REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Ben M'hidi- Oum El Bouaghi-
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département de Mathématiques et Informatique

Matière

L'algorithmique et Structure de données1

Cours / TDs / TPs

Niveau : 1ère année MI

Proposé par
Dr. SID Karima

Version : 2.0

Année universitaire : 2021-2022

Table des matières

Préambule

Le présent document s'agit d'un support de cours, portant des TDs et des TPs, proposé pour apprendre la matière intitulée « **Algorithmique et Structure de Données 1**», enseignée en premier semestre, au Département de Mathématiques et d'Informatique à l'université d'Oum El Bouaghi, et qui est destinée aux étudiants de la première année tronc commun MI.

D'abord, les objectifs pédagogiques de ce cours sont :

1. Comprendre et acquérir les notions d'un algorithme et d'une structure de données ;
2. Ecrire un algorithme simple et le traduire en programme C ;
3. Evoluer vers la maîtrise des structures conditionnelles et itératives afin d'écrire un algorithme plus structuré ;
4. Après la compréhension et la maîtrise des types simples, l'étudiant passera aux types plus avancés, qui sont les types structurés (tableaux et matrices) et personnalisés (énuméré et enregistrement).

En effet et dans l'optique d'achever les objectifs précités, nous avons mis des efforts prodigieux pour aborder ce travail dans plusieurs aspects ; où nous avons synthétisé les informations les plus importantes et pertinentes en s'appuyant sur des différentes sources documentaires (ouvrages, articles, cours, sites internet…). Tout en respectant le canevas officiel déterminé par le ministère de l'enseignement supérieur et de la recherche scientifique.

Par ailleurs, ce document est composé de six chapitres, le premier chapitre est introductif, suivant par cinq d'autres cours. En effet, chaque cours est soutenu par une série de TD et une autre de TP (en langage C) synthétisées selon un ensemble des objectifs spécifiques, et ce pour permettre à l'apprenant d'acquérir facilement des informations et des concepts de base nécessaires pour qu'il puisse écrire des algorithmes et les traduire en programme C afin de résoudre n'importe quel problème.

Cependant, le document en question restera partiel et non exhaustif. C'est la raison pour laquelle nous allons le mettre à jour constamment afin d'enrichir son contenu. En revanche, nous saurions reconnaissants aux lecteurs de nous faire parvenir de toute éventuelle erreur, observation, etc. Ainsi, de nous proposer des avis en ce sens.

Chapitre 1
« Introduction »

Chapitre 1 : Introduction

Au cours de ces dernières années, les logiciels ont envahi notre quotidien, quasiment tous les domaines reposent sur les programmes informatiques. Les algorithmes sont largement utilisés dans les différents domaines, pour effectuer une recherche sur le web, investir dans les meilleures actions, pour le dépistage des maladies (bioinformatique) ... avant d'entamer les concepts fondamentaux de l'algorithmique, nous allons présenter un bref historique ainsi les périodes clés de l'évolution de l'informatique.

1. Bref historique sur l'informatique

→ En 1613, la première utilisation du mot «ordinateur» a été enregistrée, faisant référence à une personne qui a effectuée des calculs, et le mot a continué d'être utilisé jusqu'au milieu du 20e siècle.

→ Un tally stick était un ancien dispositif d'aide à la mémorisation, pour enregistrer et documenter des nombres, des quantités ou même des messages.

→ Un appareil mécanique Abacus a été inventé en babylonie, il permet d'effectuer des opérations arithmétiques de base.

→ En 1820, une calculatrice mécanique inventée par Thomas de Colmarin, c'est la première machine de calcul fiable, utile et commercialement réussie.

→ En 1822, le premier ordinateur mécanique inventé par Charles Babbage.

→ En 1840, Augusta Ada Byron suggère à Babbage d'utiliser le système binaire (toute information doit représenter sous forme des 0 et 1).

→ En 1890, la machine de tabulation a été inventée par Herman Hollerithin pour résumer les informations et effectuer quelques opérations de comptabilités.

→ Entre 1936-1938, le premier ordinateur programmable, créé par Konrad Zuse en Allemagne.

→ En 1943, le premier ordinateur électromécanique également connu sous le nom d'IBM Automatic Sequence Controlled Calculator (ASCC) inventé par Howard H. Aikenin.

→ En 1952, le premier ordinateur avec des programmes stockés, conçu par Von Neumann. Il dispose d'une mémoire pour stocker à la fois le programme et les données.

→ En 1981, le premier ordinateur portable réalisé par l'Osborne Computer Corporation.

2. Introduction à l'algorithmique

Avant d'aborder les algorithmes, nous allons introduire quelques concepts de base pour comprendre l'objectif final de ce cours.

2.1.Définitions

a. Informatique

C'est la science du traitement automatique de l'information.

Information + Automatique = Informatique.

