Département d'informatique

Algorithmique & Structures de données 1

Cours & Exercices corrigés

Samira Benbadallah, Maitre assistante, Université Oran1

Hafid Haffaf, Professeur, Université Oran1

Edition Septembre 2023

Sommaire

«*L'ennemi du savoir n'est pas l'ignorance mais l'illusion de la connaissance* »

Stephen Hawking

Avant-propos

Il y a beaucoup d'ouvrages sur l'algorithmique, sur les langages de programmation pour apprendre l'algorithmique, sur l'introduction et l'initiation à l'informatique. Lorsque nous avons vu cet appel de laisser pour les générations montantes et futures d'informaticiens un ouvrage de référence sur l'algorithmique et les structures de données, on s'est dit « it's a big challenge ! ».

Finalement au fil de l'écriture, nous nous sommes rendu compte, qu'il fallait transmettre non seulement un savoir mais aussi un savoir-faire. On dit souvent que la programmation est un art, et en ce sens, l'artiste qu'est le programmeur doit savoir comment va s'exécuter son programme avant d'écouter sa machine entrain de la faire. Cette empathie nouvelle qui lie désormais un étudiant en informatique à un PC qu'il va devoir dompter, se forge avec l'apprentissage pratique. Un programme (assez important) ne marche jamais du premier coup. On apprend une astuce, on change le programme, et hop on voit ce que ça donne, et si vous êtes mordus d'algo –c'est le terme que vous utiliserez en parlant de nous- vous allez être très content quand ça marche. Et au fil du temps, on construit sa propre personnalité de programmeur. On construit de plus en plus court, de plus en plus rapide, et on est dans les fondements de l'optimisation.

Peut-on penser programme sans penser données ? C'est le fameux dilemme pour concevoir un programme. Commence-t-on par définir un cercle ou dessiner un cercle pour comprendre sa définition ? La philosophie du langage C (ce dilemme a été résolu grâce aux langages objets, tel le C++) comme référence de quelqu'un qui doit aller plus loin en programmation est basée sur la vision globale du projet, et sa décomposition en sous programmes ; ce qui permettra à l'étudiant de commencer avec des petits programmes, puis de construire de plus grands.

Nous avons insisté, en suivant le programme de première année auquel est destiné ce manuel, sur certains points avec des remarques et des exemples qui expliquent pourquoi c'est comme ça et pourquoi pas autrement. La première étape est d'abord le choix des meilleures structures de données à utiliser, puis écrire les algorithmes par les structures de contrôle adéquates.

Chaque partie est étayée avec des exercices corrigés et non corrigés. Apprendre à programmer nécessite une machine afin de voir rapidement si l'implémentation de l'algorithme donne le résultat voulu.

C’est aussi le meilleur moyen, par essai-erreur et la pratique, de localiser les erreurs de syntaxe et les erreurs sémantiques. Les listings des programmes écrits en C fournis ont été testés.

L’apprentissage des structures de données en C permet de comprendre en détail la nature des objets manipulés à travers les librairies Java ou la STL du C++. Nous avons essayé, comme ‘Gallilée’ (ouvrage intitulé « l’essayeur » dans les années 1600), de mettre noir sur blanc un ensemble d’idées qui constituent la base de l’algorithmique, c’est-à-dire la base de votre personnalité de programmeur. A la fin de la matière, nous espérons avoir fait plus qu’essayer.

P.S : Nous tenons à remercier Mme Amrane B., maitre de conférences au département informatique d’Oran 1 pour sa relecture du document.

1 Introduction générale

1.1 Bref historique de l'informatique

L'informatique est la science du traitement automatique de l'information. Autrement dit, appliquer des traitements sur des informations avec des moyens automatiques pour obtenir des résultats. Cette science applicative est utilisée partout et dans tous les domaines. A l'ère du numérique et des technologies de l'information et des communications, les applications mobiles ou les objets autonomes ou robotisés, dits connectés, font appel aux algorithmes. Cela fait plusieurs millénaires que l'on résout des **algorithmes** à la main. Au quotidien dans la vie moderne, chacun de nous utilise un algorithme pour mener une vie satisfaisante en dépensant la moindre énergie.

Le même algorithme peut être vu sous différentes formes, Celui d'Euclide du calcul du PGCD a une version itérative et une version récursive.

L'information est une séquence de données relatives à un objet informatique, elle représente un élément de connaissances pouvant être transmis sous différentes formes. Elle est vendue très chère et constitue une source d'enrichissements de ses grands manipulateurs. Les systèmes d'information gèrent actuellement tous les processus qui nécessitent une automatisation de certaines tâches.

L'ordinateur (angl. *Computer*) (parfois sous forme de dispositif spécial : automate programmable, …) est un appareil informatique dédié au traitement automatique de l'information. Il est donc le moyen de mettre en œuvre nos algorithmes, il est en particulier capable de : Acquérir des informations, Conserver des informations, Effectuer des traitements sur des informations, et enfin Restituer des informations.

Aujourd'hui, l'ordinateur est capable d'apprendre et donc ne traite pas uniquement de l'information mais aussi des connaissances. Même en limitant à un domaine technique comme les sciences de données, on peut trouver toute sorte d'algorithmes de quoi remplir plusieurs ouvrages. Même celui de D.Knuth, *the art of computer programming* (Adisson Wesley), en totalisant 3168 pages ne fait pas le tour du sujet.

On dit que La Pascaline 1642 est le premier ordinateur à faire des calculs. Bien avant, le boulier était déjà utilisé par les grecs et les égyptiens. « L'Eniac » en 1946, est le premier ordinateur pouvant être programmé et qui sera à l'origine plus tard, de l'architecture de Von Newman IBM en 1959.

L'*Eniac* occupait la surface d'un terrain de football, car construit avec des anneaux pour les calculs et les tubes à vide qui n'a rien à voir avec l'électronique d'aujourd'hui. Il ne sera utilisé d'ailleurs à peine une dizaine d'année. Le mathématicien anglais pensait déjà au fondement de l'informatique en présentant en 1936, un calculateur universel (en fait une machine graphique) capable de résoudre toute fonction récursive, c'est la **machine de Turing**. A la fin des années 50, les transistors ont révolutionné l'électronique de puissance, ce qui a permis de réduire considérablement la taille des ordinateurs. Ils seront suivis plus tard (1963-1972) par les circuits intégrés (**3ième génération**) qui continuent jusqu'à présent leur miniaturisation vers les nano-technologies.
Les années 1970-1980 ont vu la **4ième génération** basée sur les micro-processeurs notamment par les firmes Intel et Motorola afin de rendre encore accessible les ordinateurs au grand public. Selon la loi de « **Moore** » au début des années 70, prévoyait une évolution qui ferait doubler les performances des ordinateurs tous les dix-huit 18 mois.
En 1987, les supercalculateurs et réseaux ont fait leur apparition, profitant de l'avancée de l'électronique d'une part et des moyens de télécommunication d'autre part. « Network is computing » est le slogan de Sun micro-systems et Java. Pendant ces années, c'était l'euphorie de l'informatique qui allait connaitre un développement fulgurant avec l'apparition des premiers ordinateurs personnels, ainsi que l'émergence des grands éditeurs tel que Microsoft qui a fait de Windows le système d'exploitation de référence.
'Dr Watson', la machine d'IBM constituée de 90 serveurs (gros ordinateurs) tourne à 80 téraflops, c'est une machine qui a pu diagnostiquer une maladie rare.
Le boom réel de l'informatique est venu en 1990 lorsque le web et les TICs (Teem bernes Lee), ont installé une vision système d'information, grâce également aux bases de données, où le partage et la navigation à travers les pages html est devenu le moyen le plus populaire pour réaliser toutes les tâches quotidiennes : e-mails, e-commerce, e-gouvernance, **facebook**, **Twitter**, ….etc. Au passage, une enquête a révélé que 60 % du temps des utilisateurs Internet est consacré aux réseaux sociaux, à travers bien sûr les téléphones mobiles qui sont devenus plus que de simples téléphones.
Google traite plus de 90000 requêtes par seconde. En 2010, **Le cloud**, l'internet des objets, la réalité virtuelle ont fait progresser la mise en place d'une numérisation accentuée. La vitesse avec laquelle nous sombrons de jour en jour dans cette virtualité dépasse celle qui nous permet d'analyser d'abord de quoi sont faits ces objets si loquaces mis en boite et connectés au grand réseau qu'est

Internet aujourd'hui. Il y a déjà 10 milliards d'objets connectés au net, donc déjà plus que la population mondiale, ce qui prouve que nous ne sommes pas les seuls « intelligents » à user du web. L'immensité des données qui y sont stockées chaque jour, encouragée par le développement technologique des serveurs et la miniaturisation des espaces de stockage, font de nous les esclaves de ce monde virtuel.

Aujourd'hui, c'est l'intelligence artificielle qui domine les débats de l'informatique de demain et qui envahit de plus en plus notre mode de vie. Déjà en 2016 l'Alpha GO (le jeu de Google) avec ses 30 couches de R.N.A. (réseaux de Neurones) est imbattable par les plus intelligents des champions. Aujourd'hui (en 2023), les systèmes conversationnels (**CHATGpt**) ont tendance à se substituer aux humains pour résoudre leur problème (y compris celui de concevoir des algorithmes) faisant ainsi peur pour l'avenir de la créativité. En 2020, les grands du numérique mondial plus connus sous le groupe de la **GAFEM** (Google, Amazone, facebook, et Microsoft) qui détiennent 92% des données mondiales ont lancé deux grands projets de l'I.A Google 'Brain', et 'Fair' de faceBook afin de rendre nos machines capables de mieux nous comprendre et surtout anticiper nos actions. Le danger est d'ordre éthique, car avec le développement de la réalité virtuelle, '**Metaverse**' qui est le futur internet ne nous permettra d'accéder à la réalité qu'à travers cette augmentation.

Pour *fonctionner*, un ordinateur a besoin d'être équipé d'une carte mère, d'un microprocesseur, d'une mémoire centrale, de périphériques d'entrée, périphériques de sortie et de périphériques d'entrée /sortie. Puisque l'ordinateur est une machine électronique basée sur les transistors et les circuits imprimés, le code de base de la machine est binaire (basé sur 0 et 1).

L'information élémentaire est par conséquent **le bit (binary digit)**. C'est pour cette raison que les unités de stockage sont des puissances de 2 pour représenter l'aspect combinatoire de ce stockage. Avant d'arriver au code binaire, notre programme informatique doit subir une batterie de transformations qui passent par le compilateur puis par le système d'exploitation.

Tous ces objets connectés, ordinateurs, serveur, automates ou autre, il faut les programmer pour les faire fonctionner. Nous avons besoins donc de logiciel et d'applications dédiés (qui coûtent parfois plus chers que le matériel). C'est ici qu'intervient le rôle de l'ingénieur ou du programmeur.

Cet ouvrage se base sur le langage C, mais il est conseillé d'apprendre d'autres langages pour être polyvalent.

Aujourd'hui, Python semble être un langage à tout faire, créer et gérer des sites avec bases de données, développer des applications aussi bien pour desktop que pour des mobiles, automatiser des scripts et même de l'apprentissage automatique grâce aux différentes bibliothèques.

La Mémoire Centrale ou Mémoire vive (RAM : Ramdom Access Memory) est une mémoire qu'on peut à tout moment effacer. Rappelons que la mémoire se mesure en octets ou puissances de 2 d'octets : méga, giga, téra, peta et zeta pour respectivement 10^6 10^9 10^{12} et 10^{18}
Enfin le développement technologique et la bio-technologie vont révolutionner, par l'apparition de l'ordinateur quantique (dont l'information élémentaire n'est plus le bit mais le Qbit) et l'ordinateur à base d'ADN, notre manière de concevoir des algorithmes déterministes. Notre cerveau lui, conditionné pourra-t-il s'adapter à cette évolution ?

1.2 Objectifs de l'ouvrage

L'ouvrage a pour but de donner l'essentiel des notions d'algorithmique, en orientant le lecteur plutôt vers une programmation en langage C (la plupart des exemples). Après une brève introduction à l'algorithmique (section suivante), nous entamerons à travers le chapitre 2, la notion de séquence où le respect de l'ordre des instructions est important. Nous abordons par la suite la notion de variable comme élément de base d'une structure de donnée. En effet, le programmeur doit concevoir non seulement la séquence d'instruction qui résout un problème mais aussi sur quelles données, cette séquence va s'éxecuter. Les chapitres suivant aborderont les structures de contrôle classiques : les branchements conditionnels et les boucles (chapitre 4). On mettra l'accent sur les types de boucles ainsi que leur mode d'utilisation. Les premières structures de données complexes apparaitront au chapitre cinq par les tableaux et chaines de caractères, où il faudra regrouper les données dans une ou deux dimensions. Nous finirons l'ouvrage par la présentation des enregistrements ou structures personnalisées. Dans ce chapitre, l'étudiant est amené à concevoir des petites « architectures » de données hiérarchisées, pour lui donner la possibilité d'aller plus loin. A la fin de l'ouvrage, on doit pouvoir analyser un problème (son aspect simpliste) et écrire un algorithme (ou un programme C) afin de le résoudre.

1.3 Introduction à l'algorithmique ?

Pour se déplacer dans un espace donné, pour installer un logiciel, ou tout simplement la suite des actions que l'on exécute chaque matin sont des exemples d'algorithme. Un algorithme est une suite d'instructions pour résoudre un problème ou faire un calcul. Ce problème est destiné à être traité par un programme informatique. Par conséquent l'algorithmique est la base et le fondement de l'informatique dans son aspect software (ou logiciel). L'algorithme est alors vu comme l'ensemble des méthodes qui permettent d'étudier ces algorithmes. Depuis la pascaline au 16^{ième} siècle, la transformation des problèmes et sa décomposition en une série d'instructions n'a cessé d'évoluer.
Un **programme informatique** : lit des données en entrée, effectue des calculs à partir des données en entrée et en dernier affiche des résultats, pour cela le programmeur doit suivre une **méthodologie** pour expliquer à l'ordinateur dans un langage de programmation le problème à résoudre. Cette résolution d'un problème par ordinateur peut être schématisée en 6 étapes *figure1.1*.
Le premier programme qui s'exécute quand on démarre une machine est son **système d'exploitation (windows, Linux**,..) ; il a pour rôle de faire fonctionner le matériel et les périphériques (hardware) et offrir des interfaces homme/machine à l'utilisateur. Le **virus** informatique est aussi un programme, il a été écrit délibérément pour nuire à la machine ou à l'environnement. Un algorithme doit vérifier certaines propriétés mathématiques et logiques.

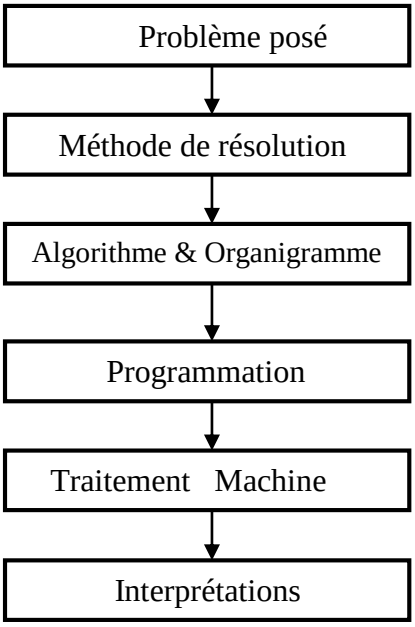


Fig 1.1 étapes de construction du programme

En fait dans ce schéma, la résolution d'un problème peut faire appel à plusieurs algorithmes, et même après à plusieurs langages. Le rôle du développeur est de décomposer son problème en une suite de programmes ou unités (qu'il peut compiler séparément) qui traduiront les étapes de cette résolution.

Les langages évolués aujourd'hui facilitent la tâche du programmeur pour le laisser se concentrer davantage sur la conception. L'algorithmique comme introduit par ***el khawarizmi***, mathématicien arabe du 9ième siècle, est donc principalement la théorie de la résolution ou de t'automatisation du processus de résolution, cela représente l'ensemble des fonctions calculables. Une fonction est non calculable si donc il n'existe aucun algorithme qui génère le résultat de cette fonction pour une entrée donnée.

Vous pouvez utiliser n'importe quel langage de programmation pour réaliser un algorithme. Les plus en vogue aujourd'hui sont Python, R (gratuits sur le net) pour la simplicité, la gestion des données (data mining notamment avec R) et la capacité de s'adapter au web et applications mobiles, sans parler des plateformes de développement web (django, ..) web ou autre.

Un langage comme le C (écrit par D.Richie dans les années 1970) est dit compilé (comme d'autres langages comme le pascal), i.e nécessite l'installation d'un compilateur pour pouvoir s'exécuter. 80% du compilateur C est écrit en …C. Nous reviendrons dans le tome 2 sur cette notion de récursivité. En outre, une erreur de syntaxe ne permet pas d'aller plus loin. Un langage de programmation comme un langage naturel possède son propre alphabet (sa syntaxe), et sa grammaire, i.e comment écrire un programme compréhensible par la machine.

D'autres sont dits interprétés (tels que le Basic, javascript ou les navigateurs web). Avec le développement Web justement, les outils de développement s'enrichissent (à l'image de flutter par exemple) et se spécialisent dans ce qu'on appelle développement web. Comme cela évolue très vite, surtout avec l'avènement de l'intelligence artificielle, nous ne pouvons donner une référence pour le téléchargement.

Les interfaces visuelles offertes permettent de glisser déposer prêtes à l'emploi sans écrire du code. Il en est ainsi des langages dits orientés objets où les objets informatiques incluent leur propre code qui les gère. Imaginer un objet horloge, il doit contenir toutes les fonctions d'affichage et de manipulation du temps.

Théoriquement, tout algorithme correspond à la machine universelle de Turing (mathématicien anglais des années 40 qui a été à l'origine du premier algorithme de décodage qui a permis de décrypter les messages radio allemands),

et peut être écrit dans n'importe quel langage. C'est un automate sous forme d'un ruban infini, qui peut écrire 0 ou 1, lire une donnée (un caractère) et se déplacer à droite ou à gauche.

