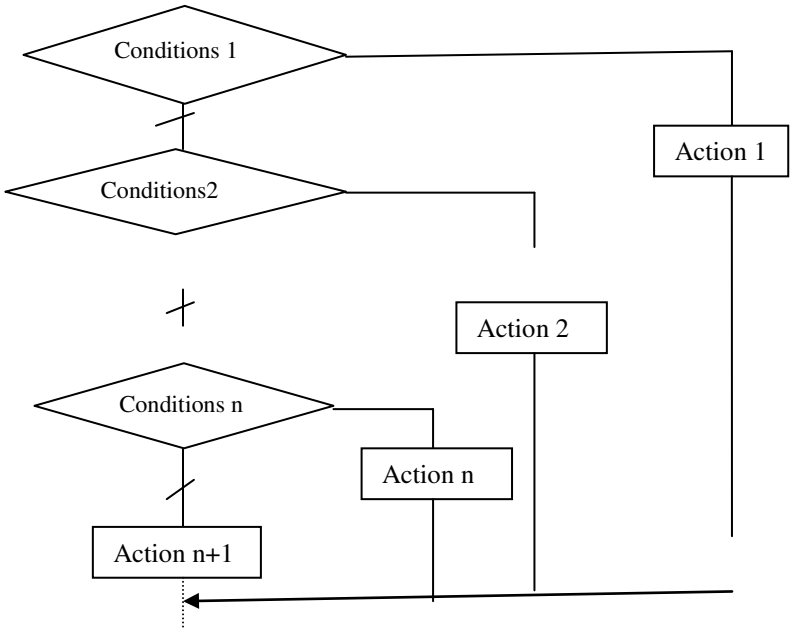
.

.

Cas valeur N : action N ;

Autres cas : action N+1 ;

Fin suivant ;



Travaux Diriges 3 - Structures de contrôle conditionnel

Exercice 1:

Ecrire un algorithme puis un organigramme permettant de trouver et d'afficher la plus petite valeur entre trois nombres réels distincts A, B et C

Correction

Algorithme :

```
Algo petite
Début
Variables A, B, C : réel
Lire (A, B, C)
Si (A<B) alors
Si (A<C) alors
Ecrire A
Si non
Ecrire C
Fin Si
Sinon
Si (B<C) alors
Ecrire B
Si non
Ecrire C
Fin Si

   Fin
```

Exercice 2:

Écrire un algorithme qui demande un réel à l'utilisateur et affiche sa valeur absolue (sans utiliser de fonction prédéfinie évidemment).

Correction

```
algo absolue
Début
Réel a
Lire "tapez un réel", a
Si (a>0) alors
Écrire "|", a,"|=", a
Sinon
Écrire "|", a,"|=",-a
Fin si
Fin
```

Exercice 3 :

Ecrire un algorithme qui demande à l'user d'entrer la note est qui affiche la mention comme suite :

Faible ; si note<=10
Passable ; si 10<=note<=12

A. Bien ; si 12<=note<=14
Bien ; si 14<=note<=16
T. Bien; si 16<=note<=18
Excellent ; si 18<=note<=20

Correction

```
Algo mention
Variable note : réel
Début
Ecrire (entrer la note :)
Lire (note)
Si (note <10) alors
Ecrire (faible)
Si non
Si (note <12) alors
Ecrire (passable)
Si non Si (note <14) alors
Ecrire (A. Bi en)
Si non Si (note <16) alors
Ecrire (Bien)
Si non Si (note <18) alors
Ecrire (T. Bien)
Si non
Ecrire (Excellent)
Fin Si
Fin
```

Exercice 4 :

Ecrire un algorithme permettant de résoudre une équation du second degré.

Correction

```
Algo second_degré
Début
Réel a, b, c, delta
Début : Ecrire (saisissez les valeurs a, b et c de l'équation ax²+bx+c=0 )
Lire (a, b, c)
Si (a=0) alors
Écrire (équation du premier degré)
Sinon
Delta=b²-4*a*c
Si (delta>0) alors
Écrire (les solutions de l'équation sont)
Écrire (-b-sqrt(delta))/(2*a), (-b+sqrt(delta))/(2*a))
SinonSi (d=0) alors
Écrire (-b/ (2a))
Sinon
Écrire (pas de solutions réelles)
Fin Si
Fin
```