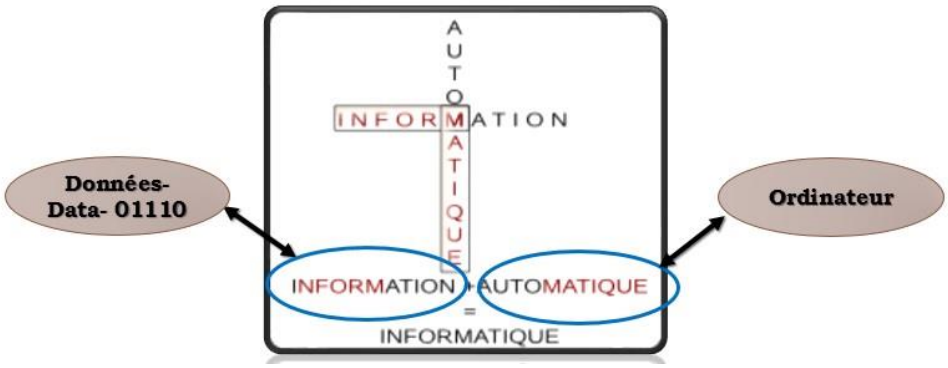


Figure 1 : Informatique : information + automatique

Une autre définition, comme étant l'ensemble des applications mettant en œuvre deux éléments distincts mais indissociables :

- Le hardware (le matériel) : qui est l'ensemble du matériel constitutif de la machine.
- Le software (le logiciel) : qui comprend l'ensemble des programmes qui s'exécutent sur le matériel. Ces programmes peuvent être divisés en deux catégories :

→ *Les programmes de base* : ils constituent le système d'exploitation (DOS, Windows, Linux, Unix, Mac OS, …).

→ *Les programmes d'application* : Selon les besoins, installés par les utilisateurs. Exemples : Word (pour rédiger les documents, …), Excel (Pour faire des calculs : facture, moyennes des notes, …), Turbo C (pour programmer en langage C)……

b. Ordinateur

L'ordinateur est la machine qui se charge du traitement automatique des informations. Il peut traiter différents types d'informations (textes, dessins, images, sons) mais à l'intérieur toutes ces informations sont converties sous forme binaire (0 et 1), nommée représentation interne.

Un ordinateur est généralement composé d'une unité centrale et des périphériques d'entrée et de sortie.

- *L'unité centrale* : est le boitier contenant tout le matériel électronique permettant à l'ordinateur de fonctionner ;
  → Processeur : CPU « Central Processing Unit » c'est le cerveau de l'ordinateur chargé de l'exécution des programmes, est une puce électronique caractérisée par sa marque (Intel486, Intel Pentium, Intel Pentium II, Intel Pentium III, Cyrix,..), et sa fréquence. Lorsque vous regardez à l'intérieur d'un ordinateur, il est souvent caché par un dispositif de refroidissement, qui lui permet de fonctionner à une vitesse élevée en conservant une température optimale.

Figure 2 : Le processeur et son système de refroidissement

→ Mémoires : Chargées de stocker les programmes et les données, organisées en cases dans lesquelles on peut stocker les informations / données.

Les informations sont très simples « information binaire »: 0 et 1. Chaque case élémentaire capable de mémoriser 0 ou 1 est appelée un bit (BInary digiT). La capacité d'une mémoire peut se mesurer en nombre d'octets disponibles tels que :

1 octet (o)/Byte = 8 bits ;
1 Kilo-octet (Ko) = 1024 octets ;
1 Méga-octet (Mo) = 1024 Ko ; 1 Géga-octet (Go) = 1024 Mo.
1 Téra-octet (To) = 1024 Go.

- Les périphériques d'entrées-sorties (E/S) : les périphériques en entrée sont utilisés pour faire entrer les données à un programme, permis ces périphériques : le clavier, la souris,.. Les périphériques en sortie sont utilisés pour communiquer les résultats à l'utilisateur, ex. l'écran.

c. Système informatique

C'est l'ensemble des moyens logiciels (software) et matériels (hardware) nécessaires pour satisfaire les besoins informatiques des utilisateurs.

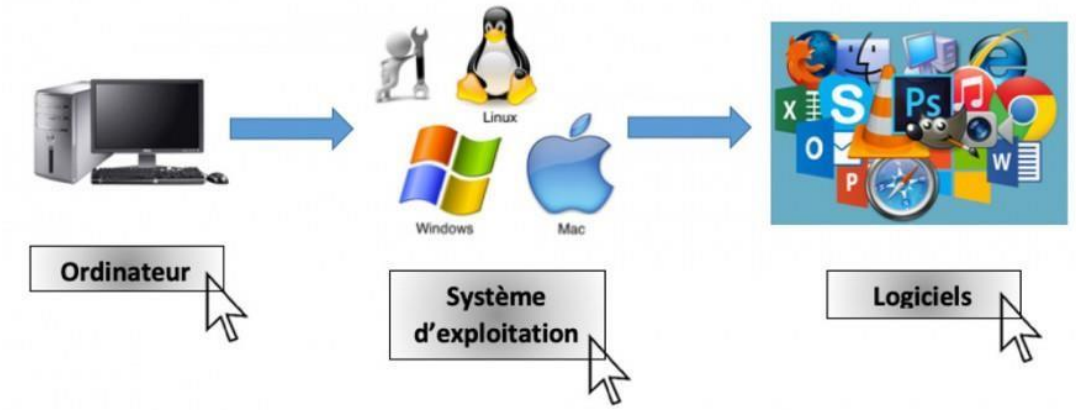


Figure 3 : L'ensemble d'un système informatique

2.2. Qu'est-ce qu'un algorithme ?

→ Programme=> Il s'agit de fournir une solution à un problème ;