Le code informatique est lui-même une pensée traduite dans une langue, mais on ne peut décrire une langue par elle-même (théorème de Godel). Il existe des systèmes de réécriture qui permettent de traduire un code dans un autre (à l'image d'un translateur de langues). Nommé TransCoder, il permet de traduire du Java ou C++ en python et vice versa. La première préoccupation était celle de traduire du Cobol (langage des anciens systèmes bancaires) vers du java (pour un cout de 750milions de dollars)

Certains langages ont une orientation particulière (comme SQL: Structured Query Langage) pour la manipulation des données (ou bases de données).

La notion même d'algorithme a changé. Ce déterminisme qui le caractérisait n'est plus une condition indispensable ; il est maintenant bio-inspiré, heuristique ou probabiliste, laissant à la machine le choix de meilleure façon de résoudre un problème, encore faut-il lui apprendre.

Aujourd'hui, informatique et algorithmes sont indispensables à toutes les sciences et tous les domaines, et s'en passer signifierait vivre dans une ère préhistorique.

Dans cet ouvrage, un algorithme sera écrit en langage algorithmique (et parfois en C) qui est aussi appelé un **pseudo-code**, afin de laisser le choix au programmeur du langage d'implémentation sur machine.

2 Algorithme Séquentiel Simple

2.1 Langage et langage algorithmique

L'algorithme est une suite d'actions appelées ***instructions*** dont l'exécution fournit le résultat recherché, chaque action étant formée d'un certain nombre d'opérations, un algorithme doit toujours se terminer. C'est aussi la description des étapes à suivre pour résoudre un problème.

Tous les jours, en vous réveillant le matin, vous exécutez machinalement des algorithmes pour vous préparer, pour aller au travail, pour préparer un plat de cuisine et pour planifier vos tâches quotidiennes. En voici quelques exemples

- Conduire une voiture (GPS), suivre un chemin, venir à l'université, installer un programme, acheter un produit, démarrer une machine ou un processus.
- Diagnostiquer un système (ou un patient).
- Résoudre une équation ou un système d'équations, le premier algorithme pour ce faire a été celui d'Al Khawarizmi au 9 ième siècle à Baghdad, dans son ouvrage « El Djabr oua el Moukabala ». C'est d'ailleurs le premier à avoir exprimé cette notion d'algorithme comme un ensemble d'étapes pour résoudre un problème. L'équation du second degré en est un bon exemple pour comprendre le 'si …alors …sinon' en testant le signe du Delta.
- Répondre à une requête : Algorithme de **ranking** de Google.
- Faire une expérience chimique ou physique : les étapes à suivre : pratiquer un geste chirurgical (ensemble des actes au bloc chirurgical).
- Appeler au téléphone, lire (et répondre à) ses mails.
- Crypter des données, reconnaitre une forme…..
- Jouer aux dames, jouer une partition de musique. Par contre écouter ou regarder ne relève pas d'action algorithmique.
- Programmer en Intelligence artificielle (démontrer un théorème).
- Algorithmes d'apprentissage, training en bourse.

⚠ Parfois, il y a confusion avec le concept de protocole. En général ce dernier terme est utilisé quand plusieurs acteurs exécutent séparément des actions (protocole de communication).

2.1.1 Exemples pour choisir un algorithme

Exemple 1 : pour trouver un passage dans un livre ?

a) Parcourir le livre du début jusqu'à la fin 1, 2,3 … jusqu'à trouver.
b) Lire au hasard 12, 24, 35,7, …jusqu'à trouver le texte.
c) Aller à la table des matières ou index pour rechercher par mot clé.

Exemple 2 : rechercher un lieu dans une ville Algorithme de recherche

a) Se déplacer et demander aux gens.
b) Aller à l'agence de tourisme la plus proche et demander un plan.
c) Utiliser le GPS.

Exemple 3 : Classer une liste de nombres du plus petit au plus grand (liste ordonnée), plusieurs algorithmes de tri existent.

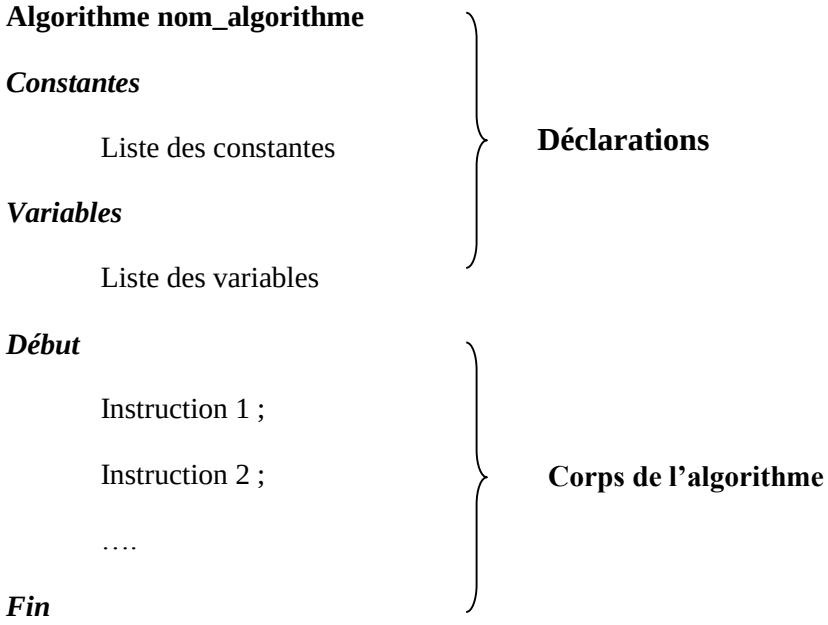
2.1.2 Quelles sont les caractéristiques d'un algorithme ?

- Tout algorithme est une suite d'actions **ordonnées** (l'ordre des instructions est important).
- Le temps d'exécution d'un algorithme est fini (encadré par début, …. Fin).
- Un algorithme ne fait que ce qu'on lui dit de faire (même si on lui dit d'apprendre à le faire).
- Pour les mêmes données, l'algorithme donne toujours le même résultat (déterminisme). Par exemple les solutions d'une équation du second degré si les coefficients a, b et c sont connus. Il existe cependant des algorithmes dits probabilistes ou indéterministes basés sur des lois aléatoires.
- Le cheminement de l'exécution dépend de la nature et la taille des données (exp si interrupteur éteint, faire une suite d'opérations, sinon autres opérations).
- L'algorithme doit être structuré et clair (on doit pouvoir suivre son exécution).
- Un algorithme est meilleur s'il est plus **court** (taille).
- Un algorithme est meilleur s'il est plus **rapide.**
- Un algorithme est meilleur s'il nécessite **moins de place mémoire.**
- ***Ces trois dernières propriétés*** sont relatives à l'optimisation de l'algorithme. Nous verrons dans cet ouvrage, des cas où on pourra choisir entre plusieurs algorithmes, selon l'un des critères ci-dessus.
- Tout algorithme peut être traduit en **langage de programmation (thèse de turing –church)**

2.2 Parties d'un algorithme

Un algorithme est composé de **2** blocs :

- Un **environnement** de résolution du problème qui rassemble les déclarations des objets (Déclarations).
- Le **corps** de l'algorithme qui décrit les traitements à faire.



Remarquer d'abord que le ' ;' sépare les instructions (comme en lange C ou Pascal, par contre en python, un simple retour de ligne les sépare). Tous les langages de programmation n'ont pas forcément cette structure. C'est pour cela que nous distinguons entre langage algorithmique et langage de programmation. D'abord nous décrivons la résolution du problème en langage algorithmique avant de la transcrire ou de le « coder » dans un langage donnée. Les langages objets par exemple ne séparent pas données et programmes, l'objet contient les propriétés et les méthodes appliquées sur ces données (principe d'encapsulation). Pourquoi déclarer les variables (et les constantes) ? Puisque l'ordinateur réserve de l'espace mémoire à chaque variable déclarée (du moins pour les langages où la déclaration est obligatoire comme le **C**), cette place servira à stocker les valeurs de cette variable. En outre le type va déterminer la taille en octets à réserver. Donc le nom servira à identifier la variable pour la localiser en mémoire et son type l'espace que les valeurs occuperont. Les opérations possibles avec un type ne sont pas possibles avec d'autres. On ne peut diviser des caractères par exemple. Certains langages permettent de changer de type (Transtypage) pour faire des opérations.
Exemple : distinguer le nombre 12 de la chaine de caractères '12'.

2.3 Les données : variable et constante

L'objectif est d'automatiser le traitement des données par l'ordinateur. Qu'est qu'une donnée ?

2.3.1 Donnée

La donnée est issue du processus de numérisation des informations. Nous pouvons citer :

- Des données personnelles (textuelles, nom, adresse, commentaire, biométriques,…..,).
- Des données chiffrées (Prix, bilans –tout type- , couts de production, graphiques, niveaux, reporting, marketing,..).
- Des informations universelles (la date et l'heure, les conditions météo, l'instant de création de l'univers, emploi du temps, la constante de Plank, la contante de la gravitation,….).
- Des informations audio visuelles (photos, scanner, voix IP, vidéos ….).
- Des avis (j'aime sur **facebook**, scores, notations, résultats des élections électroniques.
- L'état d'un système (interrupteur, micro-onde, un stock, facture payée –ou non-) représenté ici par une donnée numérique ou booléenne.
- Les valeurs issues des capteurs physiques d'un système (température, pression, glycémie, humidité, …) qui sont des valeurs analogiques transformés.
- Les coordonnées de géolocalisation (espace, cartographies,..).
- Le code (password, sortie décodeur, Qr-code, signature numérique, Pass Sanitaire,..).
- L'Historique (statistiques, archives, le temps –date heure-….).
- Du contenu numérisé (e-gouvernance, suivi, e-commerce, transactions bancaires, …).
- Des Méta-données (informations sur un site, mots clés, index). Ces dernières expriment des données sur d'autres données ou information.

La donnée actuellement vaut de l'or. Les géants de la donnée (**Gafam : google, facebook, amzone et microsoft**) aspirent nos données personnelles pour les revendre aux enchères aux institutions étatiques et commerciales.

2.3.2 Variable

Une variable est une référence sur une cellule mémoire qui possède une adresse, identifiée par un nom « **identificateur** » et destiné à stocker une valeur pouvant être modifiée durant tout l'algorithme.

***Astuce** : choisir toujours un nom de variable qui a un sens, ne pas utiliser d'accent ou chapeau, utiliser toujours l'inderscore _ au lieu du 'tiret de 6' – qui peut être confondu avec le signe moins.*

Syntaxe algorithmique

<Identificateur> : <type>

En langage C

<type> <identificateur>

l'identificateur : nom que le programmeur donne à une constante, une variable ou à une fonction est une suite de caractères alphanumériques commençant par une lettre et pouvant contenir un trait d'union. Le choix de noms mnémoniques (ayant un sens) est important en programmation pour une bonne lisibilité du code par un autre programmeur.

Le type : il décrit le contenu de la variable ou de l'entité à manipuler. Le type peut être simple dit aussi standard, ou composé.

En langage algorithmique	En C
Variables Rayon, racine, périmètre, surface : **réel ;** Variables Compteur, X, i, N : **entier** ; Variable C : **caractère** ;	**float** Rayon, racine, périmètre, surface; **int** Compteur, X, i, N ; **char** C ;

Exemples :

Le nombre d'octets réservés dépend donc du type de la variable. Pour un float par exemple, on le met sur 4 octets, un caractère dans un octet, ..etc.

les réels sont aussi représentés par le type qui veut dire double précision, qui se traduit également par le double de l'espace réservé pour un float.

2.3.3 Constante

Une constante est une case mémoire elle aussi possédant une adresse mémoire identifiée par un nom « **identificateur** » et contenant une valeur qui ne change pas du début jusqu'à la fin de l'algorithme.

L'écriture langage algorithmique est la suivante :

Constante Nom_Constante = Valeur

Exemples :

En langage algorithmique	En C
Constante PI = 3.141592653589 Constante TVA = 19	#define PI 3.141592653589 #define TVA 19

Remarquer que la constante en informatique n'est pas une constante universelle, comme la gravité ou la constante de **Plank**, mais une manière de laisser pour une certaine durée une valeur fixée à l'avance. L'exemple de la TVA indique que cette valeur est fixée au moins pour une année. L'avantage de la déclarer comme constante dans le programme est de ne pas avoir à changer toutes les valeurs de la TVA à chaque fois que cette valeur change. Il suffit de revenir à la constante et la changer. Tout cela pour dire qu'elle n'est constante que ne le suggère son adjectif.

2.4 Les types de données

Tout langage de programmation offre un certain nombre de ***types standards*** préalablement définis. Les langages sont mêmes dit typés ou fortement typés. Il existe **5 types standards :**

2.4.1 Type logique (booléen) :

Valeur pouvant être soit Vraie, soit Fausse.

Constante true = Vrai
Variable T : booléen ;
Constante Trouve = faux
Les opérations possibles sont :

- Operateurs logiques : la conjonction (et), la disjonction (ou) et la négation (non).
- Operateurs de relation : =, <, >‡…

Exemples :
x<y et z=0 expression comportant une conjonction.
99<100 expression qui fournit vrai.
99=100 expression qui fournit faux.
En langage C, la valeur « **faux** » est représentée par 0 et la valeur « **vrai** » est représentée par 1.

2.4.2 Type entier :

Valeur numérique entière pouvant être signée ou non signée (codée sur un ou plusieurs octets).

Exemples :

Constante g = 10 ;
Variables x,y : entier ;

Les opérations possibles sur les entiers sont :

Les opérations mathématiques : +,-,*, div (division entière ou euclidienne), mod (reste de la division entière).

En C, le type entier peut être représenté par *int* ou *short* (sur 2 octets) ou *long* (sur 4 octets). De plus *'signed'* désigne les entiers relatifs.

2.4.3 Type réel :

Les valeurs numériques du type réel sont codées avec une mantisse (23 bits qui permet une précision) et un exposant pour une variable réelle de type simple.

Constante g =9.8

Variables rayon , surface : réel ;

- Les opérations possibles sur les réels sont :

- Les operateurs: +, -, *,/

- Quelques fonctions prédéfinis sur les entiers/réels en langage algorithmique.

power (x, n) : x à la puissance n ; pour le carré , x * x est plus simple.
abs (n) : valeur absolue d'un entier n.
sqrt (n) : racine carrée d'un nombre.
trunc (n) : la partie entière d'un nombre.
round (x) : donne l'entier le plus proche (arrondi) d'un nombre réel.
sin(x) : sinus de x.
cos(x) : cosinus de x.

- **Expression arithmétique**

L'ordinateur fait le calcul d'une expression dite arithmétique avant de faire l'affectation. Celle ci doit suivre les règles (selon le langage) logiques de manipulation des nombres, les priorités sont données aux parenthèses, puis les opérateurs eux même ont des règles de priorité entre eux. '/' et '*' sont prioritaires par rapport aux '+' et '-'.

En C, nous retrouvons la fonction modulo %, ** est l'opérateur de puissance, si les deux valeurs sont entières, la división / le será également. Inclure la bibliothèque math.h pour les fonctions mathématiques usuelles.

2.4.4 Type caractère :

Ce type comporte :

-Les lettres de l'alphabet : " A, B , …Z, a ,b,…z ".

- Les chiffres 0 à 9.

- Les signes de séparation :, - ; - (-) - : - [-] - = - blanc,….

- Les Caractères spéciaux @,%,$,….
- Les Opérateurs +,*,<,>…

Exemples :

Constante plus = '+'
Constante C = 'a'
Variables C1, C2: caractère ;

En machine chaque caractère est représenté sur un octet donc on peut représenter 2^8 soit 256 caractères différents.

Le code ASCII (Américain Standard Code for Information Interchange) permet de coder 255 caractères en leur associant des entiers.

- Les fonctions prédéfinies sur les caractères sont :

- ord (c) : renvoie un entier correspondant au rang du caractère c dans la table Ascii.
- chr (i) : C'est une fonction qui est l'inverse de ord (c), elle renvoie le caractère correspondant à l'entier i.

Le langage C fait la différence entre majuscule et minuscules.

Au passage, il faut savoir qu'il existe d'autres codage que le code ASCII, *UTF-8 ou unicode* pouvant représenter beaucoup plus de caractères. http://www.unicode.org/charts/PDF/U0100.pdf.

Exemples :

Ord ('T') = 84 ; chr(84)= 'T' ; ord('c') = 99; ord(chr(79))=79; chr(ord('*'))='*'; ord(succ('6')) =55;

2.4.5 Type chaîne :

Une chaine est une suite arbitraire de caractères du code ASCII, elle peut être formée de plusieurs mots.

Constante université= 'benbella'
Espace= ' '
Variables Nom, prénom : chaine de caractères.

- Les fonctions prédéfinis sur les chaines :

- Long (chaine) : retourne la taille de la chaine.
- Concat (chaine1, chaine2): fournit une chaine obtenue par concaténation de la chaine c1 et c2, c'est-à-dire rajouter à la fin de la chaine C1 la chaine C2.
-Efface (ch,p,n) : efface n caractères de ch à partir de la position p.

Important : En C, les chaînes sont implémentées par des tableaux de caractères se terminant par le caractère spécial \0.

2.4.6 Types énumérés (non standard)

En plus des types standards, l'utilisateur a la possibilité de créer ses propres types pour les objets qu'ils manipulent, ces types sont appelés types énumérés.
Les types énumérés concernent des objets pouvant prendre leur valeur dans une liste finie et ordonnée.

- Les mois de l'année (janvier, février...).
- Les couleurs (bleu,……..).
- Les jours de la semaine.
- Les notes de musique.
- Les noms de cartes à jouer (as, roi, dame...).
- Les marques de voiture.
- Les indications d'état civil (célibataire, marié, divorcé...).