13.3. Structures itératives ou répétitives:

Une boucle permet de répéter une instruction (ou une liste d'instructions) plusieurs fois.
Il y a principalement deux types de boucles

a) Les boucles pour répéter une instruction un certain nombre de fois, il s'agit de la boucle **Pour**

b) Les boucles pour répéter une instruction jusqu'à une condition d'arrêt, il s'agit des boucles **Tant que**

Le passage dans une boucle est appelé **itération**

13.3.1. Les boucles Tant que :

On ne connait pas le nombre d'itérations à effectuer, mais à chaque itération, on vérifie si la condition est vraie ou fausse. Dés que cette condition est fausse, on sort de la boucle**.**

Syntaxe : TantQue (condition)
instructions
FinTantQue

La condition (condition de contrôle de la boucle) est évaluée avant chaque itération

- si la condition est vraie, on exécute les instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution, …
- si la condition est fausse, on sort de la boucle et on exécute l'instruction qui est après FinTantQue.
- Il est possible que les instructions à répéter ne soient jamais exécutées.
- Le nombre d'itérations dans une boucle Tant Que n'est pas connu au moment d'entrée dans la boucle. Il dépend de l'évolution de la valeur de la condition.

Une des instructions du corps de la boucle doit absolument changer la valeur de la condition de vrai à faux (après un certain nombre d'itérations), sinon le programme va tourner indéfiniment

En Fortran la boucle Tant que s'écrit de la forme suivante

dowhile (condition)
instructions
enddo

13.3.2. Les boucles Pour :

Les boucles permettent de répéter une instruction un nombre donné de fois.

Elles se caractérisent par le fait que l'on connait à l'avance le nombre d'itérations que l'on va devoir effectuer.

Syntaxe Pour compteur **allant d'initiale à** finale par **pas** valeur du pas

Instructions

FinPour

Remarque : le nombre d'itérations dans une boucle *Pour* est connu avant le début de la boucle

Compteur est une variable de type entier (ou caractère). Elle doit être déclarée

Pas est un entier qui peut être positif ou négatif.

Pas peut ne pas être mentionné, car par défaut sa valeur est égal à 1.

Expressions de même type que compteur.

Déroulement des boucles Pour :

1) La valeur initiale est affectée à la variable compteur

2) On compare la valeur du compteur et la valeur de finale :

a) Si la valeur du compteur est > à la valeur finale dans le cas d'un pas positif (ou si compteur est < à finale pour un pas négatif), on sort de la boucle et on continue avec l'instruction qui suit FinPour

b) Si compteur est <= à finale dans le cas d'un pas positif (ou si compteur est >= à finale pour un pas négatif), instructions seront exécutées

Syntaxe de la boule pour en Fortran

```
do i=début, fin, pas
        Instructions
   enddo
```

Exercice : Ecrire un programme qui permet de calculer le factoriel d'un entier N.

NB : utiliser la boucle **Pour**

Solution :

```
 Program Factoriel
Integer  N, Fact
Read(*,*) N
Fact=1
Do i=1,N
Fact=Fact*i
Enddo
Write(*,*) Fact
End
```

Travaux Dirigés 4 - Les structures itératives

Exercice 1:

1- Ecrire un programme fortran qui lit une valeur entière n, puis calcule et écrit les n premiers termes de la suite $U_{n+1} = 2Un + 3$, sachant que $U_0 = 1$.

2- Écrire un algorithme pour afficher les premiers termes des suites suivantes (nombre de termes (n) demandé à l'utilisateur) :

Suite arithmétique : $U_{n+1} = U_n + 2$, $U_0 = 1$

Correction

```
PROGRAM suite1
INTEGER I,U,n
U=1
read(*,*)n
DO I=1,n
WRITE(*,*)U
U=2*U+3
ENDDO
END
```

```
PROGRAM suite2
INTEGER I,U,n
U=1
read(*,*)n
DO I=1,n
WRITE(*,*)U
U=U+2
ENDDO
END
```

Exercice 2 :

Ecrire un programme fortran qui lit les valeurs des variables : x, n, a, b puis calcule les sommes et les produits suivants :

Correction

```
PROGRAM SOMME
INTEGER i,j,n,F
real S,X
S=0
read(*,*)n , X
DO i=1,n
F=1
DO j=1,i
F=F*j
ENDDO
S=S+((X**i)/F)
ENDDO
WRITE(*,*)S
END
```

Exercice 3

Ecrire **un** algorithme qui calcul la somme de la série arithmétique de raison 1, suivante :

Som = 0 + 1 + 2 +….1000.