→ La première étape consiste donc à analyser le problème ;

→ Le langage de description utilisé pour écrire le résultat de l'analyse (solution d'un problème) est le langage algorithmique => **algorithme** ;

→ Un algorithme est une séquence bien définie d'opérations (calcul, manipulation de données, etc.) permettant d'accomplir une tâche en un nombre fini de pas.

→ La mise en œuvre de l'algorithme consiste à écrire ces opérations dans un langage de programmation => programme.

a. Processus de résolution d'un problème

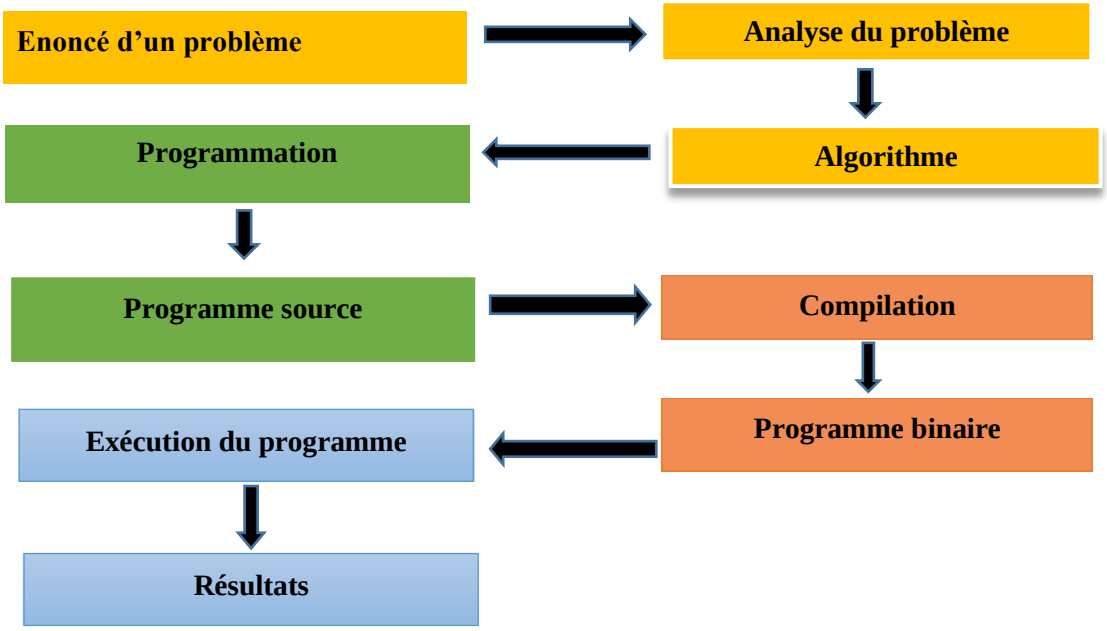


Figure 4 : Processus de résolution d'un problème

Selon le schéma au-dessus, la résolution d'un problème donné nécessite les étapes suivantes :

1. Comprendre l'énoncé du problème.
2. Décomposer le problème en sous problèmes plus simples à résoudre.
3. Associer à chaque sous problème une spécification :
   - Les données nécessaires (entrées),
   - Les données résultantes (sorties),
   - La démarche à suivre pour atteindre le résultat.
4. Ecrire un **algorithme** en utilisant le langage algorithmique.
5. Traduire l'algorithme en programme en utilisant un langage de programmation, ex. le langage C.
6. Compiler le programme via un compilateur spécialisé, pour détecter s'il y a des erreurs (sémantiques ou syntaxiques).
7. Exécution du programme compilé et production des résultats.

Analyse du problème

b. Algorithmique et Algorithme

L'algorithmique est la logique d'écrire des algorithmes.

Le nom algorithme vient du nom d'un mathématicien musulman ELKHAWARISMI. En effet, un algorithme est une suite d'instructions élémentaires écrite dans un langage (graphique, texte, pseudo-code) dont le but est de résoudre un problème donné.

Un algorithme peut être considéré comme une machine fonctionne en trois étapes :

1. Introduire les données nécessaires : les entrées (ou les données).
2. Exécuter séquentiellement des instructions sur ces données : les traitements / les actions.
3. Afficher les résultats obtenus : les sorties (les résultats).