Déclaration d'un type énuméré : On commence par le mot clé **enum**, suivi d'un identificateur qui représente le nom du type.
Exemple :
enum nationalité ={Algérienne, Tunisienne, Nigérienne ,Chinoise }
Variables N1, N2 : nationalité ;
enum couleur ={bleu, violet, rose ,vert }
Variables C1,C2 :couleur ;

- **Conversion Forcée de type**

Le langage C permet de faire une conversion forcée, mais elle est rarement utilisée.
<type> : expression ;
Exemples

```
char A=16;
int B=4;
float C;
C = (float)A/B;  /* conversion forcée */
printf (" %f\n",C); } //le %f indique le type float, voir manuel de référence C
```

⚠ Les contenus des variables A et de B restent inchangés.
⚠ La commande \n indique d'afficher le résultat sur une nouvelle ligne

2.5 Les Instructions de base

2.5.1 L'Affectation

Une affectation est une instruction qui stocke dans une variable la valeur d'une expression. C'est l'instruction de base à comprendre car nécessitant le concept de calcul (expression à évaluer) ou de contenu et de contenant (la variable).
Syntaxe générale : **Variable ← Expression (oubien variable = expression)**
Les opérandes à droite et à gauche du signe d'affectation (ici la flèche qui indique bien le sens droite à gauche ←) doivent être de même type.
*Ainsi A = B * 2 (si B contient 6, la valeur 12 sera affectée à A)*
Exemple 1:
A ←2 ; A est illustrée comme une boite (ou une enveloppe) pouavant contenir une valeur

La variable A de type entier contient la valeur 2. A | 2 |

A ←10 ; la valeur 10 écrase la valeur 2. A | 10 |

Incrémentation d'une variable :

Incrémenter A avec un pas de 1.

A ←A+1 (également A =A +1 est accepté);

En C

A=A+1 ; ou bien A++ ;

Décrémentation d'une variable :
Décrémenter A avec un pas de -1 revient à écrire A ←A-1 ;
En C
A=A-1 --A ou bien A-- ;

⚠ En C nous écrivons A = A+1. Pour un mathématicien, cette écriture est impossible parce que cela signifie que 0=1 ; l'égalité en informatique signifie qu'on évalue d'abord l'expression située à droite de l'égalité, puis on affecte le résultat à gauche du signe '='. C'est pour cette raison que des langages tels que Pascal utilisent ':=' pour l'affectation. Quant au test logique de l'égalité dans le vrai sens mathématique, le langage C le représente par '=='.

Par conséquent, l'instruction a = (b==c) en 'C' a tout son sens, 'a' est une variable logique qui reçoit le résultat d'un test logique (b = c ?).

Exemple 2 :
x←sin(x)+sin(y) ;
delta←b*b-4*a*c ;
Q←k*(S/d)*(T1-T2) ;

2.5.2 Les instructions d'entrées/sorties

L'algorithme a besoin de données en entrée, et fournit un résultat en sortie. Lorsqu'on utilise un ordinateur, le clavier permet de saisir les données et l'écran d'afficher un résultat ou des textes qui donnent des directives sur les données à fournir.

- **Écriture des données**

Ecrire permet d'afficher à l'écran

1- Afficher un texte : ***Ecrire ('message à afficher');***
2- Afficher la valeur d'une variable : ***Ecrire (nom_de_la_variable);***
3- Afficher message et valeurs : ***Ecrire ('texte ', non_var1, 'texte', non_var2,……etc);***

La virgule sépare les informations à écrire entre les ( ). Tout le texte contenu entre des guillemets est affiché à l'écran, alors que lorsqu'une variable apparaît dans l'instruction 'Ecrire' c'est sa valeur qui est affichée.

En C
printf("<format>",<Expr1>,<Expr2>, ...)

```
printf("%f",N); // le %f c'est le spécificateur du format de la variable N

printf("El salam alaykoum");
```

- **Lecture des données**

L'instruction « **lire** « permet d'introduire des données en utilisant le clavier.

En langage algorithmique	En C
lire (nom de variable) ;	**Scanf**("<format>",<AdrVar1>,<AdrVar2>, ...) ;

Exemple :

```
int A ,B;
scanf ("%d %d", &A, &B); /* les &A et &B représentent les adresses
en mémoire des variables A et B*/
```

Remarque : le %d, signifie qu'un entier sous forme décimal est saisi, on peut même préciser la taille %4d sur 4 positions. Un %s signifie chaine de caractères

On peut trouver sur le net ou dans les manuels du langage C, l'ensemble des formats d'entrée sortie disponibles.

2.6 La Construction d'un algorithme simple

Un programme choisi dans une machine à laver, ou un fichier Audio regroupant plusieurs morceaux de music sont des programmes à exécuter en bloc ou de manière séquentielle. La séquence se présente sous forme d'une suite ordonnée d'instructions regroupées en bloc délimité par les mots clés début et fin; c'est la manière la plus simple d'exécuter une suite d'instructions les unes après les autres. Les regrouper dans un bloc signifie que l'on ne s'arrête que lorsqu'on a terminé la dernière instruction du bloc. Un bloc est considéré comme une seule entité indivisible de code.

Langage algorithmique	**En C**
Algorithme **Debut** Instruction 1 ; Instruction 2 ; … Instruction n ; **Fin.**	**Main ()** **{** Instruction 1 ; Instruction 2 ; … Instruction n ; **}**

Voici un exemple de calcul séquentiel qui calcule les puissances d'un entier N

Exemple : Ecrire un algorithme qui Calcule et affiche N^3, N^6, N^9

***Astuce** : transformer les puissances de N en noms de variables*

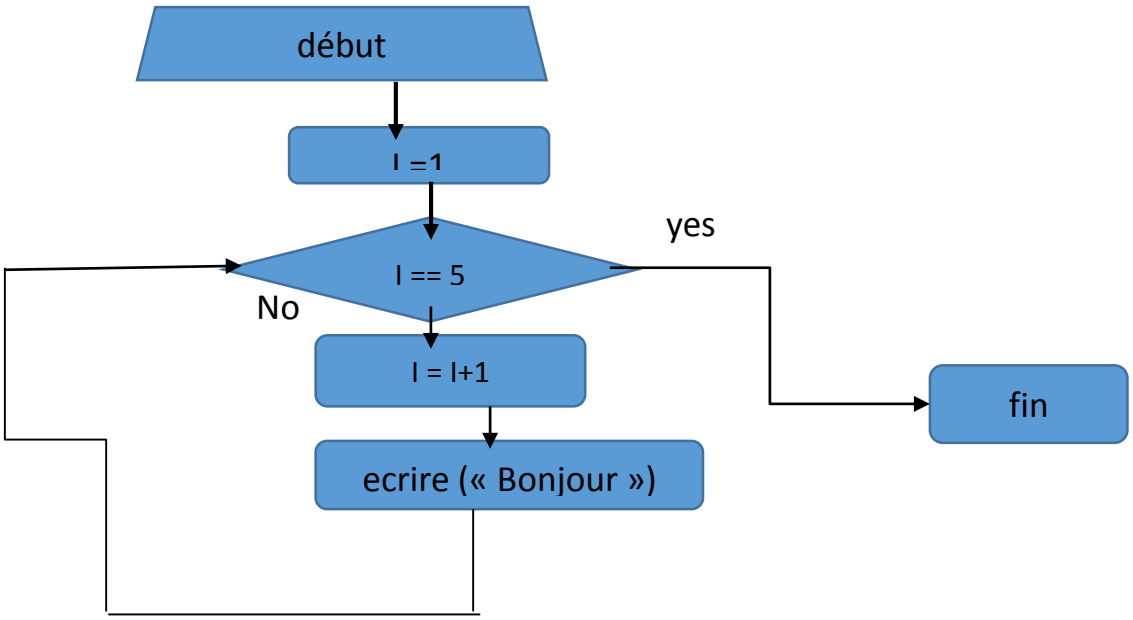
```
Algorithme puissance
Variables N, N3, N6, N9 : entier ;
Debut
Ecrire (″donnez la valeur de N″) ;
Lire (N) ;
N3← N*N*N ;
N6 ← N3*N3 ;
N9 ←N3*N6 ;
Ecrire (N,″ à la puissance 3 =″, N3,N,″ à la puissance 6 = ″, N6, N ,″ à la
puissance 9 =″ N9) ;
Fin.
```

Dans cet exemple, remarquer l'utilisation des variables déjà calculées au fur et mesure pour ne pas reprendre le calcul à zéro.
Un deuxième exemple concerne les instructions de lecture écriture :

Algorithme lecture
Variables nom, password : chaine ;
Debut
Ecrire ('introduisez votre nom') ;
Lire (nom) ;
Ecrire ('Bonjour ', nom) ;
Ecrire ('introduisez votre mot de passe : ') ;
Lire (password) ;
Si length (password) < 8 alors ('ecrire mot de passe incorrect') ;
Fin.

2.7 La représentation d'un algorithme par un organigramme

Un organigramme est un schéma qui représente le séquencement logique des instructions représentées par des rectangles. Un bloc d'instructions est donc une suite de rectangles superposés et exécutés en série. Le test ou l'expression booléenne (que nous verrons au chapitre suivant) est représenté par un losange. L'organigramme permet de visualiser le schéma du déroulement du programme, cependant quand celui-ci est de grande taille, ce schéma devient vite complexe à dessiner, c'est pour cette raison, il n'est utilisé que pour les débutants.



Dans cet exemple, la notion de boucle développée au chapitre 4 est bien visible, c'est un chemin fermé dans l'organigramme.

2.8 La Traduction en langage C

Par la suite, à chaque fois qu'il y aura un code langage algorithmique nous donnerons sa transcription en langage C dans un tableau à deux colonnes. La traduction est simple et repose sur la connaissance de quelques bribes en anglais.

Le bloc « debut- fin » d'instructions sera délimité par des accolades { }. Les commentaires sont introduits par les //, ou bien encadrés par /* commentaire */

Nous verrons également pour chaque instruction ou bloc, son équivalent en C. Le C étant un langage compilé, il n'acceptera aucune erreur de syntaxe (oubli d'une accolade par exemple). Le programmeur se conforme au template, c'est-à-dire au modèle type d'un programme pour qu'il soit exécuté, notamment la présence de la fonction main ( ) ou programme principal, ainsi que des pré-directives de compilations (les instructions # include).

Enfin pour terminer le chapitre donnons les étapes de construction d'un exécutable en utilisant le compilateur C, disponible gratuitement sur le net (avec linux notamment). Le compilateur C produit un programme en langage machine, donc exécutable sans passer par un interpréteur. Le processus de compilation en C est la conversion d'un code compréhensible par l'homme (programme C) en un code compréhensible par une machine (code binaire).

- Mettre un fichier contenant le code des fonctions (par exemple Fonct.c), un fichier contenant l'entête des fonctions et les déclarations (Fonct.h) et un fichier contenant la fonction main (Prog.c).
- Faire une compilation séparée avec option –c : gcc -c Prog.c Fonct.c, puis l'édition de liens avec gcc Prog.o Fonct.o -o pp. (pp sera le fichier exécutable c'est-à-dire qu'il aura l'extension .exe capable de s'exécuter directement)
- Sous windows, les boutons compiler, construire, exécuter .. du compilateur adopté font le travail de manière automatique. La touche de raccourci Ctrl + F9 permet d'éxécuter un programme C. Certains langages objets ne génèrent pas directement de l'exécutable mais du code objet ; cela permet d'inclure et de construire des programmes plus complexes avec l'option 'make project', puis le bouton Run.

2.9 Enoncés des exercices d'application

Exercice 2.1

Ecrire un algorithme qui calcule le volume V d'une sphère étant donné son rayon R , $V = 4\pi R^3$.

Exercice 2.2

Ecrire un algorithme qui calcule et affiche la somme des n premiers entiers naturels, la somme est égale à la division entière de n*(n+1) par 2.

Exercice 2.3

Ecrire un algorithme qui saisit le prix hors taxe d'une imprimante, le prix TVA et affiche le prix total.

Exercice 2.4

Ecrire un programme C qui permute deux nombres entiers.

Exercice 2.5

Ecrire un programme C qui calcule la résistance équivalente à trois résistances R1, R2, R3 , dans les cas suivants :

- les résistances sont branchées en série : R = R1+R2+R3.
- les résistances sont branchées en parallèle :
  R = (R1*R2*R3)/(R1*R2+R1*R3+R2*R3).

Exercice 2.6

Ecrire un programme C qui demande la saisie d'un entier et affiche sa valeur en base 8 et en base 16.

Exercice 2.7

Donner le type des informations suivantes : la marque d'un ordinateur, la capacité du disque dur, les composants d'un ordinateur, la marque et les unités.

Corrigés

2.1

Algorithme sphère
Variables R, V : réel ;
Début
Ecrire (" Entrez la valeur du rayon de la sphère ") ;
Lire(R) ;
V ←4*3 .14*R*R*R ;
Ecrire(" Le volume de la sphère est " ,V) ;
Fin.

2.2

Algorithme somme
Variables n, S : entier;
Début
Ecrire ("introduisez une valeur :") ;
lire(n);
S ← n*(n+1) div 2;
Ecrire ('La somme =', S) ;
Fin.

2.3

Algorithme prix
Variables pht, TVA ,P : réels;
Début
Ecrire ("introduisez le prix hors taxe :") ;
Lire (pht);
Ecrire ("introduisez la TVA:") ;
Lire (TVA);
P←pht*(1+TVA) ;
Ecrire ('Le prix de l'imprimante =', P) ;
Fin.

2.4

```
#include<stdio.h> /* ceci est appelée directive de pré-compilation, elle inclut la bibliothèque stdio.h pour exécuter les fonctions de lecture écriture scanf et printf */
main()
{ int  a, b, temp;
printf("Donnez les 2 valeurs");
```

```
scanf("%d",&a);
scanf("%d",&b);
temp=a;
a=b;
b=temp;
printf("la nouvelle valeur de a est : %d\n, la
nouvelle valeur de b est %d\n",a,b);
 }
```

2.5

```
main()
{
double  R1, R2, R3;
printf("Introduisez les valeurs pour R1, R2 et R3 :
");
scanf("%lf %lf %lf", &R1, &R2, &R3);
printf("Résistance en série : %f\n", R1+R2+R3);
printf("Résistance parallèle : %f\n",
(R1*R2*R3)/(R1*R2+R1*R3+R2*R3));
return 0;
}
```

2.6

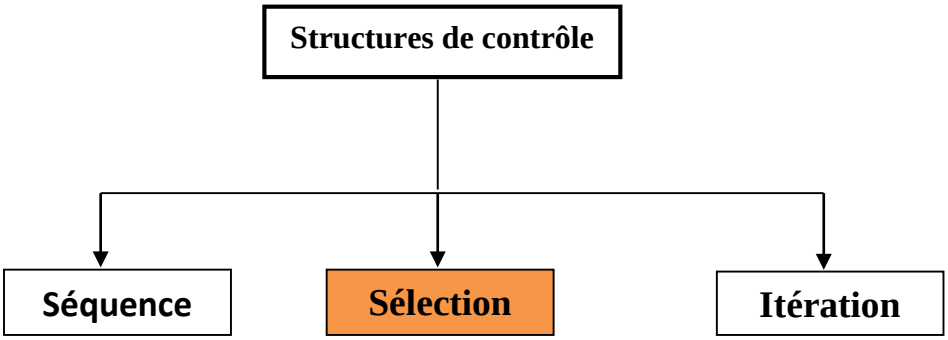
```
int main(void)
{
int A ;
printf("Introduisez un entier :");
scanf("%d ", &A);
printf ("le nombre en base 8 :%o .\n", A);
printf("le nombre en base 16 :%x .\n", A);
return 0;
}
```

2.7 : la marque d'un ordinateur : **chaine de caractères**, la capacité du disque dur : **en Go ou To (entier)**, les composants d'un ordinateur : **de type énuméré ou liste**, la marque d'un produit : chaine de caractères

3 Les structures conditionnelles

3.1 Introduction

Tout d'abord, nous rappelons que les algorithmes sont répartis en 3 grandes parties : La séquence, la sélection et l'itération.



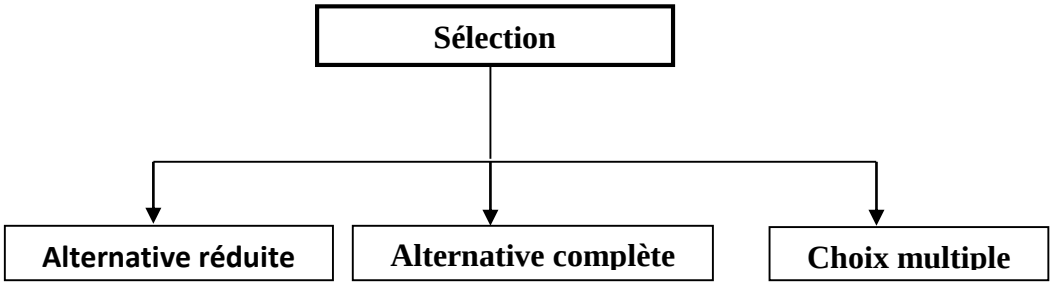
Dans le chapitre précédant, nous avons vu la séquence, ou les blocs d'instructions, dans ce chapitre nous allons développer les instructions conditionnelles ou la sélection.

« Si tu réussis ton bac, tu pourras t'inscrire à l'Université », c'est pour cela que tu peux lire cet ouvrage. L'exécution conditionnelle permet de faire deux choses différentes selon le cas qui se produit. L'instruction ne sera exécutée que sous condition. Le programme teste une condition, si la condition est satisfaite le programme traite le bloc concerné, dans le cas contraire, le programme traitera un autre bloc ou pas. On distingue l'alternative réduite, complète et imbriquée (choix multiple).

On dit souvent à un enfant : si tu termines ta soupe, tu peux regarder la télé.