En déduire l'algorithme qui calcule les sommes suivantes :

S1= 0 + 1 + 2 + 3 + 4 + 5 +….N

$s2 = 1 + x^1 - x^3 + x^5 - x^7 + \ldots\ldots \pm x^n$

Correction

```
Algo somme
début
        entier   Som,S1, n,i
        réél     x, S2, S3

        Som ←0
        écrire('calcul de la SOM')
       Pour i←1 à 1000 par pas 1
       Som←Som + i
       Fin pour
        écrire('la somme SOM=',Som)
        écrire('donner n')
        lire(n)
                S1 ←0
       Pour i←1 à n  par pas 1
       S1←S1 + i
       Fin pour
        écrire('la somme S1=',S1)
                écrire('donner x')
        lire(x)
        S2 ←1
        signe←-1
       Pour i←1 à n  par pas 2
       signe←signe * (-1)
       S2←S2 +signe* x**i
       Fin pour
        écrire('la somme S2=',S2)

Fin
```

14. Les tableaux

Pourquoi les tableaux ?

Imaginons que l'on veuille calculer la moyenne des notes des étudiants, mais en gardant en mémoire toutes les notes des étudiants. Il faudrait alors déclarer toutes les variables qu'il y a d'étudiants. Le problème est que si on en a 10, il faut déclarer 10 variables.

Si on en a n, il faut déclarer n variables . . . ce n'est pas réaliste.

Il faudrait pouvoir par l'intermédiaire d'une seule variable.

. . . c'est le rôle des tableaux

14.1. Définition

Un tableau est un conteneur pouvant inclure un nombre fixe d'éléments, qui doivent être du même type. Dans le langage de programmation, un tableau peut être défini comme une structure de données qui regroupe plusieurs entités du même type. Si nécessaire, les tableaux peuvent avoir plusieurs dimensions, ce qui facilite l'organisation du programme par le programmeur.

Un exemple commun, l'utilisation de tableaux multidimensionnels est la mise en œuvre d'opérations matricielles, ou algorithmes qui utilisent des matrices.

Voici les termes importants pour comprendre le concept de tableau.

***Elément* -** Chaque élément stocké dans un tableau est appelé un élément.

Indice - Chaque emplacement d'un élément dans un tableau a un indice numérique, qui est utilisé pour identifier l'élément.

Un tableau à une dimension est parfois appelé vecteur. Il peut être représenté sous la forme suivante :

L(1)	L(2)	L(3)	…….	L(N)

- Dimension du tableau : 1
- Taille du tableau : n
- Les L(i), i=1, 2, …, n doivent être de même type

14.2. Représentation de tableau

Les tableaux peuvent être déclarés de différentes manières dans différentes langues. Tous les éléments du tableau sont de même type.

Syntaxe :

type var (*m1*, *m2*, ...)

m1, *m2*,... déterminent la taille du tableau. Elles doivent être des *des constantes entières*. Il est interdit de mettre des variables entières. Autrement dit : la taille d'un tableau FORTRAN est fixée une fois pour toutes.

Le type défini par un tableau est fonction :

- nombre d'éléments maximal que peut contenir le tableau
- type des éléments que peut contenir le tableau

Les instructions **REAL** et **INTEGER** définissent explicitement le type de variable de tableau. Tous les éléments du tableau sont de même type.

Si le nom du tableau apparait dans un ordre **DIMENSION**, et la première lettre lève toute confusion, si cette lettre est I, j, K, L, M ou n le tableau est entier, sinon réel.

Example:

Real v (100)

Double precision A (100, 50)

Integer C (20)

INTEGER X, Y (0:3)

V est un tableau réel à 1 dimension, de 100 éléments.

A est un tableau réel à 2 dimensions, de 100 lignes et 50 colonnes.

C est un tableau entier à 1 dimension de 20 éléments.

DIMENSION Z (7, 3),

Z est un tableau réel à 2 dimensions, de 7 lignes et 3 colonnes.

La déclaration d'un tableau s'effectue grâce à l'attribut DIMENSION qui indique le profil du tableau, mais aussi éventuellement les bornes, séparées par le symbole « : ».

14.3. Terminologie des tableaux

–Le nombre de dimensions d'un tableau représente **le rang**:

– L'étendue d'un tableau selon une de ses dimensions : **nombre d'éléments dans cette dimension.**

– bornes d'un tableau selon une de ses dimensions : limites inférieure et supérieure des indices dans cette dimension. **La borne inférieure par défaut vaut 1**.
– deux tableaux sont dits conformant (conformables) s'ils ont le même profil

Exemples :

REAL, DIMENSION X(15)

REAL, DIMENSION Y (1:5,1:3)

REAL, DIMENSION Z (-1:3,0:2)

Le tableau X est de rang 1, Y et Z sont de rang 2.