Les entrées et les sorties forment la partie **déclarative** dans un algorithme ; La partie actions contient la liste des **instructions**.

Figure 5 : Algorithme / machine en trois étapes

c. Langage algorithmique et langage de programmation

Le langage algorithmique est un pseudo-langage qui tient en compte les caractéristiques de la machine, tout en étant plus flexible qu'un langage de programmation. C'est, donc, un compromis entre le langage naturel et un langage de programmation, caractérisé par une liste de mots clés.

Langage de programmation est un ensemble de notations capable de traduire des algorithmes en programmes compréhensibles par un ordinateur. Un langage est composé de deux parties : *syntaxique* et *sémantique*. La syntaxe est liée à la forme d'un programme écrit en certain langage de programmation. C'est la syntaxe qui détermine si l'on utilise des accolades ou des mots-clés comme *begin* et *end* pour déterminer un bloc d'instructions, etc. La sémantique, par contre, détermine la signification d'un programme.

d. Programme

Un programme est composé d'un ensemble fini d'instructions écrites dans un langage de programmation compréhensible par la machine, en d'autre terme c'est le résultat de traduction d'un algorithme dans un certain langage de programmation. En effet, un programme est composé de données et d'instructions à exécuter pour traiter ces données afin de produire un résultat.

Chapitre 2
« Algorithme séquentiel simple»

Chapitre 2 : Algorithme séquentiel simple

Dans ce chapitre, nous allons présenter la structure générale d'un algorithme en précisant les principales parties, les types de données, les instructions de base et la représentation d'un algorithme par un organigramme.

L'objectif principal de ce chapitre est de permettre à l'étudiant d'écrire un algorithme simple en respectant une structure bien déterminée et le traduire en programme C.

1. Notion de langage et langage algorithmique

1.1.Algorithmique

Consiste à définir des règles et des techniques qui sont utilisées pour concevoir des algorithmes

1.2.Algorithme

Un algorithme est une suite d'instructions élémentaires qui permettent de résoudre un problème donné.

Un algorithme est appelé aussi « **pseudo-code** », il doit être :

→ *lisible* : l'algorithme doit être compréhensible même par un non-informaticien ;

→ *de haut niveau* : l'algorithme doit pouvoir être traduit en n'importe quel langage de programmation ;

→ *précis* : chaque élément de l'algorithme ne doit pas porter à confusion, il est donc important d'éliminer toute ambiguïté ;

→ *concis* : un algorithme ne doit pas dépasser une page. Si c'est le cas, il faut décomposer le problème en plusieurs sous-problèmes ;

→ *structuré* : un algorithme doit être composé de différentes parties facilement identifiables.

Exemple d'un algorithme à partir de la vie quotidienne :

Creuser un trou, placer un arbre dans un trou et reboucher un trou sont des opérations élémentaires (des instructions simples) que toute personne (machine en informatique) est censée savoir exécuter. Cependant, si un jardinier (programmeur) veut faire planter un arbre par une

personne qui ne sait pas le faire, il doit lui fournir un descriptif « un algorithme » qui lui indique les opérations à faire ainsi que leur ordre d'exécution (séquencement).

Algorithme de plantation d'un arbre
1. Creuser un trou ; 2. Placer l'arbre dans le trou ; 3. Reboucher le trou ;

1.3. L'algorithme dans le processus de résolution d'un problème (problème -> programme)

Le développement d'un programme pour résoudre un problème donné peut suivre plusieurs étapes :

→ *La première étape (Réflexion)* : consiste à analyser le problème donné. Le résultat de cette étape est la décomposition du problème en composants élémentaires qu'on appelle aussi des opérations.

→ ***La deuxième étape*** : consiste à établir un algorithme qui est une présentation des étapes de résolution du problème analysé en respectant un certain formalisme (un ensemble de règles d'écriture –langage algorithmique[1]-) bien déterminé.

→ *La troisième étape* : est la traduction de l'algorithme écrit en programme, en utilisant un langage de programmation[2] choisi (Par exemple, le langage C). Une fois le programme écrit, il faut le vérifier et le corriger en lançant la compilation à travers un logiciel nommé le compilateur. Le compilateur est un logiciel qui détecte les erreurs syntaxique du programme, mais ne détecte pas les erreurs liées à la logique ou sémantique.

→ *La dernière étape* : consiste à exécuter le programme compilé pour produire les résultats finaux.