Nuance, on ne lui dit rien sur le cas contraire. De manière un peu plus méchante : si tu termines ta soupe, tu peux regarder la télé, sinon…..

Il existe trois classes d'alternatives schématisées comme suit :



3.2 La structure conditionnelle simple (ou réduite)

Le programme teste une condition, si la condition est satisfaite le programme traite le bloc d'instructions, et continue son chemin.

En fait, il n'y pas de « dans le cas contraire », parce que dans le cas contraire, il continue tout simplement vers l'instruction qui suit directement l'instruction 'si'.

Langage algorithmique	En C
Si condition **Alors** Bloc d'instructions **Fin si**	**if** (condition) { Bloc d'instructions ; }

Exemple : algorithme qui calcule la racine d'un nombre en utilisant la fonction prédéfinie **racine carrée ( ).**

```
Algorithme racine
Variable x : entier ;
Debut
 Ecrire ("Saisir le nombre x :") ;
 lire (x) ;
 Si (x > 0) Alors
   r <-- racine carrée(x) ;
   Ecrire  ("la racine de", x, " est", r) ;
Fin si ;
Fin.
```

3.3 La Structure conditionnelle composée

Le programme teste une condition, si la condition est satisfaite le programme traite le bloc 1 d'instructions sinon il traite le bloc 2 d'instructions.

Langage algorithmique	*En C*

Si condition **Alors**
Bloc 1 d'instructions ;
Sinon
Bloc 2 d'instructions;
Fin si
	if condition
	{
	Bloc 1 d'instructions;
	}
	Else {
	Bloc2 d'instructions;}

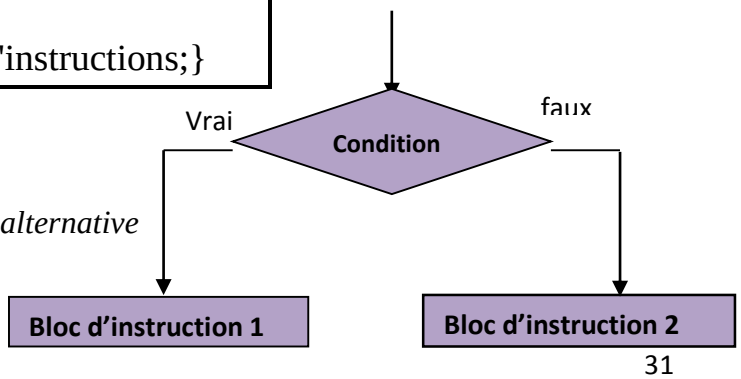


Figure 3.1Organigramme structure alternative

Exemple 1 :
Comparons deux nombres N1 et N2 quelconques, le morceau de code sera le suivant :

```
Si N1 = N2 alors écrire (‘les deux nombres sont égaux’)
        Sinon  si N1 >N2 alors écrire (‘N1 est supérieur à N2’)
                        Sinon écrire (‘N1 est inférieur à N2’) ;
                Fsi ;
Fsi ;
```

Remarquer qu’avec trois cas ou situations possibles, deux ‘si’ sont suffisants pour les traiter, le dernier cas est naturellement celui qui reste.

Exemple 2 :
Transformons l’algorithme de l’exemple précédent sur la racine carrée en rajoutant que l’algorithme doit afficher un message « erreur réessayez » s’il y a erreur, on va utiliser la fonction prédéfinie **racine_carrée**().

Algorithme racine
Variables x,r : entier ;
Debut
écrire ("donnez la valeur de x :") ;
lire (x) ;
Si (x > 0) Alors
   r <-- **racine carrée** (x) ;
   écrire ("la racine de", x, " est", r) ;
   Sinon
   écrire ("Erreur réessayez") ; // ici réessayer nécessite de ré exécuter
Fin Si
Fin.

***Remarque** : Le nombre de cas en choix simple est de deux, **true et false** selon la condition. C’est la raison pour laquelle, on l’appelle condition booléenne ou logique. Il est donc clair que cette condition peut prendre la forme d’une expression logique complexe combinant des OU et des ET, et dont la valeur finale vaut **true** ou **false**.*

3.4 La Structure conditionnelle de Choix multiple

Dès que le nombre de conditions devient important, on se retrouve dans des situations où il faut faire attention aux différents cas qui se présentent, et le code peut devenir important avec des risques de se tromper.

Voir le tableau suivant

Si condition1 **Alors**
Bloc 1 d'instructions;
Sinon Si condition2
Alors
Bloc 2 d'instructions
Sinon Si condition3
Alors
Bloc 3 d'instructions
Sinon
Bloc 4 d'instructions
Fin si
Fin si
Fin si
	if (condition 1)
	{ Bloc 1 d'instructions ;
	else if (condition 2)
	{
	Bloc 2 d'instructions ;
	else if (condition 3)
	{ Bloc 3 d'instructions
	else
	Bloc 4 d'instructions;
	}
	}
	}

On verra que dans ce cas, il vaut mieux utiliser les instructions de type 'case' ou switch, dites à choix multiples.

Exemple :

Cet algorithme affiche l'état de l'eau selon la température.

Algorithme température
Variable T: Entier ;
Début
Ecrire ("Entrez la température de l'eau :")
Lire(T) ;
Si (T<=0) Alors
Ecrire ("C'est de la glace") ;
Sinon
    Si (T < 100) Alors
    Ecrire ("C'est du liquide") ;
    Sinon
    Ecrire ("C'est de la vapeur") ;
    Fin si ;
Fin si;
Fin.

3.5 L'Instruction sélective le Si généralisé

On peut remplacer le Si généralisé par l'instruction selon Que (le case ou le switch).
Utiliser cette instruction lorsque le nombre de cas dépasse trois afin de ne pas s'embrouiller avec les if ….else if…

En Langage algorithmique	En C
Selon expression **vaut** Condition 1 : bloc d'instructions 1 ; … Condition n : bloc d'instructions n; **Default** : bloc d'instructions n+1; **Fin selon**	**switch (expression)** case value 1 : bloc of instructions 1; break ; **……..** case value n : bloc of instructions n; break ; **Default** : bloc of instructions n+1;

Explication
Si condition 1 est vrai alors le bloc d'instructions 1 est exécuté et on quitte le selon.
Si condition 1 est fausse alors on passe alors à l'évaluation de Condition 2 et
Ainsi de suite.
Certaines valeurs peuvent être oubliées, elles sont donc regroupées dans la dernière rubrique **« default ».**
Le langage C offre l'instruction **switch** qui représente le *« if généralisé »*, l'instruction **default** est optionnelle.
Exemple 1 :
Algorithme qui donne les jours d'un mois donné de l'année 2022

```
Algorithme nombre jours
Variables mois, nbrj : entier ;
Debut
  Ecrire ("donnez un mois") ;
  Lire (mois) ;
  Selon mois vaut
   2: nbrj←28 ;
   4,6,9,11: nbrj←30 ;
   default : nbrj←31;
  Fin selon
  Ecrire ('le nombre de jours est :', nbrj) ;
Fin.
```

Exemple 2 :

```
enum nat {algérienne, tunisienne ,anglaise} ; /*nat
est un type énuméré */
Void affichage_nat(int nat)
{ Switch (nat)
  {
  case algerienne : printf("Algérie" ) ; break ;
  case tunisienne :printf("Tunisie" ) ; break ;
  case anglaise :printf("Angleterre" ) ; break ;
  Default : Exit (0);
  }
}
```

On l'utilise souvent pour dialoguer avec l'user **:**

```
#include <stdio.h>
main()
{    char c;
    printf("entrez votre choix \n") ;
    printf("entrez A, B ou C \n");
    while ( c!='A' && c!='B' && c != 'C')
    c=getchar();  //
     switch (c)
    {
    case 'A' : /* traitement dans le cas A */ break;
    case 'B' :/*traitement dans le cas B*/ break ;
    case 'C' :/*traitement dans le cas C*/ break ;
     }
```

L'instruction ***break*** en langage C permet de quitter un switch. Elle est également utilisée dans une boucle afin de sortir de la boucle (ne plus aller jusqu'à la fin si une condition est vérifiée) et de passer à l'instruction qui suit le bloc.

A***stuce*** *: pour une meilleure lisibilité de l'algorithme, décaler vers la droite l'écriture d'un nouveau bloc ou un nouveau sinon de façon à mettre les 'fin si' sur la ligne en dessous du 'si' correspondant.*

3.6 Le branchement inconditionnel

Enfin pour terminer ce chapitre, il faut savoir qu'il existe toujours un moyen d'aller vers un point précis du programme. Ce point est référencé ou labélisé par une étiquette qui doit être aussi déclarée; cette instruction s'appelle l'instruction **goto** ou le branchement inconditionnel.

Son schéma global est le suivant

```
 Lable  etiquette ; // étiquette doit être déclarée comme un point
…….
Etiquette :    instruction_labelisée ;
…….
        goto etiquette ;
```

Le programme se rend directement (sans condition) à l'instruction labélisée par l'étiquette. Le ***Goto*** peut venir avant ou après l'étiquette et plusieurs goto peuvent renvoyer à la même instruction.

Cette instruction n'est plus utilisée de nos jours car elle prête à confusion et peut donner lieu à des erreurs de compréhension. Bien qu'elle se voie clairement sur l'organigramme (notamment avec les retours arrière) où elle peut traduire directement ce dernier, elle est le résultat d'une mauvaise structuration du programme (des boucles enchevêtrées).

3.7 Enoncés des exercices d'application

Exercice 3.1

Ecrire un algorithme qui permet de résoudre une équation du 2ème ordre en utilisant la structure à choix multiples « **Selon** expressions ».

Exercice 3.2

Ecrire un algorithme qui détermine la valeur la plus grande de 3 réels.

Exercice 3.3

Un magasin de matériels informatiques vend des **Ardouino** avec une TVA à 5% et des CPU Intel i7-9700K à 6.5%. Écrire un programme qui introduit le prix hors taxe du matériel, saisit au clavier le type du matériel et affiche le taux de TVA et le prix TTC du matériel, pour faciliter le programme, le **Ardouino** se représente par le caractère « A », et le CPU par le caractère « C ».

Exercice 3.4

Écrire un programme en C qui lit deux nombres entiers a et b et donne le choix à l'utilisateur :

1. de savoir si la somme a + b est paire.
2. de savoir si le produit a*b est pair.
3. de connaître le signe de la somme a + b.
4. de connaître le signe du produit a*b.

Exercice 3.5

Les années bissextiles sont les années qui sont :

- Soit divisibles par 4 mais non divisibles par 100.
- Soit divisibles par 400.

Écrire un programme en C qui montre si une année donnée est bissextile.

Exercice 3.6

Écrire un programme en C demandant à l'utilisateur de simuler une calculatrice

Exercice 3.7 non corrigé

Écrire un programme en C demandant à l'utilisateur de classer 3 nombres par ordre décroissant.

Exercice 3.8 non corrigé

Ecrire un algorithme permettant de calculer le montant des allocations familiales sachant que le montant dépend du nombre d'enfants et du statut des 2 parents salariés ou pas :
- Si le nombre d'enfants est inférieur ou égale à 2 alors les allocations sont de 3000 DA/enfant si un seul des parents est salarié.
- Si le nombre d'enfants est strictement supérieur à 4 alors les allocations sont de 2000 DA / enfant quel que soit le statut des parents.

Corrigés

3.1

```
Algorithme équation
Variables a, b, c, delta : entier ;
Début
Ecrire (‘introduisez les valeurs des coefficients : ‘) ;
Lire (a, b, c ) ;
Si a≠0
Alors delta ← b*b – 4 *a *c ;
Selon delta  vaut
      0  : Ecrire (‘la solution est :’,-b/ 2a ) ;
    > 0 : Ecrire (‘les solutions sont:’,-b - √delta)/2a, -b + √delta)/2a ) ;
   default : Ecrire (‘pas de solution ‘) ;
Fin selon
Fin.
```

3.2

```
Algorithme maximun
Variables: x,y,z, max :réel ;
Debut
Ecrire("Tapez le 1er nombre:") ;
Lire(x) ;
Ecrire("Tapez le 2ème nombre:") ;
Lire(y) ;
Ecrire("Tapez le 3ème nombre:") ;
Lire(z) ;
    Si (x>y) alors
          max←x ;
          Sinon
          max←y ;
    Fin si
    Si  z> max
    Alors max←z ;
    Fin si
    Ecrire ("Le plus grand nombre est:", max) ;
Fin.
```

3.3

```
int main(void)
{
float p, tva, ttc;
char materiel;
printf("Entrez le prix : ");
scanf("%f", &p);
printf("matériel  de type Ardouino ou CPU ? ");
getchar();
materiel = (char) getchar();
if (materiel == 'A')
{ tva = 2.5; }
else
{ tva = 16.5; }
ttc = prix * (1.0 + tva / 100.0);
printf("Prix TTC : %f et TVA vaut %.1f\n", ttc, tva);
return 0;
}
```

3.4

Programme C

```
#include<stdio.h>
#include<conio.h>
int main(void)
{
int a, b;
char choix;
printf("Entrez 2 valeurs : ");
scanf("%d %d", &a, &b);
printf("Tapez\n");
printf("1  si la somme est paire\n");
printf("2  si le produit est pair\n");
printf("3  le signe de la somme\n");
printf("4 le signe du produit\n");
getchar();
  choix = (char) getchar();
  switch (choix)
```

```
  {
  case '1':
     if ((a + b) % 2 == 0)
     printf("La somme est paire\n");
     else
     printf("La somme est impaire\n");
     break;
  case '2':
     if ((a * b) % 2 == 0)
     printf("Le produit est pair\n");
     else
     printf("Le produit est impair\n");
     break;
  case '3':
     if (a + b >= 0)
     printf("La somme est positive\n");
     else
     printf("La somme est négative\n");
     break;
  case '4':
     if (a * b >= 0)
     printf("Le produit est positif\n");
     else
     printf("Le produit est  négatif\n");
     break;
  default:
     printf("erreur...\n");
  }
 return 0;
}
```

3.5

```
main()
{ unsigned short annee;
printf(”Saisissez une annee\n”);
scanf(”%hu”,&annee);
if(annee % 400 == 0 || annee % 4 == 0 &&annee % 100
!= 0 )
```

```
    printf(”L'annee %hu est bissextile”,annee);
    else printf(”L'annee %hu est non
    bissextile”,annee);
printf(”\n”);
}
```

3.6

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main(){
 float G, D, resultat;
 int e=0;
 char operateur;
 printf("Operande gauche:");
 scanf("%f",&G);
 getchar();
 printf("Operateur:");
 scanf("%c",&operateur);
 printf("Operande droit:");
 scanf("%f",&D);
 getchar();
switch(operateur)
{
 case '+' : resultat=G + D; break;
 case'-' : resultat=G - D; break;
 case'/' :resultat=G / D; break;
 case '*': resultat=G * D; break;
 default: e=1;
} ;
if(e)
printf("erreur" );
else
printf("%.2f %c %.2f=%f\n",G,operateur, D);
}
```

4 Les boucles ou Algorithmes Itératifs

4.1 Introduction

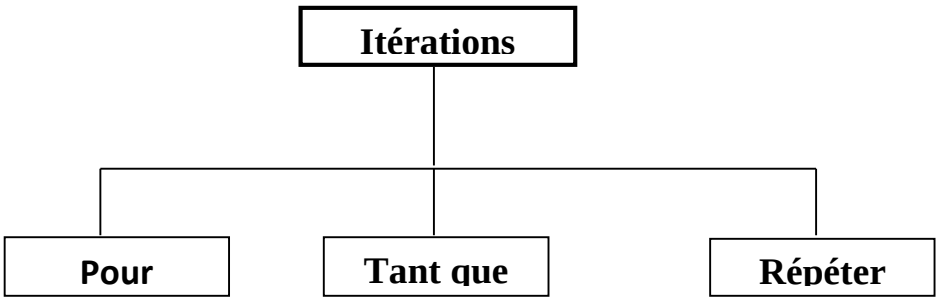
Nous avons deux façons d'apprendre un texte en général : la première est « *Tant que vous n'avez pas appris, répétez la lecture de ce paragraphe* ». La seconde « *Répétez dix fois la lecture de ce paragraphe* ». Dans la première version, l'opération de lecture est soumise à condition, elle est répétée autant de fois que nécessaire, elle peut même être **infinie** si la personne a la tête dure. Dans la seconde, le nombre de répétitions est connu à l'avance (ce n'est pas toujours le cas des boucles tant que ou répéter).

Sur un organigramme, on peut voir schématiquement une boucle par un chemin fermé, c'est-à-dire qu'on revient plus haut à une instruction que l'on a déjà exécuté. La boucle doit comporter au moins une condition pour pouvoir s'en sortir.

On appelle itération, toute répétition de l'exécution d'un traitement.

A la notion d'itération, est associée la notion de boucle.

⚠ Le nombre d'itération doit être fini : soit par une condition, soit par un compteur. Il existe trois types de structures d'itérations : *répéter, tant que, pour*.



4.2 La boucle répéter

Une action ou un groupe d'actions est exécuté répétitivement jusqu'à ce qu'une Condition soit vérifiée. La condition est formulée par une expression booléenne

Langage algorithmique	En C
Répéter /* instructions */ **Jusqu'à** Condition	**Do** { /* instructions */ } **while** (Condition)

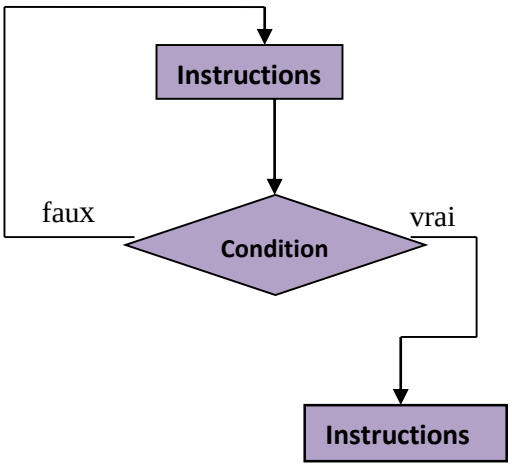


Figure 4. 1 Organigramme de la boucle répéter

Remarque : la vérification de la condition s'effectue après les actions. Celles-ci sont donc exécutées au moins une fois.

Exemple :
Algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre (pour N>1). On souhaite afficher uniquement le résultat final, on va utiliser la boucle répéter par deux fois.