L'étendue de X est 15, Y et Z ont une étendue de 5 et 3.

Le profil de X est le vecteur (/ 15 /), celui de Y et Z est le vecteur (/ 5,3 /).

La taille des tableaux X, Y et Z est 15.

Les tableaux Y et Z sont conformants.

14.3. Manipulation d'un tableau

Les trois manipulations de base sont l'affectation, la lecture et l'écriture.

a- L'affectation :

Pour affecter une valeur à un élément i d'un tableau nommé par exemple A, on écrira :

A(i) ← valeur.

Exemple: A(0) ← 20 ;

affecte au premier élément du tableau A la valeur 20.

b- La lecture :

La lecture des éléments du tableau est réalisée lors de l'exécution.

Exemple : Afin de remplir le tableau V de 7 valeurs numériques, on utilise la boucle Pour.

```
Pour i←1 à 7 Faire
Lire (v (i))
Fin Pour
```

c- L'écriture :

De façon analogue à la lecture, l'écriture de la valeur d'un élément donné d'un tableau s'écrira comme suit : écrire A(i) Cette instruction permet d'afficher la valeur de l'élément i du tableau A.

```
Pour i←1 à n Faire
  Ecrire (V(i))
Fin Pour
```

Exemple :
Ecrire une programme qui recherche dans un vecteur V de dimension N, l'indice de la composante contenant la valeur X. Considérer d'abord le cas où l'on est sûr que X se trouve dans le vecteur, puis le cas général.
Solution :
Algorithme :

```
Algorithme Indice
Tableau V(N) : réel
Variable I, N, Indice : entier
Début
Lire (N)
Pour I = 1 à N
Si (V(I)=X) alors
Indice = I
Fin Si
Fin Pour
Ecrire (l'indice de la composante contenant la valeur X est), Indice
Fin
```

Programme Fortran:

```
Program Indice
Dimension V (N)
INTEGER I, N, Indice
Read (*,*) N
Do I=1, N
IF (V (I).EQ.I) then
Indice = I
ENDIF
ENDDO
write(*,*) 'l''indice de la composante contenant la valeur X est', Indice
END
```

14.5 Tri d'un tableau

Réaliser un tri (ou un classement) est une opération relativement courante dans la vie quotidienne : trier les cartes d'un jeu, trier par ordre alphabétique les livres d'une bibliothèque, classer des concurrents selon leurs performances, etc...

Gérer un agenda téléphonique est aussi une forme de tri, puisqu'un ordre (alphabétique selon le nom des individus) est appliqué, même si ce tri est fait petit à petit, au fur et à mesure de l'insertion de nouveaux numéros.

Trier un tableau c'est obtenir, à partir d'un tableau, un tableau contenant les mêmes éléments mais rangés par ordre croissant. Du point de vue du traitement des données, cette définition

n'est pas suffisante : doit-on construire un nouveau tableau et laisser le tableau t inchangé ? ou bien doit-on transformer le tableau de sorte qu'il soit trié ? Le problème du tri d'un tableau peut donc se décliner sous ces deux aspects :

1. à partir d'un tableau t, construire un nouveau tableau t 0 trié de même longueur que t, contenant les mêmes éléments que t;

2. transformer un tableau t en un tableau trié, sans globalement changer son contenu, mais en déplaçant les éléments au sein du tableau.

De nombreux algorithmes de tri ont été conçus. Parmi eux, on distingue

- les tris comparatifs qui opèrent par comparaison d'éléments du tableau (tri par sélection, tri par insertion, ...),
- d'autres tris qui opèrent par d'autres moyens (comme le tri par dénombrement par exemple).

On présente quelques algorithmes utiles, qui permettent d'ordonner les éléments d'un tableau dans un ordre croissant ou décroissant. L'ordre est par défaut croissant.
Un vecteur est dit trié si V(i) <=V (i+1), □ i □[1..n-1].

Il existe plusieurs stratégies possibles pour trier les éléments d'un tableau ; nous en verrons deux : le tri par sélection, et le tri à bulles. Champagne !