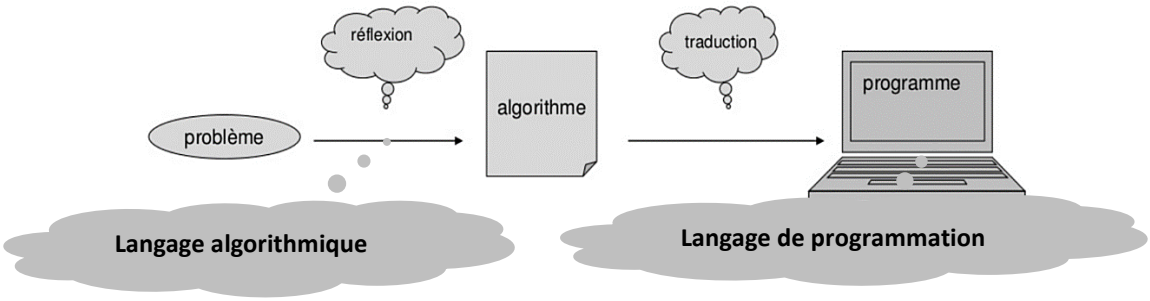


Figure 6 : Problème au programme

[1] Un pseudo-langage qui tient en compte les caractéristiques de la machine, plus flexible qu'un langage de programmation.

[2] Un ensemble de notations capable de traduire des algorithmes en programmes compréhensibles par un ordinateur.

2. Structure d'un algorithme

Un algorithme est composé de trois parties :

→ *La partie en-tête* : permet d'identifier l'algorithme, elle sert à lui donner un nom.

→ *La partie déclarative* : c'est une liste de toutes les données (constantes, variables, ...) utilisées et manipulées dans le corps de l'algorithme

→ *La partie instructions (corps de l'algorithme)* : dans cette partie sont placées les opérations et les instructions à exécuter sur les données déclarées dans la partie précédente. Elle commence par le mot clé **Début** qui indique le début de la partie, et se termine par le mot clé **Fin** qui indique la fin de notre algorithme. Chaque instruction se termine par un point-virgule.

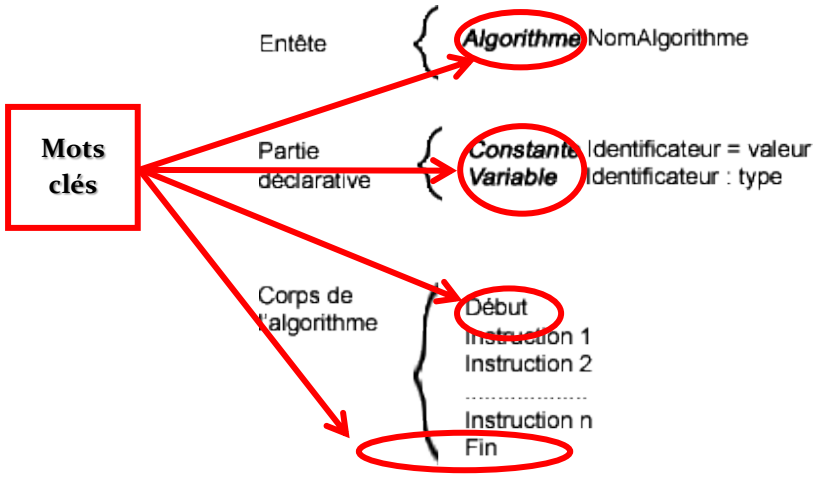


Figure 7 : Structure générale d'un algorithme

3. Les données : variables & constantes

3.1.Une variable

Une variable est un emplacement mémoire identifié par un nom « identificateur » et un type de données, destinée à stocker une valeur, qui peut être modifiée durant les traitements sans changer la structure ou le type de données. La variable est caractérisée par trois éléments.

→ *Identificateur ou Nom* : le nom d'une variable doit être unique, il respecte une syntaxe particulière :

- Il est constitué d'une suite de lettres de l'alphabet (latin) et de chiffres.
- Il commence, obligatoirement, par une lettre.
- Le caractère souligné "_" peut jouer le même rôle qu'une lettre de l'alphabet.
- Il est également recommandé d'utiliser des identificateurs évoquent l'information stockée.

- Il doit être différent de tous les mots clés (Début, Fin, Si, Alors, Sinon, …).
- Les caractères de l'alphabet grec sont interdits. Si on a besoin d'utiliser α, β, δ, ou Δ, on doit les écrire en alphabet latin comme alpha, beta, gamma, delta …..
- Il est interdit d'utiliser des caractères accentués (é, è, î, ç, ', ", …).
- Il est interdit d'utiliser des indices ou des exposants ;

Exemples : x, y, PI, rayon_du_cercle, _x1, _y2, …

→ *Type* : le type d'une variable indique la nature de la valeur que peut une variable contenir, en effet il existe quatre types simples (élémentaires) : Entier, Réel, Booléen et Caractère.

→ *Contenu* : le contenu d'une variable est la valeur initialisée ou attribuée à la variable dans une étape de l'exécution du programme.

Syntaxe de déclaration des variables

Variable nomVariable1, …, nomVariableN1 : Type ;

Exemples *:*

- Variables A, B, C, D : Entier // Déclaration de 4 variables de type Entier.
- X, Y, Z : Réel // Déclaration de 3 variables de type Réel.
- Reponse : Booléen // Déclaration d'une variable de type Booléen.
- Rep : Caractère // Déclaration d'une variable de type Caractère.

3.2.Une constante

Une constante est une variable dont la valeur ne change pas au cours de l'exécution d'un programme.