```
Algorithme somme
Variables N, i, S: Entier ;
Debut
  Ecrire ( "Entrez un nombre : ") ;
  Répeter Lire (N) jusqu'à N>1;  // précaution en cas de mauvaise saisie
  S← 0 ;
  i ←1 ;
  repeter
     S ← S + i ;
     i  ← i+1;
     jusqu'a   (i=N)    /* condition*/
   fin
 Ecrire ( "La somme est : ", S) ;
Fin.
```

4.3 La boucle Tant que

Lorsque l'ordinateur rencontre cette structure :
• La condition est testée.
• Si la condition est fausse, l'instruction ou les instructions du bloc ne sont pas exécutées et on passe aux instructions suivantes (après la structure de contrôle).
• Si la condition est vraie, l'instruction ou les instructions du bloc sont exécutées, répétitivement *tout le temps où une condition est vraie*. Il doit y avoir un lien entre la condition et les instructions afin de changer la condition à un certain moment.

Remarques :

- La vérification de la condition s'effectue avant les actions. Celles-ci peuvent donc ne jamais être exécutées.
- On déduit donc que le corps de la boucle *« répeter »* est executé au moins une fois
- Pour transformer une boucle while en do while, la condition est remplacée par sa négation.

En Langage algorithmique	En C

Tant que Condition **Faire**
instruction 1;
...
instruction n;
 	
Fin Faire
	while (condition)
	{
	instruction 1;
	...
	instruction n;
	}

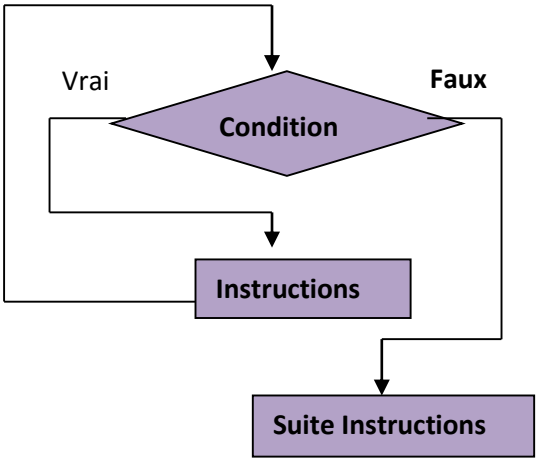


Figure 4.2 Organigramme de la boucle tant que

Exemple 1 : On reprend l'exemple précédent en utilisant la ***boucle tant que***

Algorithme somme
Variables N, i, S : Entier ;
Debut
Ecrire ("Entrez un nombre : ") ;
Lire (N) ;
S ← 0 ;
i ←1 ;
Tant que i<=N faire
S← S + i;
i ← i+1 ;
fin faire // on peut également écrire fait, ou fin tant que ou ftq
Ecrire("La somme est : ", S) ;
Fin.

Exemple 2 : Calculez le factoriel N, (N! = 1x 2 x 3 x... x N) d'un entier naturel.

Algorithme factoriel
Début
Variables I,F: Entier;
I←1; F← 1;
Tant que (I ≤ N) faire
F←F * I;
I←I + 1;
Fin tant que
Ecrire ('le factoriel est :',F);
Fin.

Remarque

Le while en python a la structure while condition : bloc_instructions. Il n'y a pas de { }, le retour chariot deux fois terminera le premier bloc.Voyons maintenant un célèbre paradoxe dit paradoxe de Zenon. Dans la Grèce antique, le philosophe avait imaginé une course entre disons un lièvre (L) et une tortue (T). Puisque le lièvre court plus vite (supposons pour simplifier deux fois plus vite), il va donner de l'avance à la tortue. Soit D la distance qui les sépare. Un, deux, trois, Feu !lorsque le lièvre aura atteint la position de la tortue, celle-ci aura parcouru la moitié de la distance D. Et là problème ! si on répété le processus, la nouvelle position du lièvre après avoir parcouru la nouvelle distance D/2, la tortue aura avancé de D/4. Et donc théoriquement, le lièvre ne rattrapera jamais la tortue, parce qu'on va avoir des distances infiniment petites mais jamais égales. Mais en pratique (c'est d'ailleurs le paradoxe, le lièvre va non seulement rattraper la tortue mais la dépasser). Essayons de donner une solution langage algorithmique à notre paradoxe, la fin du programme correspond ici à la fin de la boucle tant que signifie que lièvre a rattrapé la tortue. Voilà un premier programme de simulation.

L'astuce, en boucle « tant que » est justifiée. Remarquer l'utilisation par la condition négative de la boucle : tant que L n'a pas rattrapé T faire.

Le lecteur pourra revenir sur ce problème dans le tome 2. C'est d'ailleurs un bon exemple de récursivité.

Le morceau d'algorithme correspondant est :

```
L= x1; T= x2;
Tant que  L != T faire
  X1= L;
  L= T
  T= T+ (T-X1)/2;
Fin tant que;
```

Boucles infinies

On peut se retrouver dans des cas de boucles qui s'exécutent de manière indéfinie, en bloquant ainsi la suite du programme ; cela arrive quand on oublie par exemple d'incrémenter une variable qui sert à contrôler la boucle.

```
#include<stdio.h>
int main()
{
float x = 1;
while (x<10)
    {
      printf ("x est égale  %f\n", x);
     }
  }
```

C'est équivalent à écrire : while (true) { }

Il faut faire en sorte que la condition change dans le corps de la boucle, sinon la condition restera toujours vraie, et la boucle est dite infinie. Le programme ne s'arrête qu'après une interruption forcée (Ctrl+Alt+Suppr).

Remarque :

En fait le nombre d'itérations dans une boucle pour n'est pas forcément connu pour le programmeur, mais il l'est pour le compilateur. On peut par exemple (voir tome2) faire un travail pour l'ensemble des arguments d'une fonction sans connaitre leur nombre, ou bien comme dans l'exemple ci-dessous en python, faire pour un ensemble ou une liste :

```
for v1, v2 in zip( listA, litB) :
 ……
```

4.4 La boucle Pour

La structure **pour** utilise une variable appelée ***compteur*** afin de contrôler le nombre de répétitions (nombre de boucles), les valeurs initiales et finales vi et vf sont incluses dans le comptage du nombre de boucles.

Par défaut le pas d'incrémentation (ou de décrémentation) est égal à 1, mais il est éventuellement possible de spécifier un autre pas d'incrémentation.

Attention, dans la boucle Pour, il n'est pas conseillé de modifier la valeur du compteur comme dans la boucle ***while***, cette valeur est automatiquement incrémentée à chaque itération.

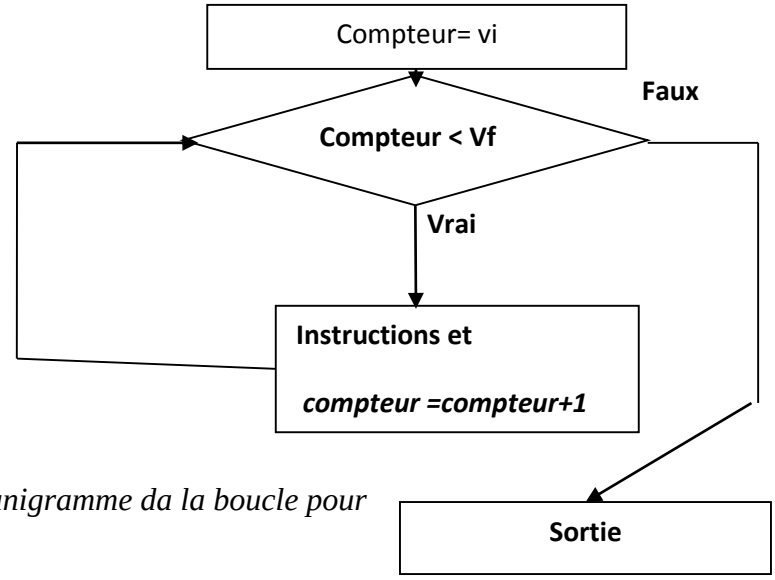


Figure 4.3 Organigramme da la boucle pour

Langage algorithmique	En C
Pour i de vi à vf **Faire** bloc d'instructions, **Fin**	**for** (/* Expression/Déclaration */; /* Condition */; /* Expression */) { /* Instructions à répéter */}

***Astuce** : Les boucles 'Pour' sont très utilisées avec les tableaux à une ou deux dimensions, car l'indice sera celui du tableau. Le parcours du tableau est fini et le traitement concerne tous les éléments.*

Exemple : On reprend l'algorithme qui demande un nombre de départ, et qui calcule la somme des entiers en utilisant la boucle pour.

Algorithme somme
Variables N, i, S : entier ;
Debut
Ecrire ("Entrez un nombre : ") ;
Lire (N) ;
S ← 0 ;
Pour i de 1 à N faire
S← S + i ;
fin pour
Ecrire ("La somme est : ", S) ;
Fin.

4.5 Imbrications des boucles

Dans de nombreux cas on est amené à utiliser des boucles l'une à l'intérieur de l'autre, à l'image des poupées russes. On n'est pas obligé d'imbriquer des boucles

de même type, c'est-à-dire on peut mélanger « **pour »** avec « ***tant que*** « par exemple. Ci-dessous, nous avons trois boucles imbriquées.
Remarquer la structuration de l'algorithme par le niveau du décalage d'écriture des instructions. Cette structuration a banni le recours aux instructions GoTo.

Pour i de 1 à n faire
Instructions ;
Pour j de 1 à m faire
……..
Tant que condition faire
Instructions ;
Fin tant que
Fin pour
Fin pour.

L'exercice le mieux indiqué pour comprendre les boucles imbriquées est celui du produit matriciel.
Si C (nxp) -de dimension n lignes et p colonnes- est la matrice résultat du produit des deux matrices A(nxm) et B(mxp), trois boucles imbriquées sont comme suit dans le morceau de code suivant:

```
Pour i de 1 à n Faire
  Pour j de 1 à p Faire
   C[i,j] = 0 ;
   Pour k de 1 à m Faire
      C[i,j] = C[i,j] + A[i,k] * B[k,j];
   Fpour ;
 Fpour ;
Fpour ;
```

Enfin un bon exercice en langage C qui génére l'affichage suivant

```
1 ! = 1 x1 =1
2 ! = 1x2  =2
3 !=  1 x2 x3 = 6
4 ! = 1 x2 x3 x4 = 24
……
9 ! = 1 x2x 3 x……..x 9 = 362880
```

```
# include <stdio.h>

main()
{  int i,j,f;
 for(i=1;i<=9;i++)
 {
  f=1;
  printf("%d!=1x",i);
    for (j=2;j<=i;j++)
     {
     f=f*j;
     if (j<i)
      printf("%dx",j);
     }
  printf("%d =%d \n",i,f);
 }

}
```

4.6 Enoncés des Exercices d'application

Exercice 4.1

Ecrire un algorithme qui calcule la puissance x^n, pour x et n entiers.

Exercice 4.2

Deux entiers N, M sont dits amicaux si la somme des diviseurs de N (N non compris) vaut M et si la somme des diviseurs de M (M non compris) vaut N.
Par exemple, 220 et 284 sont amicaux car :
220 est divisible par 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 et par 110
1+2+4+5+10+11+20+22+44+55+110=284
284 est divisible par 1, 2, 4, 71 et par 142, leur somme vaut 220, c'est à dire
1+2+4+71+142=220

Exercice 4.3

Ecrire un algorithme qui permet de calculer la somme :

$$S= -1+\frac{1}{2}-\frac{1}{3}+\frac{1}{4}-\frac{1}{5}+\frac{1}{6}-\frac{1}{7}+\frac{1}{8}-\cdots\frac{1}{n}$$

Exercice 4.4

Ecrire un algorithme qui permet d'inverser un nombre.

Exercice 4.5

Ecrire un algorithme qui permet de calculer le PGCD de deux nombres.

Exercice 4.6

Ecrire un programme en langage C qui calcule la fonction suivante :

$$F = 1 + \frac{nx}{1!} + \frac{n(n-1)x^2}{2!} + \frac{n(n-1)(n-2)x^3}{3!} + \frac{n(n-1)(n-2)\ldots x^n}{n!}$$

Exercice 4.7

Sans utiliser la fonction puissance écrire un algorithme qui calcule pou n
$S = 1 + 2^2 + 3^3 + 4^4 \ldots\ldots + N^N$

Exercice 4.8 non corrigé

Ecrire un algorithme qui compte le nombre de mots dans une phrase. On suppose que les mots sont séparés par le caractère ' ' (espace) et que la phrase se termine par un point.

Les exercices suivants sont d'un niveau plus élevé et laissés pour réflexion.

Exercice 4.9 non corrigé

On rappelle que **le crible d'Eratostène** est un algorithme inventé par les grecs pour déterminer les nombres premiers inférieurs ou égaux à N, entier naturel.

L'astuce est de faire plusieurs passage sur cet intervalle, et éliminer à chaque passage les multiples du nombre premier sélectionné jusqu'à ne plus pouvoir éliminer de nombres. Construire deux tableaux, l'un contenant les éléments, l'autre un tableau de booléans, indicateurs si l'élément est premier ou non.

Exemple dans [1, 26], on élimine d'abord tous les entiers pairs, multiples de 2, au $2^{ième}$ passage les multiples de 3 qui seront 9, 15, et 21 (car 6, 12 et 18 sont éliminés au premier passage), …etc.

Exercice 4.10 (corrigé)

Deux joueurs lancent un dé, le joueur qui a le plus grand résultat marque un point. On arrête le jeu lorsque l'un des joueurs (le gagnant) atteint la valeur 12. Simuler ce jeu (par simple lecture des valeurs à chaque itération) que l'on connait aussi sous le nom de jeu de l'oie.

Exercice 4.11 non corrigé

Calculer les sommes

S1 = 1+ ½ +1/4+ 1/8+ 1/16+ …..

S2 = 1+ ½+ ¼+ 1/7+ 1/11 +…..+ 1/46

Calculer exp(x) d'ordre n en utilisant l'approximation (x et n en input)

$\exp(x) = 1 + x/1! + x^2/2! + \ldots + x^n/n!$

Corrigés

4.1

Algorithme puissance

```
Variables p, x : reel ;
Variables i,n :entier ;
Début
  Ecrire (‘donnez les deux valeurs :’) ;
  Lire(x) ;
  Lire (n) ;
  p ←1 ;
  Pour i de 1 à n faire
     p ←p*x ;
  fin pour ;
 Ecrire(‘la puissance  de x est :‘ ,p) ;
Fin.
```

4. 2

Algorithme nombres_amicaux

```
Variables i, N, M , S1, S2 : entiers ;
 Début
 Ecrire (″Entrez le 1er entier : ″) ;
 Lire(N) ;
 Ecrire (″Entrez le 2ème entier : ″) ;
 Lire(M) ;
S1←0 ;
S2←0 ;
Pour i de 1 a N-1 faire
   Si (N mod i) =0
   Alors  S1←S1+i;
   Fin si ;
Fin pour ;
```

```
  Pour i de 1 a M-1 faire
    Si (M mod i) =0 Alors
     S2←S2+i;
    Fin si
  Fin pour
Si (S1=M) et (S2=N) alors
Ecrire (″ (N, M) sont des nombres amicaux″)
     Sinon
     Ecrire (″ (N, M) ne sont pas des nombres amicaux″)
Fin Si ;
```
Fin.

4.3

Algorithme suite
Variable S: réel ; i, n : entier ;
Début
Ecrire (‘donnez la valeur de n’) ;
Lire (n) ;
S←0 ;
Pour i de 1 à n faire
    Si i mod 2= 0 Alors S←S+ 1/i;
                   Sinon S←S- 1/i ;
    Finsi ;
 Fin pour ;
Ecrire(‘la somme est :‘ ,S) ;
Fin.

Astuce: *une meilleure solution est de prendre un entier j initialisé à 1, et devient de signe opposé à chaque itération (-j), cela évite de tester i mod 2 à chaque itération (ce qui est un appel de la fonction modulo à chaque fois).*

La dernière partie du code devient

```
  j =1; S = -1;
  Pour i de 1 à n faire
    S = S + j /i;
    j = - j ;
  fin pour
  fin.
```

4.4

Algorithme inverser

```
Variables  X,N,VD, NINV : entier ;
Debut
N←0 ;
VD←1 ;
NINV←0 ;
Répéter
     Ecrire ("Entrez un chiffre");
     Lire (X);
     Si  (X<0 ou X>9) alors
     Ecrire ("erreur ") ;
     Sinon
      Si (X>0 et X≤ 9) alors
        NINV ← NINV+VD*X;
        N←N+1;
       VD ←VD*10;
      Fin si ;
     Fin si ;
  jusqu'à  (X=0)
Ecrire ("La valeur du nombre renversé est : ", NINV)
Fin.
```

4.5

Algorithme PGCD

```
Variable A, B : entier ;
Debut
Répéter
  Ecrire ("Entrer l'entier A ≠0 : ") ;
  Lire (A) ;
Jusqu'à (A≠0)
Répéter
Ecrire ("Entrer l'entier B≠0: ");
lire(B) ;
jusqu'à (B≠0)
Tant que (A≠ B) faire
        Si (A>B)
        alors  A←A-B ;
```

Sinon B←B-A ;
Fin si
Fin Tant que
Ecrire ("Le PGCD = " ,A) ;
Fin.

4.6