14.5.1 Tri par sélection

Le principe de l'algorithme de tri par sélection consiste à construire petit à petit une tranche triée grandissante du tableau en :

- sélectionnant à chaque étape le plus petit élément de la partie non triée
- échangeant avec l'élément du début de la tranche non triée. La technique du tri par sélection est la suivante :

On met en bonne position l'élément le plus petit. Puis en met en bonne position l'élément suivant. Et ainsi de suite jusqu'au dernier.

Par exemple, si on a un tableau de 10 éléments et on veut que le trier dans l'ordre croissant.

45	122	12	3	21	78	64	53	89	28

On commence par rechercher, parmi les 10 valeurs, quel est le plus petit élément, et où il se trouve. On l'identifie en quatrième position (c'est le nombre 3), et on l'échange alors avec le premier élément (le nombre 45). Le tableau devient ainsi :

3	122	12	45	21	78	64	53	89	28

On recommence à chercher le plus petit élément, mais cette fois, seulement à partir du deuxième (puisque le premier est maintenant correct, on n'y touche plus). On le trouve en troisième position (c'est le nombre 12). On échange donc le deuxième avec le troisième :

3	12	122	45	21	78	64	53	89	28

On recommence à chercher le plus petit élément à partir du troisième (puisque les deux premiers sont maintenant bien placés), et on le place correctement, en l'échangeant, ce qui donnera in fine :

3	12	21	45	122	78	64	53	89	28

On répète l'opération jusqu'à l'avant dernier.

Donc on peut écrire le processus de la manière suivante :

```
Algo tri
Début
Entier i, Posmini, j
Réel temp
Dimension t(100)
Pour i=0, 10
Lire t(i)
Fin Pour
Pour i=0, 10
posmini = i
Pour j=i + 1, 11
Si (t(j) < t(posmini)) Alors
Posmini= j
Fin si
Fin Pour
Temp=t(posmini)
t(posmini)=t(i)
t(i)=temp
Fin Pour
Fin
```

14.5.2 Tri à bulles

L'idée de départ du tri à bulles consiste à se dire qu'un tableau trié en ordre croissant, est un tableau dans lequel tout élément est plus petit que celui qui le suit. En effet, prenons chaque élément d'un tableau, et comparons-le avec l'élément qui le suit. Si l'ordre n'est pas bon, on permute ces deux éléments. Et on recommence jusqu'à ce que l'on n'ait plus aucune permutation à effectuer. Les éléments les plus grands remontent ainsi peu à peu vers les dernières places. En fait, tout ce qu'on peut dire, c'est qu'on devra effectuer le tri jusqu'à ce qu'il n'y ait plus d'éléments qui soient mal classés.

```
Algo Classement
Début
Variable logique Yapermute
Dimension t(100)
Pour i=0, 10
Lire t(i)
Fin Pour
Yapermut=Vrai
TantQue (Yapermut) alors
Yapermut=Faux
Pour i=0, 10
Si (t(i) > t (i+1)) alors
Temp=t(i)
t(i)=t(i+1)
t(i+1)=temp
Yapermut=Vrai
Finsi
Fin Pour
FinTantQue
Fin
```

Travaux Dirigés 5 - Les tableaux

Exercice 1 :
Que produit l'algorithme suivant ?
Algo Carre
Début
Tableau Nb(5) en Entier
Variable i en Entier
Pour i = 0, 5
Nb(i) ← i * i
Fin Pour
Pour i = 0, 5
Ecrire Nb(i)
Fin Pour
Fin
Peut-on simplifier cet algorithme avec le même résultat ?

Correction
Cet algorithme remplit un tableau avec le carré de six valeurs : 0, 1, 2, 3, 4, 5.
Il les écrit ensuite a l'écran.
Simplification :
Algo Carre
Début
Tableau Nb(5) en Entier
Variable i en Entier
Pour i = 0, 5
Nb(i) ← i * i
Ecrire Nb(i)
Fin Pour
Fi

Exercice 2 :
Écrivez un algorithme permettant, à l'utilisateur de saisir les notes d'une classe. Le programme, une fois la saisie terminée, renvoie le nombre de ces notes supérieures à la moyenne de la classe.