Syntaxe de déclaration des constantes

Constante nomConstante = Valeur

Exemples :

- **Constante** Nombre = 15 // Constante de type Entier
- Pi = 3.14 // Constante de type Réel
- Reponse = Oui // Constante de type Booléen
- Rep = 'N' // Constante de type Caractère
- Question= " Quoi ? " // Constante de type Chaîne

4. Types de données

Le type de données correspond au genre d'information utilisé. Les types standards (prédéfinis) en langage algorithmiques sont : Entier, Réel, Booléen, Caractère, Chaîne (suite de caractères). Chaque type est muni d'un ensemble d'opérateurs.

4.1.Entier

Représente l'ensemble des valeurs entières, positives ou négatives. Exemple : −30, 12, −1, 0. Il est muni des opérateurs suivants :

→ Les opérateurs arithmétiques : + (addition), - (soustraction), * (multiplication).

→ La division entière, notée **Div** , tel que x Div y donne **la partie entière du quotient** de la division entière de x par y. Exemple : 7 Div 2 = 3.

→ Le modulo, noté **Mod**, tel que x mod y donne **le reste** de la division entière de x par y. Exemple : 7 Mod 2 = 1.

→ Les opérateurs de comparaison classiques : <, ≤, =, ≠, ≥, >.

→ D'autres opérations sont définies par des fonctions, nous pouvons citer : **abs**(n) : fonction fournit la valeur absolue de l'entier n.

4.2.Réel

Une variable de type réel prend ses valeurs dans l'ensemble des nombres décimaux (les nombres à virgule). Exemple : -3.7, -1.5, 0, 3.0, 18.25. Le type Réel est muni des opérateurs suivants :

→ Les opérations arithmétiques classiques : + (addition), - (soustraction), * (multiplication), / (division).

→ Les opérateurs de comparaison classiques : <, ≤, =, ≠, ≥, >

→ D'autres opérations sont définies par des fonctions, nous pouvons citer :

- **trunc**(x) : fonction donne la partie entière du nombre réel x.
- **round**(x) : fonction donne l'entier le plus proche du nombre réel x.
- **abs**(x) : fonction donne la valeur absolue du nombre réel x.
- **sqrt**(x) : fonction donne la racine carrée du nombre réel x.
- Les fonctions trigonométriques : sin(x), cos(x), arctg(x), …

NB : Nous pouvons affecter à une variable de type réel une autre variable de type entier. C'est la machine qui prend en charge la conversion de la valeur entière en valeur réelle. Les variables du type entier et les variables du type réel peuvent être combinées dans des opérations définies sur les réels.

4.3.Booléen (logique)

Une variable du type booléen elle peut prendre seulement les deux valeurs VRAI ou FAUX. Les opérateurs booléens (logiques) les plus utilisés sont : le NON, le ET et le OU. Ils sont définis par les tables de vérité ci-dessous. A et B deux variables du type Booléen.

A	Non A
Faux	Vrai
Vrai	Faux

A	B	A ET B
Faux	Faux	Faux
Faux	Vrai	Faux
Vrai	Faux	Faux
Vrai	Vrai	Vrai

A	B	A OU B
Faux	Faux	Faux
Faux	Vrai	Vrai
Vrai	Faux	Vrai
Vrai	Vrai	Vrai

4.4.Caractère

Ce type est utilisé pour stocker les caractères, il comporte :

→ Les lettres de l'alphabet latin : a .. z, A .. Z.

→ Les chiffres : 0 .. 9.

→ Les symboles utilisés en tant qu'opérateurs : + - * / < = > …

→ Les caractères de ponctuation : . , ; ! ? …

→ Les caractères spéciaux : @ % & # …

Exemple : 'A', ' /', '9', ……

4.5.Chaîne

Une chaîne est une suite de caractères. Une chaîne est encadrée par deux apostrophes doublées. Exemples : "Bonjour tout le monde", "Université d'Oum El Bouaghi ", …. Ce type est caractérisé par la longueur d'une chaîne qui représente le nombre de caractères de cette chaîne. Exemples : "Bonjour" est une chaîne de longueur 7 ; "" : Représente une chaîne vide de longueur 0. Les fonctions prédéfinies sur les chaînes sont :

→ **Length**(str) : elle fournit la longueur de la chaîne str.

→ **Concat**(str1, str2) : elle fournit la chaîne obtenue par la concaténation des deux chaînes str1 et str2. Exemple : Concat("Module", " Algorithmique") donne "Module Algorithmique".

5. Instructions de base

5.1.Les expressions

Une expression est une combinaison de plusieurs opérateurs et opérandes ou facteurs. Chaque expression a une valeur et un type. Exemple : **a + 5** est une expression qui représente une addition (opérateur +) entre les deux facteurs (opérandes) a et 5. Généralement, il existe deux types d'expressions :

→ *Les expressions arithmétiques* : utilisent les opérateurs arithmétiques (+, -, /, *) ainsi que les deux opérateurs Div et Mod. Exemple : a + b – c/d est une expression arithmétique.