```
include<stdio.h>
main ()
{
int  i, pas, n, x ;
double   puiss, fact, F, prod ;
printf (" donnez la valeur de x") ;
scanf (" %d", &x) ;
printf (" donnez le degré ") ;
scanf (" %d", &n) ;
for (F=1, prod=n, fact=1, puiss=1, pas=n, i=1 ;
i<=n ;  i++)
{
puiss=puiss*x;
F=  F + prod * puiss/fact;
pas=pas-1 ;
prod=prod*pas ;
fact=fact* (i+1);
}
printf (" Le résultat de la fonction = %lf", F) ;
}
```

Corrigé 4.7

Algorithme somme _p ;

Var i, j, S, S1, N : entier ;

Début

Lire (N);

S =0;

Pour i = 1 à N faire

S1 = i ;

```
  Pour j 1 à i faire
      S1 = S1 * S1 ;
  Fpour;
  S = S + S1 ;
 Fpour ;
 Écrire (“ la somme est =”, S);
Fin.
```

Corrigé 4.9

```
Algorithme jeu ;
 Variable  s1, s2, x, y : entier ;
 debut
  S1 =0 ;
  S2 =0 ;
  Repéter
      Lire (x) ; // ou mieux repeter lire(x) jusqu’à (x >=1 et x<= 6) ;
      Lire (y) ; // ou repeter lire(y) jusqu’à (y >=1 et y<= 6) ;
      Si x >y alors s1 =s1 +1 ;
      Sinon
            si y>x alors s2 = s2 +1 ;        // en cas d’égalité pas de points
            fsi ;
        fsi ;
      Jusqu’à (s1 =12 ou s2 = 12) ;
      Si s1 =12  alors ecrire (« joueur 1 gagnant »)
                  sinon ecrire ( « joueur 2 gagnant ») ;
      Fsi ;
 Fin.
```

5. Tableaux & chaines de caractères

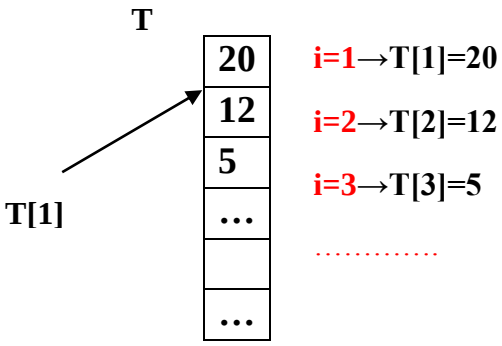
5.1 Introduction

Si le nombre de variables -*de même type*- à traiter est important, il n'est pas adéquat de les nommer X1, X2, ….Xn. Il serait intéressant de les rassembler dans une seule structure où chaque variable sera accessible par son indice. Cette variable est un tableau unidimensionnel ou tableau linéaire qui est une variable indicée permettant de stocker n valeurs de même type. Le nombre maximal d'éléments précisé à la déclaration représente la dimension du tableau. Le type du tableau est le type de ses éléments.

Dans un tableau tous les éléments sont homogènes de **même type**, La position d'un élément du tableau s'appelle ***indice*** ou ***rang*** de l'élément. La dimension est le nombre d'éléments du tableau.

5.2 La déclaration du type tableau

Un tableau est caractérisé par:
- Son nom (identificateur).
- Sa taille (le nombre d'éléments).
- Le type de ses éléments.
- Les indices pour accéder aux éléments du tableau.



Remarque importante: en C, l'indice commence à 0 et non à 1, le premier élément du tableau sera donc référencé par T[0].

Syntaxe 1

Type Nom_Tableau= Tableau [1..taille] de type_element
Variable nomVariable : Nom_Tableau

Exemple :

```
Type T=Tableau [1..10] de entier ;
Variable Tab :T ;/* T et Tab de même type*/
```

Syntaxe 2
nom [taille] : type des elements ;
T[n] : type des éléments**;**
Avec **T**: nom du tableau et **n** le nombre des éléments.
C'est la syntaxe 2 qui sera élaborée dans ce cours

⚠ *: On ne doit jamais dépasser la capacité du tableau*

Implémentation en C

On déclare un tableau (statique) par type, identificateur, et le nombre d'éléments du tableau. La dimension n du tableau indique une réservation statique (à l'avance) de n cases contiguës et de même type en mémoire.
type identificateur [longueur];

Exemples :

Langage algorithmique	En C
Tab[200] : Réel ;	Float Tab[200] ;
T [100] : Entier ;	int T [100] ;
nom [20] : caractère ;	char nom [20] ;
Constante max=100; Tab[max]: réel;	#define max 100 Float Tab[max] ;

***Astuce** : define est une macro de précompilation « # » elle servira à remplacer max par la valeur déclarée dans toutes les occurrences de celle-ci avant la compilation du programme .*

5.2.1 Accès aux éléments d'un tableau

Les éléments d'un tableau sont des cases rangées successivement dans la RAM. Les éléments d'un tableau sont numérotés par des indices. Par exemple, les éléments du tableau Tab déclarés ci-dessous, qui comporte 10 éléments. Les indices sont schématisés en dessous du tableau : 1, 2, ... , 10. Les éléments sont du tableau sont donc les valeurs 20, 30, …i.e le contenu de tab[1], tab[2], tab[3]...

Tab[1] : valeur du tableau d'indice i=1

20	**30**	**40**	**90**	**60**	**70**	**80**	**99**	**100**	**50**
1	2	3	4	5	6	7	8	9	10

Tab désigne le nom du tableau (qui sera aussi l'adresse du premier élément).

L'élément d'indice 3 dans le tableau Tab est noté Tab [3].

Plus généralement, l'élément d'indice i dans le tableau tab est noté tab[i].

L'indice i doit impérativement être un nombre entier positif ou nul, il peut aussi être représenté par une expression arithmétique qui une fois calculée donne un entier.

Exemple 1 : tab [3*m+2], où m est un entier.

Exemple 2 : Algorithme qui remplit les valeurs (2, 4, 6, 8, 10, 12, 14, 16, 18, 20) dans un tableau Tab.
Algorithme tableau
Début
Tab [10], i : entier;
pour i de 1 à 10 faire
Tab [i] ← 2*i; // on pouvait sans boucle écrire tab[10] = {2,4,6,…}
fin pour;
Fin.

⚠ En langage C, nous avons signalé que les indices des éléments d'un tableau commencent à 0 et non pas à 1. On verra par la suite que ceci est dû au fait que le nom du tableau désigne aussi l'adresse du premier élément.

5.2.2 Saisie et affichage d'un tableau

- **Lecture**

La lecture (saisie) d'un tableau, c'est le remplir, ceci nécessite l'utilisation d'une boucle. Pour un tableau en général, il est préférable d'utiliser une boucle « pour » qui porte sur l'indice.

Syntaxe Langage algorithmique	Implémentation en C

Pour i de 1 à 10 faire
Ecrire ("donnez la valeur de la case [",i, "]") ;
 	
Lire (T[i]) ;
Fin faire
	for (i=0; i<10; i++)
	{
	printf ("donnez la valeur de la case [%d] : ", i);
	Scanf ("%d", &T[i]);
	}

Lire (T[1]) →c'est introduire par le clavier la valeur **20**

Lire (T[2]) →c'est introduire par le clavier la valeur **30**

……

Lire (T[10]) →c'est introduire par le clavier la valeur **50**

T [1] : première valeur du tableau d'indice i=1

20	**30**	**40**	**90**	**60**	**70**	**80**	**99**	**100**	**50**
1	2	3	4	5	6	7	8	9	10

En Mémoire centrale, les adresses des cases du tableau se suivent (du moins pour les caractères ou entiers).

	Adresse mémoire	valeur
T[0]	00001100	10
T[1]	00001101	20
	00001010	30
	00001011	40
	…….	

- **Ecriture :**

L'écriture signifie l'affichage des valeurs qui ont été introduites à l'écran

En Algorithmique	Implémentation en C
Pour i de 1 à 10 faire Ecrire (« la case »,i, « contient la valeur », T[i]) Fin pour	for (i=0; i<10; i++) printf ("la case %d contient la valeur %d ", i, T [i]);

L'affichage à l'écran donnera (en exécution C)
La case 0 contient la valeur 20→T[0]
La case 1 contient la valeur 30→T[1]
La case 2 contient la valeur 40→T[2]
……

***Remarquer** l'obligation de '<' et non '<=' car l'indice commence à 0.*

4.3 Les Matrices ou tableaux à deux dimensions

Un calendrier, une table de multiplication, un bulletin de notes sont des exemples de tableaux à deux dimensions. Vous avez peut être joué à la bataille navale ou utilisé Excel qui est un tableur. Le principe est le même, nous avons besoin de deux indices pour accéder à l'information, celle-ci se trouve à l'intersection d'une ligne et d'une colonne.

4.3.1 Définition

Une matrice est tableau à deux dimensions avec n lignes et m colonnes ; on parlera de matrice carrée si n=m.

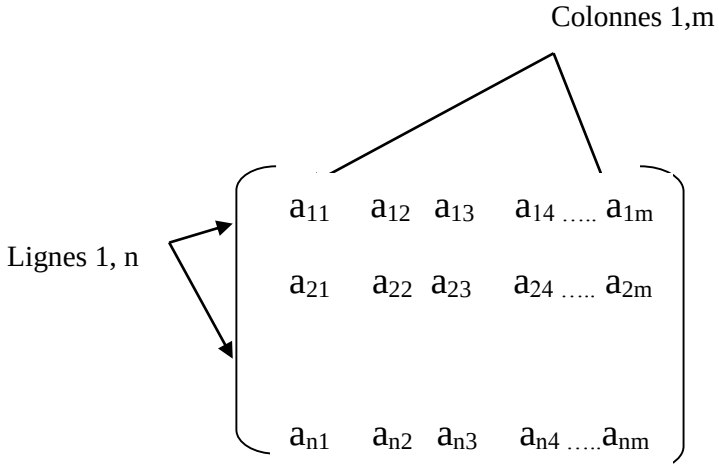


Figure 5.1 matrice

On note généralement aij l'élément qui se trouve à la ième ligne et jème colonne où 0<i< =n et 0<j < =m.

4.3.2 Déclaration d'une matrice

Un tableau à deux dimensions (matrice) est caractérisé par deux cardinalités (nombre de ligne) et (nombre de colonnes).

Dans la figure précédente n représentera le nombre de lignes, m le nombre de colonnes, *i* est l'indice pour les lignes et *j* l'indice pour les colonnes.

De la même façon, nous pouvons imaginer des tableaux à plusieurs dimensions (> 2), c'est le nombre d'indices utilisés qui déterminera la dimension de celle-ci.

Syntaxe 1

On déclare une matrice à deux dimensions de la façon suivante :

Type Nom_Matrice **= Tableau [***nombre_lignes, nombre_colonnes* **]** de *nom_type*
Variable nom Variable: Nom_Matrice ;

Exemple

Type Matrice =Tableau [10,12] de entier ;
Variable M : Matrice ;

Syntaxe 2

nom [nb_ligne, nb_colonne] : type d'éléments
Voir d'autres exemples ci contre

Langage algorithmique	En C
M[200,300] : Réel ;	Float M[200][300] ;
T [100,100] : Entier ;	int T [100][100] ;
Constante max=100; Tab[max,max]: réel;	#define max 100 Float Tab[max][max] ;

4.3.3 Accès aux éléments d'une matrice

Dans une matrice, l'élément est désigné par l'identifiant de la matrice et sa position représentée par deux indices (i indice pour les lignes et j indice pour les colonnes).

Identifiant [indice ligne, indice colonne]

M[i , j] : élément positionné à la ième ligne et jème colonne.

Exemple :

Soit la matrice M [4,6] avec 4 lignes et 6 colonnes contenant des entiers naturels.

	1	2	3	4	5	6
1	**12**	**30**	**20**	**90**	**100**	**62**
2	**5**	**5**	**3**	**4**	**5**	**20**
3	**200**	**45**	**89**	**52**	**5**	**10**
4	**90**	**12**	**1**	**3**	**2**	**12**

M[1,2] = 30 valeur à la 1ère ligne et 2ème colonne

4.3.4 Saisie et affichage d'une matrice (Lecture et écriture)

Le traitement de matrices nécessite d'utiliser des boucles imbriquées.

- **Lecture :**

Pour remplir une matrice on aura besoin de boucles imbriquées. Le remplissage se fait ici ligne par ligne.

```
Pour i de 1 à n  faire
    Pour j de 1 à m  faire
    Ecrire ("donnez la valeur de la case [",i, ,j, "]") ;
    Lire (M[i,j]) ;
    Fin pour ;
Fin pour ;
```

- **Ecriture**

Syntaxe

```
Pour i de 1 à n  faire
    Pour j de 1 à m  faire
       Ecrire ("la ligne",i, "et la colonne ",j, "contient la valeur", M[i,j]) ;
    Fin pour ;
Fin pour  // on peut écrire aussi par abus fpour, fait, ou fpr
```

Exemple :

Ecrire un algorithme qui calcule et affiche l'addition de deux matrices de réelles M1 et M2 de dimensions M et N.

```
Algorithme matrice
Constante N=100, M=100;
Variables  M1[N, M], M2[N, M], S[N, M] : réel;
           i, n, m : entier;
Début
Ecrire ("Veuillez saisir le nombre des lignes (≤ 100)");
lire (n);
Ecrire ("Veuillez saisir le nombre des colonnes (≤ 100)");
lire (m);
/*remplissage  de la matrice M1*/
pour i de 1 à n faire
    pour  j de 1 à m faire
      lire ( M1[i, j]);
    fin pour;
fin pour;
```

```
/* Remplissage de la matrice M2*/
pour i de 1 à n faire
        pour j de 1 à m faire
           lire( M2[i, j]);
        fin pour;
fin pour;
/* calcul de la matrice somme */
pour i de 1 à n faire
        pour j de 1 à m faire
           S[i, j] ← M1[i, j] + M2[i, j];
        fin pour;
fin pour;
/* affichage de la nouvelle matrice résultat*/
pour i de 1 à n faire
        pour j de 1 à m faire
          Ecrire ("s [" , i , " , " , j , "] : " , s [i, j] );
        fin pour;
fin pour;
Fin.
```

Remarque importante: dans un souci d'optimisation, nous pouvions réunir les partie calcul et affichage dans la même boucle imbriquée. Nous aurons alors la portion de code suivante :

```
/* calcul et affichage de la matrice somme */
pour i de 1 à n faire
        pour j de 1 à m faire
            S[i, j] ← M1[i, j] + M2[i, j];
            Ecrire ("s [" , i , " , " , j , "] : " , s [i, j] ); // instruction ajoutée
        fin pour;
fin pour;
```

4.4 Les Tableaux et chaines de caractères

4.4.1 Définition d'une chaine de caractères

Une chaîne de caractères est un tableau de caractères se terminant par le caractère spécial '\0' (qui a 0 pour code ASCII).

Nom

b	e	n	b	e	l	l	a	\0

Nom[9] :caractère ;

Généralement, les tableaux de type caractère sont initialisés une liste d'éléments du tableau fournis entre accolades.

Exemple :

Algorithme nom

Début
 nom[20] : caractère;
 nom[] = { 'I', 'G', 'M' ,'O' , '\0' } ; /* initialisation*/
Fin.

4.4.2 Traitement de chaînes de caractères

Fonctions fréquemment utilisés en C.

- **Strlen** (chaine) : fournit la longueur d'une chaine.
- **Strcpy** (chaine1,chaine2)**:** copie la chaine 2 vers la chaine 1.
- **Strcat** (chaine1,chaine2)**:** colle la chaine 2 à la fin de la chaine 1.
- **Strcmp** (<s>, <t>) compare les chaînes de caractères <s> et <t> de manière lexicographique .

Exemple 1 : utilisation de strcmp

```
#include<stdio.h>
#include<string.h>
int main()
{
char chaine1[20], chaine2[20];
printf(" Entrez la 1ère chaine ");
gets(chaine1);
printf(" Entrez la 2ème chaine ");
gets (chaine2);
 if (strcmp(chaine1,chaine2) == 0)
printf(" Les deux chaînes sont égales.\n");
else
printf(" Les deux chaînes ne sont pas égales.\n");
return 0;
}
```

Exemple 2 : utilisation de strcpy

```
#include<stdio.h>
#include<string.h>
main()
{
char    chaine1[10]="etudiant";
char    chaine2[10]="Licence";
strcpy(chaine1,chaine2);
printf("%s\n%s\n",chaine1,chaine2);
}
```

Exemple 3 : utilisation de strlen

```
#include<stdio.h>
#include<string.h>
int main () {
charch [] = "licence ";
printf("La longueur de %s est : %d",ch,strlen(ch));
  }
```

4.4.3 Tableaux de chaines

On peut définir des tableaux de caractères à plusieurs dimensions qui peuvent contenir des chaines de caractères.

Exemple :

Algorithme jour
Début
jour [7,9] : caractères ;
jour [7, 9] = {”dimanche”, “lundi” , ”mardi” , ”mercredi” , ”jeudi” , ”vendredi”, ”samedi”, };
Ecrire (“Aujourd’hui c’est : ”, jour[2]);
Fin
Soit la matrice jour[7,9]

	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9
i=1	**d**	**I**	**m**	**a**	**n**	**c**	**h**	**e**	**\0**
i=2	**l**	**u**	**n**	**d**	**i**	**\0**			
i=3	**m**	**a**	**r**	**d**	**i**	**\0**			
i=4	**m**	**e**	**r**	**c**	**r**	**e**	**d**	**i**	**\0**
i=5	**j**	**e**	**u**	**d**	**i**	**\0**			
i=6	**v**	**e**	**n**	**d**	**r**	**e**	**d**	**i**	**\0**
	s	**a**	**m**	**e**	**d**	**i**	**\0**		

Enfin, on dit qu'un tableau est trié si ses éléments sont classés par ordre croissant (ou décroissant) ; cela facilite la recherche d'un élément.

Implémentation en C

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

{
 char JOUR[7][9]= {"dimanche"}, "lundi", "mardi",
"mercredi", "jeudi",  "vendredi", "samedi"}
int i = 2;
printf ("Aujourd'hui, c'est %s !\n", JOUR[i]);
}
```

Les tableaux et matrices sont très utilisés en calcul numérique. Par exemple en exercice vous pouvez calculer la valeur d'un polynôme de degré n en représentant ses coefficients dans un tableau de même dimension. Des langages comme Matlab possèdent des fonctions intégrées qui manipulent ces structures de données (produit scalaire, produit matriciel, inverse de matrice …)

Remarques 

Les tableaux de chaînes sont mémorisés ligne par ligne. La variable JOUR aura donc besoin de 7*9*1 = 63 octets en mémoire. Il est possible d'accéder aux différentes chaînes de caractères d'un tableau, en indiquant simplement la ligne correspondante.