Correction
Algo classe
Début
Tableau T(Nb)
Variable i, Nb, Nbsup en Entier
Variable Som, Moy en réel

```
Ecrire “Entrez le nombre de notes à saisir : “
Lire (Nb)
Pour i = 1, Nb
Ecrire “Entrez le nombre numéro”, i
Lire T(i)
Fin Pour
Som = 0
Pour i = 1, Nb
Som = Som + T(i)
Fin Pour
Moy =Som / Nb
NbSup =0
Pour i =1, Nb
Si (T(i) > Moy) Alors
NbSup =NbSup + 1
FinSi
Fin Pour
Ecrire NbSup, “élèves dépassent la moyenne de la classe”
Fin
```

Exercice 3:

Que font les algorithmes suivants ?

a)

```
Var i, n (10) entier
Debut
n [0] ← 1
Pour i allant de 1 a 9
Faire
N(i) ← n (i-1) + 2
Fin
```

b)

```
Var i, n [10] entier
Début
Pour i allant de 0 a 9
Faire
N(i) ← 2 * i
Fin
```

Correction

a) Initialisation d’un tableau avec les 10 premiers nombres impairs

b) Initialisation d’un tableau avec les 10 premiers nombres pairs

Exercice 4 :

1- Ecrire un programme fortran qui effectue la lecture d'une matrice carrée A ainsi que sa taille **n** et affiche la trace de A (pour une matrice $A(a_{i,j})$, $Trace(A)=\Sigma a_{i,i}$ la somme des éléments sur la diagonale).

2- Ecrire un programme fortran qui effectue la lecture d'une matrice carrée A ainsi que sa taille **n** et affiche la matrice transposé tA de A (Pour une matrice $A(a_{i,j})$, $tA(a_{j,i})$).

Correction

1-

```
PROGRAM matrice
INTEGER I,J,n
DIMENSION A(100,100)
WRITE(*,*)'Donner la taille de la matrice A est n='
read(*,*) n
TRACE_A=0
WRITE(*,*)'Donner les éléments de la matrice A '
DO I=1,n
DO J=1,n
read(*,*) A(i,j)
ENDDO
ENDDO
DO I=1,n
TRACE_A=TRACE_A+A(i,i)
ENDDO
WRITE(*,*)'La trace de la matrice A est:',TRACE_A
END
```

2-

```
PROGRAM transposé
INTEGER I,J,n
DIMENSION A(100,100),TA(100,100)
WRITE(*,*)'Donner la taille de la matrice A '
read(*,*) n
WRITE(*,*)'Donner les éléments de la matrice A '
DO I=1,n
DO J=1,n
read(*,*) A(i,j)
ENDDO
ENDDO
DO I=1,n
DO J=1,n
TA(i,j)=A(j,i)
WRITE(*,*)TA(i,j)
ENDDO
ENDDO
END
```

Références bibliographiques

1) Architecture et technologie des ordinateurs 4eme édition, P. ZANELLA, DUNOS 1991
2) Initiation à la programmation en Fortran. J.B KRIN, Ecole national supérieure de technologie avancé, 1975
3) Eléments d'informatique et programmation Fortran, M BELLEHULEUR, LIDEC, 1974
4) L'enseignement du langage algorithmique Fortran 77, L. BAGHDALI, A996
5) Programming with Fortran , S. LIPSCHUTZ, SHAUM outline series, 1978
6) Introduction à l'algorithmique, T. CARMEN, DUNOD, 1994
7) Analyse et mise en œuvre arithmétique des ordinateurs, M. LAPORTE, E. TECHNIP, 1974
8) Exercices et problèmes d'algorithmique, B. BAYNAT, DUNOD, 1991
9) Exercice de programmation en Fortran, J. P. LAMOITIER, DUNOD, 1977
10) Support de cours en Informatique o2, Algorithmique et programmation, T. Nateche, USTO 2017 ;
11) Cours de fortran 90, enseirb-matmeca l. M. LUCINSTITUT, université de bordeaux, 2011 ;
12) Structures conditionnelles [if], K. ZAMPIERI, Unisciel Algoprog, 2018 ;
13) Algorithmes D.E Knuth CSLI Publications 2011 ;
14) Introductipon a la science informatique G. Dowek Ed RPA 2010 ;
15) Eléments pour une histoire de l'informatique, D.E Knuth CSLI Publications 2011 ;
16) Cours et exercices corrigés d'algorithmique- J. Julliand Ed Vuibert Fev 2010 ;
17) Algorthmique méthodes et modèles , P Lignelet Ed Masson 1988 ;
18) Cours algorithme Cécile Balkanski, Nelly Bensimon, Gérard Ligozat IUT Orsay ;
19) https://www.math.sciences.univ-nantes.fr/~saad/coursf90/node34.html
20) http://docplayer.fr/69887014-Cours-de-fortran-90-licence-3-e-annee-ingenierie-mathematique-luc-mieussens-universite-paul-sabatier-toulouse-3.html