→ *Les expressions booléennes ou logiques :* utilisent les opérateurs logiques (ET, OU, NON, …) ainsi que les opérateurs de relation (>, <, <=, <>, …). Exemple : (a > b et b <= c) et une expression logique.

Le calcul de la valeur d'une expression contient plus d'un opérateur dépend du sens que l'on donne à cette expression. Exemple : X + Y * Z est une expression ambigüe. L'utilisation des parenthèses permet de lever l'ambiguïté : (X + Y) * Z ou X + (Y * Z).

Priorité des opérateurs : en absence des parenthèses, pour éviter toute ambiguïté, un ordre de priorité entre les opérateurs est introduit de façon décroissante comme suit :

→ *Priorité n°1* : Opérateurs unaires : + (exemple : +X) , - (exemple : -X), NON (NON X).

→ *Priorité n°2* : Opérateurs multiplicatifs : *, /, Div, Mod, ET.

→ *Priorité n°3 :* Opérateurs additifs (binaires) : +, -, OU.

→ *Priorité n°4 :* Opérateurs relationnels : <, ≤, =, ≠, ≥, >.

N.B : Les parenthèses ouvrantes et fermantes sont considérées comme des opérateurs les plus prioritaires.

5.2. Les commentaires

Un commentaire est un texte ou une phrase facultatif situé entre les symboles (//, /*) qui n'a aucun effet sur le fonctionnement d'un algorithme. Elle sert à expliquer l'objectif de certaines instructions utilisées dans un algorithme.

5.3.Les instructions élémentaires ou de base

En algorithmique, Il existe trois instructions élémentaires : Lire (entrée) , Ecrire (sortie) et l'affectation (←).

a. Les instructions d'entrée / sortie

→ ***Lire*** : permet à l'utilisateur d'attribuer une valeur à une variable via un périphérique d'entrée (clavier).

Syntaxe : ***Lire*** (Identificateur_variable) ;

Quelques remarques :

- *La* donnée saisie au clavier doit être de même type que la variable déclarée.
- Une variable lue ne doit pas être initialisée.
- L'instruction lire permet de lire une ou plusieurs variables à la fois. Exemple : Lire (x, y).

→ ***Ecrire*** : instruction d'affichage, permet d'afficher un résultat ou un message sur un périphérique de sortie (écran).

Syntaxe : **Ecrire** ()

Exemples :

- Ecrire(a) ; // Afficher la valeur de la variable a
- Ecrire ("la valeur de a=") ; // Afficher le message entre guillemets
- Ecrire ("la valeur de a=",a) ; // Afficher le message entre guillemets et la valeur de la variable a

b. L'affectation

Cette instruction permet d'assigner ou affecter une valeur à une variable en utilisant l'opérateur d'affectation (←), se lit reçoit.

nom _de _variable ← valeur_à _affecter ; nom_de _variable ← expression ;

Exemples : x ← 5 ; x ← (x * i) ; x ← Pi ;

Quelques remarques :

- Une variable n'a pas une valeur, la première affectation s'appelle *initialisation*.
- Le traitement d'une instruction d'affectation par un ordinateur consiste à évaluer ou calculer premièrement la partie droite de l'instruction, puis il affecte le résultat à la variable qui se trouve à gauche.

6. Représentation d'un algorithme par un organigramme

Un organigramme est une représentation schématique d'un algorithme, cette représentation permet de donner une bonne visualisation des liens entre les éléments d'un algorithme.

Un organigramme est composé de plusieurs nœuds reliés entre eux par des flèches.

- une ellipse pour représenter le début et la fin d'un algorithme.
- un parallélogramme pour représenter les Entrées-Sorties (Lire, Ecrire).
- un rectangle pour représenter les instructions.
- un losange pour les instructions conditionnelles (Tests).