Soit float t[]= {.6, .8, 9.8 }, on aura 3 flottants sachant que la taille d'un float simple en C est de 4 octets ce qui donne 3*4 octets .

Exemple : 0x10, 0x11,0x12 et 0x13 pour T[0] ;

0x14, 0x15,0x16 et 0x17 pour t[1] ,

et 0x18, 0x19, 0x20, 0x21 pour t[2].

Le programme n'aura pas à se soucier de ce problème c'est le compilateur qui prendra les 4 cases à partir de l'adresse 0x14.

.

4.5 Enoncés des exercices d'application

Exercice 5.1

Soit un tableau de n cases entières, écrire un algorithme qui somme les valeurs positives et négatives de ce tableau.

Exercice 5.2

Ecrire un algorithme qui affiche l'indice de la première occurrence d'une valeur x si elle existe sinon il affiche -1.

Exercice 5.3

Ecrire un algorithme qui tri un tableau par ordre croissant.

Exercice 5.4

Ecrire un programme C qui inverse les éléments d'un tableau**.**

Option (non corrigé) permuter les éléments des deux diagonales en utilisant un seul indice**.**

Exercice 5.5

Ecrire un programme en C qui sépare un tableau T en deux tableaux contenant respectivement les éléments positifs et négatifs de T.

Exercice 5.6

Ecrire un programme C qui déclare une matrice, saisit les éléments de la matrice et additionne les éléments de la matrice**.**

Exercice 5.7

Ecrire un programme C qui met à zéro les éléments de la diagonale principale d'une matrice carrée.

Exercice 5.8

Ecrire un algorithme qui fait la fusion de deux tableaux d'entiers triés par ordre croissant

Exercice 5.9

Soit une matrice M (nxm) de réels. On voudra créer un tableau T de n éléments où chaque élément est la somme des éléments d'une ligne correspondante de la matrice : T[i] est la somme de la ligne i, pour i allant de 1 à n. (Faire de même pour les colonnes dans un autre tableau P).

Corrigés

5.1

```
Algorithme tableau
T [50] : entier ;
Variables i, n, SP, SN :entier ;
Debut
  Ecrire (“donnez la dimension du tableau<50“) ;
  Lire (n) ;
  SP←0 ;
  SN←0 ;
  Pour i de 1 à n faire
     Ecrire (“donnez la valeur de la case [“,i, “]“)
     Lire (T [i]) ;
          Si (T [i]>=0)     alors SP←SP+T [i] ;
                            Sinon SN←SN+T [i] ;
     Ecrire (“La somme des valeurs positives est “, SP, “et la somme des valeurs
     négatives est “, SN ) ; // en fait les valeurs nulles seront ajoutées avec les valeurs négatives
   fpour ;
Fin.
```

5.2

```
Algorithme recherche
Variables N, i, X, V[100]: entier
Début
Ecrire (‘donnez la dimension du tableau<100’) ;
Lire (N) ;
Pour i de 1 à N faire
   Lire (V[i]) ;
Fin pour
Lire (X) ; /* saisie par le clavier de la valeur à chercher */
  i←0 ;
Répéter
 i=i+1 ;
 jusqu'à  i>N ou V[i]=X
 /* on sort de la boucle soit la valeur a été trouvée soit le tableau a été parcouru
jusqu’à la fin */
Si  i=N+1 ;
  Alors  ecrire (-1) ;   /* élément non trouvé */
```

Sinon ecrire (i) ; /* élément trouvé à la position i */
Fin si
Fin.

***Astuce** : ici la boucle répéter ou tant que est mieux indiquée ; elle est utilisée pour arrêter la boucle dès que l'élément est trouvé.*

5.3

Algorithme tri
T [100] :réel ;
variables N ,i,j: entiers ;
variable temp: réel ;
Debut
Ecrire ('donnez la dimension du tableau<100') ;
Lire (N) ;
Pour i de 1 à N-1 Faire
Pour j de i+1 à N Faire
Si T[i] > T[j] alors
temp← T[i] ;
T[i]← T[j] ;
T[j] ← temp ;
Fin si ;
Fin pour;
Fin pour;
Fin.

5.4

```
#include<stdio.h>
main()
{
 int T[50];
 int N;
 int I,J;   /* indices courants */
 int  tmp; /* variable temporaire pour l'échange   */
 /* Saisie des données */
printf ("Donnez la dimension du tableau<50: ");
scanf("%d", &N );
 for (I=0; I<N; I++)
    {
    printf("%d : ", I);
    scanf("%d", &T[I]);}
```

```
 /* Affichage du tableau */
printf("affichage : \n");
 for (I=0; I<N; I++)
   printf("%d ", T[I]);
   printf("\n");
 /* Inverser le tableau */
for (I=0, J=N-1 ; I<J ; I++,J--)
/* Echange de T[I] et T[J] */
        {
  tmp = T[I];
  T[I] = T[J];
  T[J] = tmp;
        }
  /* affichage  */
printf("Tableau résultat :\n");
 for (I=0; I<N; I++)
  printf("%d ", T[I]);
 printf("\n");
 return 0;
}
```

Astuce *: on pouvait aussi se passer de la variable j et faire l'échange de T[i] avec T[N-i+1].*

5 .5

```
#include<stdio.h>
main()
{
int T[50], P [50], N [50];
int N, NP,  NN;
int I ;
 /* Saisie des données */
printf("Dimension du tableau (<50) : ");
scanf("%d", &N );
for (I=0; I<N; I++)
    {
    printf("%d : ", I);
    scanf("%d", &T[I]);
    }
```

```
 /* Affichage du tableau */
printf("Tableau donné :\n");
for (I=0; I<N; I++)
        printf("%d ", T[I]);
printf("\n");
  NP=0;
  NN=0;
 /* Transfer des données à partir de T*/
for (I=0; I<N; I++)
      { if (T[I]>0) {
                        P [NP]=T[I];
                        NP++;
                              }
        if (T[I]<0) {
                       N [NN]=T[I];
                       NN++;
                           }
      }//remarquer que les valeurs nulles sont ignorées
return 0;
}
```

5.6

```
#include<stdio.h>
main()
{
 /* Déclarations */
Int M[50][50];
int L, C;  /* dimensions    de la matrice    */
int I, J;  /* indices lignes et colonnes */
 long SOM;  /* somme des éléments */
 /* Saisie des données */
printf("Nombre de lignes   (<50) : ");
scanf("%d", &L );
printf("Nombre de colonnes (<50) : ");
scanf("%d", &C );
 for (I=0; I<L; I++)
    for (J=0; J<C; J++)
        {
printf("[%d][%d] : ",I,J);
```

```
scanf("%d", &M[I][J]);
        }
  /* Affichage du tableau */
printf("Tableau donné :\n");
for (I=0; I<L; I++)
    {
for (J=0; J<C; J++)
printf("%7d", M[I][J]);
printf("\n");
    }
  /* Calcul de la somme  */
 for (SOM=0, I=0; I<L; I++)
     for (J=0; J<C; J++)
         SOM += M[I][J];
  /* affichage de la somme */
printf("Somme des éléments : %ld\n", SOM);
 return 0;
}
```

5.7

```
#include<stdio.h>
main()
{ int  M[100][100];
  int  N;
  int I, J;
  printf("donnez la dimension : ");
  scanf("%d", &N);
  for (I=0; I<N; I++)
    for (J=0; J<N; J++)
    { printf("[%d][%d] : ",I,J);
      scanf("%d", &M[I][J]);
    }
  printf("affichage :\n");
  for (I=0; I<N; I++)
```

```
 { for (J=0; J<N; J++)
   printf("%7d", M[I][J]);
   printf("\n"); }
  for (I=0; I<N; I++)
   M[I][I]=0;
   printf("affichage :\n");
  for (I=0; I<N; I++)
  {  for (J=0; J<N; J++)
     printf("%7d", M[I][J]);
     printf("\n"); // on écrit new line fin de ligne
  }
 return (0);
}
```

5.8

L'hypothèse de tableaux triés est importante pour réaliser la fusion autrement cela aurait été impossible. La méthode consiste à piocher les éléments dans le tableau qui contient les plus petits éléments puis d'aller sur l'autre tableau et ainsi de suite. Le tableau T résultat contiendra la fusion des deux de taille N1 et N2, **l**es indices i, i1, et i2 pour les trois tableaux respectivement.

Début

```
 i1 = 0; i2 = 0; i = 0
tant que i1 < N1 et i2 < N2 faire
  si T1[i1] < T2[i2] alors
     T[i] = T1[i1] ; i++; i1++
  sinon
     T[i] = T2[i2]; i++; i2++
  Finsi ;
fin tant que
si i1 < N1 alors  // fin du premier tableau, on prend tous les éléments du 2ième
   tant que i1 < N1 faire
      T[i] = T1[i1]; i++; i1++
   fin tant que
sinon
    tant que i2 < N2 faire
       T[i] = T2[i2]; i++; i2++
    fin tant que
 fin si
```

Fin**.**

5.9

```
Algorithme calcul_somme_ligne
Const M=20, M=10 ;
Variables M[N,M], T[N], i, j : entier ;
 Debut
 Pour i = 1 à N faire
      T[i]= 0 ;
       Pour j= 1 à M faire
       T[i] = T[i] + M[i,j] ;
       Fpour ;
 Fpour ;
Pour i= 1 à N     //affichage du résultat
   Ecrire (‘T[‘, i, ‘]= ‘, T[i]) ;
Fpour ;
Fin.
```

Remarque : pour le vecteur P, faire la boucle sur j au lieu de i.

5 Les Enregistrements

5.1 Introduction

Les types vus jusqu'à présents sont dits standards car ils sont offerts dans la librairie du langage de programmation. Qu'en est-il si on veut créer son propre type ? Ce nouveau type est structuré, et composé alors de types standards ou de types déjà définis par l'utilisateur. On les appelle aussi types personnalisés.

Ce type abstrait décrit en général en entité (telle que vue par les systèmes d'information). Un étudiant par exemple est caractérisé par son nom, prénom, âge, Num_inscription, …Par opposition aux tableaux, les composants des enregistrements ne sont pas nécessairement de même type. Ces éléments qui composent un enregistrement sont appelés **champs**. On retrouve cette notion aussi en bases de données, un enregistrement est une ligne de la table, et les champs sont les attributs. Les enregistrements sont appelés structures, en analogie avec le langage C. Plus tard, ils seront à la base de la construction des fichiers.

Quant à la structuration des données, XML est un bon exemple pour voir la hiérarchisation dans la description des données. On doit comprendre aussi qu'un fichier de données est aussi un moyen de communication entre deux programmes, où le premier programme génère des données qui seront exploitées par le second.

5.2 Enumérations

Ce type a déjà été abordé au chapitre deux avec les types standards. Il permet d'éviter les erreurs de saisie, notamment d'une chaine de caractères. Une énumération est un ensemble statique de valeurs prédéfinies et ordonnées.
Nous écrivons :
Enum chiffre ={ zero =0, un, deux, trois, quatre, cinq, six sept, huit, neuf}
Dans ce cas les autres, constantes d'énumération prendront automatiquement les valeurs, 1, 2, …etc.
En langage algorithmique, on considère l'introduction d'un nouveau type
Type e**num** {Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi} semaine;
Puis son utilisation en déclarant une variable de ce type (ici jour).
semaine jour;
Nous pouvons utiliser classiquement cette variable dans une instruction switch pour voir ce que l'on fait chaque jour de la semaine

```
Switch (jour)
 { case dimanche : ….. ;
   Case lundi : …
  }
```

Nous verrons son utilisation dans la section suivante, celle des structures.

5.3 Déclaration d'un enregistrement

En langage algorithmique	En C

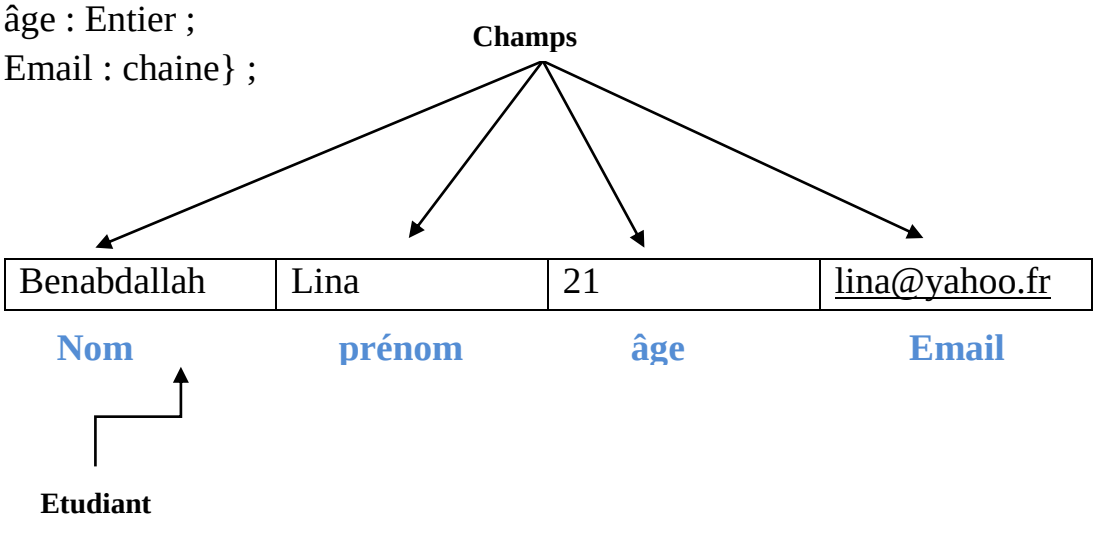
Type point=structure
{
X :réel;
Y :réel;
}
	Typedef struct
	{
	float X;
	float Y;
	} **point**;

Contrairement aux tableaux où tous les éléments sont de même type, les enregistrements regroupent des données pouvant être de types différents.
Un enregistrement est un ensemble de paires (nom_champ, type_champ) qui regroupe les données relatives à une même entité ou structure.
Déclarer un enregistrement nécessite de :

- Définir son nom.
- Définir les champs et leur type.

Langage algorithmique	En C
Type *nom_type*= **Structure {** nom_champ1: type_champ1 ; … nom_champ N:type_champ N ; }	struct<Nom_Structure> {<type_champ1><Nom_Champ1>; <type_champ N><Nom_Champ N>; };

Exemple 1
Type Etudiant = **structure** { Nom : chaîne ;
prénom : chaîne ;
âge : Entier ;
Email : chaine} ;



Exemple 2

En langage algorithmique	En C

Type semaine = **Enum** {Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi} ;
Type date = Structure {
j_semaine : semaine ;
j_mois : entier;
mois : entier;
année : entier; };
Algorithme
Début
D1, D2 : date;
Fin
	Typedef enum {Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi} semaine ;
	Typedef struct{
	semaine j_semaine ;
	int j_mois;
	int mois;
	int année; } date;
	main ()
	{
	Date D1, D2 ;
	}

Remarque : tous les langages de programmation offrent des fonctions prédéfinies pour avoir l'horloge du système. Plusieurs formats existent : time(), datetime.now(), …pour accéder aux informations sur la date et l'heure.

Exemple 4

Type fournisseur = **Structure** {Nom : chaîne ;

prénom : chaîne ;

adresse : chaine ;

Email :chaine ;

Tel : chaine ;}

5.3.2 Accès à un champ d'enregistrement

On accède à un champ d'enregistrement par un sélecteur de champ.

L'écriture syntaxique du sélecteur reflète la hiérarchie de la structure.

Le sélecteur d'un champ élémentaire de type simple **champ** d'un enregistrement nommé **Enregistrement** est noté : **Enregistrement.champ (**Le point indique le chemin d'accès) par Exemple : « point**.**X « *sélecteur du champ* X dans la structure point de l'exemple précédent.

Des champs appartenant à des structures différentes, peuvent être homonymes c'est-à-dire, avoir le même identificateur. Par exemple **etudiant.prenom** de l'exemple 1 et **fournisseur.prenom** de l'exemple 4 aucune confusion n'est possible entre les sélecteurs de champs. Dans certains langages, le sélecteur est une flèche → (en C par exemple). Nous retrouvons aussi le sélecteur dans les langages objet, c'est un moyen d'accéder aux propriétés ou méthodes de l'objet, c'est aussi un facilitateur lors de l'écriture du programme, le compilateur nous affiche la liste des champs (ou propriétés) disponibles une fois le sélecteur saisi.

5.3.3 Structures imbriquées

Un enregistrement est un type défini par le programmeur, il est alors vu comme un type personnalisé, qui peut être à son tour utilisé dans la définition d'autres types plus complexes.

Un enregistrement peut être imbriqué dans une autre structure, les champs d'un enregistrement peuvent être de type enregistrement.

```
Type date = structure
  {
  j :entier ;
  m :entier ;
  a : entier ;
  }
Type enseignant= structure
 {
  Nom : chaîne ;
  Prénom : chaîne ;
  date_naiss : date ;
  module :chaine ;
  }
```

5.3.4 Cas d'un tableau comme champ de structure

Dans une structure on peut avoir des champs de type tableau

Exemple 1

```
Type Etudiant = structure{
Nom : chaîne ;
Prénom : chaîne ;
date_naiss : date ;
Notes [10] :réel }
```

E : Etudiant **;**

Nom	Prénom	date naiss	Notes
			9 (1) | 10 (2) | 8 (3) | 12.5 (4) | 15 (5) | 7 (6) | 4 (7) | 9.5 (8) | 11 (9) | 13 (10)

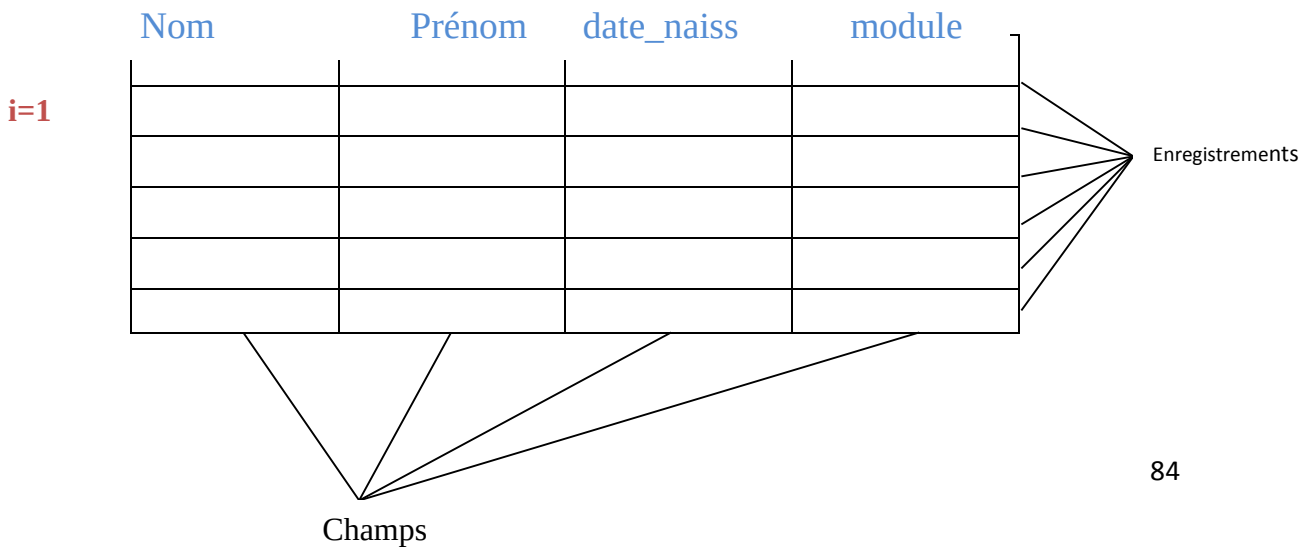
E.Notes [2] =10. La valeur 10 est contenue dans la 2ème case du tableau « Notes »qui représente le 4ème champ de la structure **Etudiant**.
Ainsi si on veut accéder au mois de naissance d'un étudiant, on utilise le sélecteur ('.') de manière hiérarchique : E.date_naiss.m.

5.3.5 Cas de tableaux d'enregistrements (ou tables)

C'est le cas le plus fréquent. En effet, un enregistrement étant un nouveau type, on peut en regrouper plusieurs de ce type. Il arrive que l'on veuille traiter non pas un seul enregistrement mais plusieurs, donc on aura un tableau contenant des enregistrements. Chaque élément du tableau est alors de type structure. Remarquer que le tableau d'enregistrements est stocké en mémoire par opposition aux enregistrements manipulés par un fichier.

Exemple

Type enseignant= structure**{**
Nom : chaîne ;
Prénom : chaîne ;
date_naiss : date ;
module :chaine ;**}**
Corps enseignant [7] : enseignant ;
Chaque élément du tableau « **Corps enseignant «** est un enregistrement,. On accède à un enregistrement par son indice dans le tableau.
Corps enseignant[2] représente le 2ème enseignant du **Corps enseignant**



5.4 Autres possibilité de définition de type :

Le langage C fournit des types supplémentaires qui permettent de s'abstraire des types qui dépendent des caractéristiques techniques de l'ordinateur cible, de la taille réservée aux données, …etc. Ils sont appelés types abstraits de données. Grâce à cette notion de bibliothèque où il est possible de définir de nouveaux types qui seront inclus dans le programme.

Le langage C++ aujourd'hui, compatible avec le C permet d'introduire le paradigme objet sans perdre de la puissance du C.

Il est à noter que les types présentés ici ne sont pas les seuls disponibles dans les langages. Chaque langage présente ses propres types mais la plupart manipulent les types standards. Certains langages offrent des types qui facilitent les manipulations de données. Ainsi le type intervalle qui précise les bornes inférieures et supérieures ; Nous en donnons quelques-uns en python :

Le type ensemble set A = set ([‘vert’, ‘blanc’, ‘rouge’]).

Des fonctions prédéfinies permettent de savoir si un élément appartient à une liste (‘in’), de même que des opérations ensemblistes peuvent être utilisées (union, intersection, différence, ..)

Le type liste qui est vu comme séquence dans laquelle on rassemble plusieurs éléments de données dans une même unité. Comme en prolog, on reconnaitra la liste par les [ ]. Ces crochets sont réservés aux tableaux dans les langages C et java. En Prolog, il n'y a pas de déclaration explicite de liste, le fait d'écrire les [ ] signifie automatiquement qu'il s'agit d'une liste.

Liste A = [1, 2, 3, 4, 5].

On pourra tester si un élément est dans une liste par la fonction ‘in’

De manière plus hiérarchisée, les tuples peuvent représenter des données complexes (à l'image de Xml)

Un_tuple = (a,b, c, (d, e), f (g, (h, i, j, k)))

Les dictionnaires sont aussi une structure de données intéressante formés d'une liste de paires clé/valeur (comme un mot et sa définition). On peut par la suite rechercher une valeur particulière par le nom et un index.

Mon_dico = {‘vert’ :1, ‘blanc’ :2, ‘rouge’ :3}

Donc mon_dico[‘blanc’] donnera la valeur 2.

Nous verrons dans les chapitres suivants les types structurés tableaux et enregistrements qui permettent de regrouper plusieurs données de même ou différents types respectivement.

5.5 Enoncés des exercices d'application

Exercice 6.1

Une carte grise contient les informations sur un véhicule :
Nom et prénom du propriétaire, adresse, N° matricule, marque, type, puissance, année, couleur.
Le matricule (en Algérie) est un code contenant quatre informations un numéro de série, le type, l'année, et la wilaya.

1) Donner la déclaration de la structure carte grise.
2) Créer un tableau de 100 véhicules pour remplir les champs déclarés.
3) Ecrire un algorithme qui compte le nombre de véhicules de la wilaya 31 et affiche ce résultat.

Exercice 6.2

Ecrire un programme en C qui calcule la distance entre 2 points (utiliser la fonction prédéfinie **SQRT** de calcul de la racine carrée d'un nombre réel).

Exercice 6.3

Un étudiant est caractérisé par les informations suivantes :

- Nom et prénom
- Date de naissance
- Filière
- Matricule
- Moyenne

Soit T un tableau d'au plus 100 étudiants. Ecrire un algorithme permettant d'afficher tous les étudiants admis. Un étudiant est admis si sa moyenne est supérieure ou égale 10. Calculer et afficher la moyenne de tous les étudiants.

Exercice 6.4

1- Déclarer des types qui permettent de stocker :
   - Un joueur de basket caractérisé par son nom, sa date de naissance, sa nationalité et sa taille.
   - Une équipe de 20 joueurs.
   - Une association comprenant le nom de l'association, l'équipe des joueurs et les points marqués par l'équipe.
2- Ecrire un algorithme qui permet de saisir les données des 20 joueurs.
3- Ecrire un algorithme qui compte le nombre de joueurs non algériens.

Exercice 6.5

Un client dans une banque est identifié par son nom, prénom, adresse, numéro de compte, e-mail, téléphone et le solde de son compte.

Ecrire un algorithme qui permet de faire la somme des soldes des clients de la banque et de calculer le solde moyen par client.

Exercice 6.6 non corrigé

On désire représenter une liste de vols (comme un tableau d'affichage d'un aéroport). Chaque vol contient les informations sur la compagnie, le Num_vol, destination, heure_départ, observation. On désire créer une autre liste des vols à destination d'Alger.

Donner la structure de données adéquate et l'algorithme correspondant. Considérer une solution avec un tableau d'enregistrements

Exercice 6.7 non corrigé

On s'intéresse à la gestion des véhicules d'un parc auto. Chaque véhicule est caractérisé par

- Une marque,
- une couleur,
- un modèle,
- le nombre de places
- une puissance
- un matricule,

Le matricule est composé de

- un numéro,
- l'année
- le numéro de la wilaya

1- Déclarer la structure « véhicule » contenant les informations ci-dessus
2- Déclarer un tableau T contenant N($\leq$ 500) véhicules
3- Ecrire une fonction qui compte le nombre de véhicule de marque « Renaut « et matriculé dans la wilaya 48.
4- Ecrire une procédure qui affiche la marque de voiture qui a la plus grande puissance parmi tous les véhicules stockés dans le tableau T.

Corrigés

6.1

TYPE matricule = Structure
{num: entier ; type : entier ; annee : entier, wilaya : entier};
TYPE carte grise = Structure
{nom : chaine; prenom : chaine; mat : matricule; adresse : chaine; marque: chaine; type : chaine; puissance : chaine; année :entier ;couleur : chaine; };
T [100] : carte grise ;
Algorithme calcul_wilaya
Debut
  Variables i, n :entier ;
  Pour i de 1 à 100 Faire
    Lire(T[i].nom , T[i].prenom , T[i].mat, T[i].adresse,T[i].marque,
    T[i].type, T[i].puissance,T[i].annee,T[i].couleur);
  Fin pour
  n=0 ;
  Pour i de 1 à 100Faire
      Si T[i].mat.wilaya =31
      alors n=n+ 1 ;
      Fin si ;
   Fin pour;
  Ecrire(‘ le nombre de véhicules de matricule 31 est :’,n) ;
Fin

6.2

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
Typedef struct
{
float X;
float Y;
} TPoint;
main()
{
  TPoint P1,P2;
  Float D;
```

```
printf("Donner l abscisse du premier point\n");
scanf("%f",&P1.X);
printf("Donner l ordonnee du premier point\n");
scanf("%f",&P1.Y);
printf("Donner l abscisse du second point\n");
scanf("%f",&P2.X);
printf("Donner l ordonnee du second point\n");
scanf("%f",&P2.Y);
D   =   sqrt((P1.X-P2.X)*(P1.X-P2.X)+(P1.Y-P2.Y)*(P1.Y-
 P2.Y));
printf("La distance est %.2f\n",D);
}
```

6.3

TYPE Date = Structure
{j : entier ; m : entier ; an : entier;};
Type Etudiant = structure
{ Nom, Prenom, filiere : chaine ;
Matricule : entier ;
Moyenne : réel ;
ddn : Date ;}
Algorithme Etudiant ;
T [100] : Etudiant ;
Variables i,N :entier ;
Variables M, S: réel;
Début
Ecrire (‘Donner le nombre d’’etudiants’) ;
Repeter
Lire(N) ;
Jusqu’à N>0 et N≤100 ;
Pour i de1 à N Faire
Lire(T[i].nom) ;
Lire(T[i].prenom) ;
Lire(T[i].filiere) ;
Lire(T[i].Matricule) ;
Lire (T[i].Moyenne;
Lire(T[i].ddn.j , T[i].ddn.m, T[i].ddn.an) ;
Fin pour ;

```
S←0 ;
  Pour i de 1 à N Faire
     Si T[i].Moyenne)≥10
     Alors Ecrire(T[i].nom) ;
     Fin si;
  Fin pour ;
  Pour i de 1 à N faire
     S=S+T[i].moyenne ;
  Fin pour ;
  M←S/N ;
  Ecrire (‘la moyenne des étudiants est :’,M) ;
```
Fin.

6.4

```
Type date= structure
{ j: entier ;
m: entier ;
a : entier ;
}
```

```
Type  joueur = structure
{
  nom, chaine ;
  dn: date ;
  nat :chaine ;
  taille :reél ;
}
E [20] : joueur ;
Type association = structure
{
  nom, chaine ;
  points :entier ;
  equipe : E [20] ;
}
Algorithme saisie
Variable i :  entier ;
debut
pour i de 1 à 20 faire
Lire (E[i].nom, E[i].nat, E[i].taille, E[i].dn.j, E[i].dn.m, E[i].dn.a);
fpour
fin
Algorithme compte
Variables c, i :entier ;
debut
pour i de 1 à n faire
si (E[i].nat≠'algerienne') alors
c←c+1 ;
finsi ;
finpour ;
Ecrire ('le nombre est :'c)
Fin.
```

6.5

```
Type client=structure
{  nom : chaine ;
Prénom : chaine ;
Adresse : chaine ;
NC : chaine ;
```

e: chaine ;
tel : chaine ;
S : réel }
B[200] :client ;
Algorithme banque
Variable n : entier ;
Variable ST,M :réel ;
Ecrire (‘donnez le nombre de clients :’) ;
Lire (n) ;
Pour i de 1 à n faire
Lire (B[i].nom, B[i]. Prénom, B[i]. Adresse, B[i].NC, B[i].e, B[i].tel, B[i].S) ;
Finpour ;
ST←0.0 ;
Pour i de 1 à n faire
ST ← ST+ E[i].S ;
Finpour;
Ecrire (‘le solde total est :’,ST)
M←ST/n ;
Ecrire (‘la moyenne du solde par client est :’,M)
Finpour ;
Fin.

Conclusion Générale

Tout est bien qui finit bien !

Donner une introduction à l'algorithmique via l'informatique est un exercice pédagogique qui n'est pas aisé. Nous avons donné un bref historique de l'évolution fulgurante de l'informatique avant d'introduire les bases de l'algorithmique.

Nous avons insisté sur la méthodologie à suivre pour construire un algorithme partant de la spécification du problème.

Par opposition aux opérations d'installation d'un produit ou de montage d'un meuble où on connait à priori à quoi on veut arriver, l'algorithme est présenté comme un ensemble d'étapes à suivre pour résoudre un problème pratique, et où la conception des instructions à suivre incombe au programmeur. La maitrise des structures algorithmiques de base est acquise à ce niveau (les itérations, les alternatives, …) ainsi que les structures de données utilisées.

Ce dernier doit raisonner par objectif. Cette approche est dite téléologique ou modèle de l'ours blanc qui a pour objectif de survire dans un environnement hostile. On connait les données, ou les inputs ; on connait le problème mais on ne sait pas comment faire.

Et là le génie du programmeur intervient, comme celui d'el Khawarizmi en indiquant les étapes à suivre pour résoudre une équation. Nous avons vu que ces instructions prenaient différentes formes, allant de l'itération simple aux boucles imbriquées, en passant par les alternatives et les boucles 'Pour'. La structure du programme est importante pour pouvoir suivre les étapes de son exécution. Souvent, au début de la programmation, on ne comprend pas pourquoi un programme ne marche alors qu'on a écrit tout correctement. Les erreurs sémantiques sont plus difficiles à détecter que les erreurs syntaxiques.

Quant à l'aspect structure de données, l'essence de construire des données complexes comme les enregistrements à partir des données simples ou types standards a été expliqué par des exemples et des exercices d'application. Le choix d'un tableau, matrice ou enregistrement pour la résolution d'un problème fait partie de la conception globale de la solution.

Nous avons vu ensuite à travers quelques exemples, l'aspect optimisation de l'algorithme, afin que celui-ci donne une meilleure solution plus rapide ou nécessitant moins de données ou moins d'espace. Cet aspect sera développé en Algorithmique avancée, notamment avec les algorithmes de tri.

Enfin, Comme l'histoire de l'œuf et la poule, on ne peut pas dissocier l'algorithme des données sur lesquelles il travaille. Par quoi commencer ?

Bibliographie

1- Nicolas Flasque, Helen Kassel , Franck Le poivre , Boris Velikson - *EXERCICES ET PROBLÈMES D'ALGORITHMIQUE-*
édition Dunod, Paris, 2010 .

2- Jean Paul Muller & Luca Massaron, *« Les algorithmes »* , First Interactive Wiley Publishing Inc, edition 2021

3- Claude Delannoy. *Programmer en langage C.* 5ième édition Brochet Eyrolles 2016.

4- Rémy Malgouyres , Rita Zrour , Fabien Feschet- *INITIATION À L'ALGORITHMIQUE ET À LA PROGRAMMATION EN C* Cours avec 129 exercices corrigés -2eme édition. Paris Dunod 2014

5- Thomas H.Cormen Charles E.Leiserson& Ronald L.Rivest.
Introduction to Algorithms third edition. MIT Press, Cambridge, Masasuchetts, London 2009

6- F. FABER & J.F *Programmation en C,*. ANNE 1998/1999

7- Brian W.Kernigham, Denis M.Richie *The C programming langage* .
Printice hall Software Series 1999.

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

9- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest Algorithmique - 3ème édition - Cours avec 957 exercices et 158 problèmes Broché, Dunod, 2010.

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

11- Damien Berthet et Vincent Labatut. *Algorithmique & programmation en langage C - vol.1 : Supports de cours*. Licence. Algorithmique et Programmation, Istanbul, Turquie. 2014, pp.232.

12- Damien Berthet et Vincent Labatut. *Algorithmique & programmation en langage C - vol.2 : Sujets de travaux pratiques*. Licence. Algorithmique et Programmation, Istanbul, Turquie. 2014, pp.258. <cel-01176120>

13- Damien Berthet et Vincent Labatut. *Algorithmique & programmation en langage C - vol.3 : Corrigés de travaux pratiques*. Licence. Algorithmique et Programmation, Istanbul, Turquie. 2014, pp.217. <cel-01176121>

14- Claude Delannoy. *Apprendre à programmer en Turbo C*. Chihab-EYROLLES, 1994.