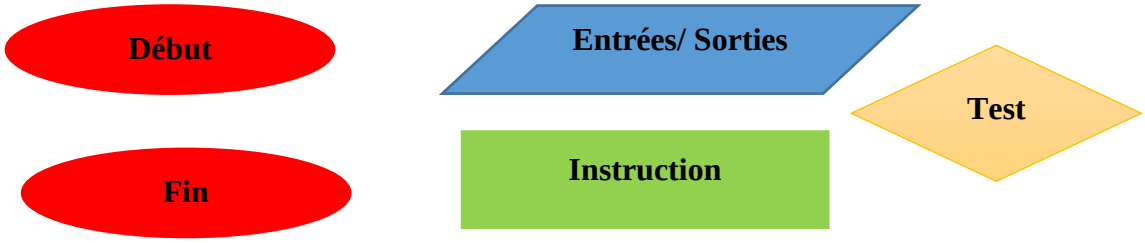
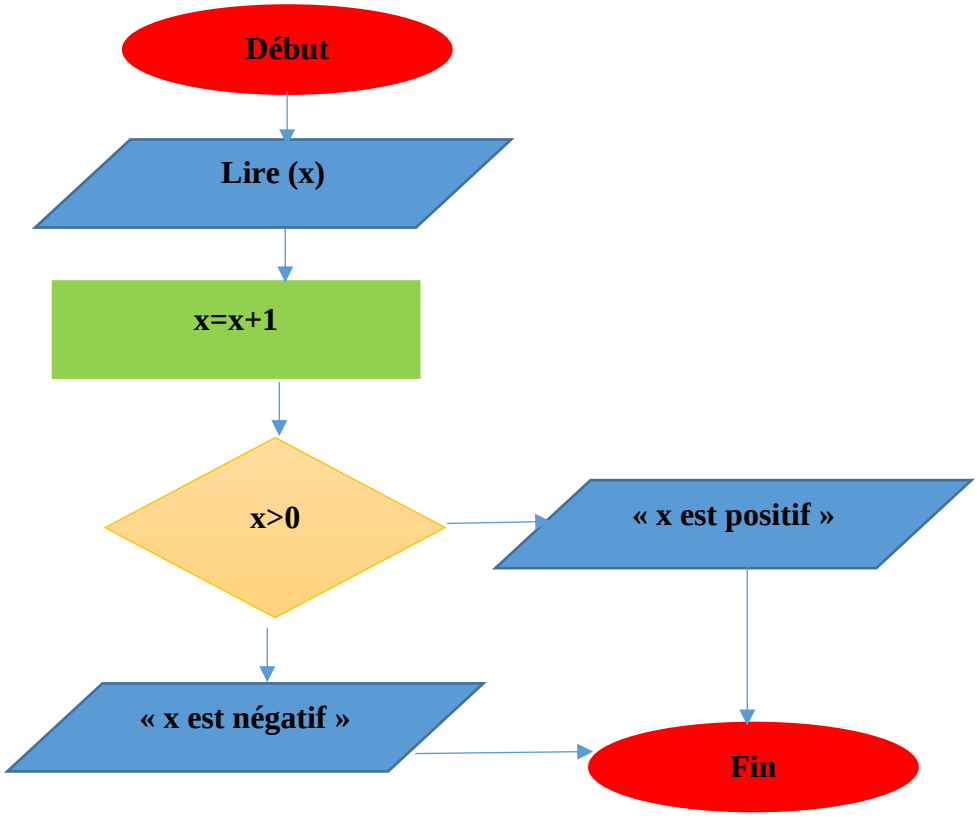


Figure 8 : Les différents nœuds d'un organigramme

Exemple : Un organigramme d'un algorithme qui permet de vérifier le signe d'un nombre donné par l'utilisateur (positif ou négatif).



7. Traduction en langage C

En 1970, Dennis RITCHIE a créé le langage C, un langage de haut niveau, pour écrire le système d'exploitation Unix. La conception de ce langage a été régie par les points principaux suivants :

→ la souplesse

→ la fiabilité

→ la portabilité

Exemple d'un premier programme en C

```
main( )
{
printf("bonjour la première année");
}
```

- **main** () indique qu'il s'agit du programme principal.
- **{** et **}** jouent le rôle de début et fin d'un algorithme.
- **Printf** est l'instruction d'affichage à l'écran, le message étant entre guillemets.
- **;** indique la fin d'une instruction.

Le tableau ci-dessous résume les principales traductions des différentes parties et éléments d'un algorithme en C.

En algorithmique	En C

Algorithme **Nom ;**
Début
……..
Fin
	main ()
	{
	……..
	}

Constante **identificateur = valeur ;**	***const*** *<Type> < Identificateur> = Valeur ;*
Variable **identificateur : type ;**	*< type > < identificateur >*;
Affectation (←).	=
Lire ()	scanf()
Ecrire()	printf() *Formats d'affichage* • %d : le format pour afficher une valeur entière. • %f : le format pour afficher un float. • %c : le format pour afficher un caractère.

	• %s : le format pour afficher une chaîne de caractères.
Initialisation d'une variable	*<Type> <Identificateur> = Valeur ;*
Commentaire	/* texte du commentaire */

Types de données en C

Type	**Signification**	**Représentation Système**	
		Taille (bits)	**Intervalle des valeurs**
int	Entier	16	-32768 à 32767
short(ou short int)	Entier	16	-32768 à 32767
long (ou long int)	Entier en double longueur	32	-2147483648 à 2147483647
char	Caractère	8	
float (ou short float)	Réel	32	$\pm 10^{-37}$ à $\pm 10^{38}$
double(ou long float)	Réel en double précision	64	$\pm 10^{-307}$ à $\pm 10^{308}$
long double	Réel en très grande précision	80	$\pm 10^{-4932}$ à $\pm 10^{4932}$

8. Construction d'un algorithme simple

Dans cette partie, nous allons écrire notre premier algorithme, en suivant les étapes suivantes

1. *Le problème*

   Écrire un algorithme qui permet de calculer le carré d'un nombre entier.

2. *L'analyse*

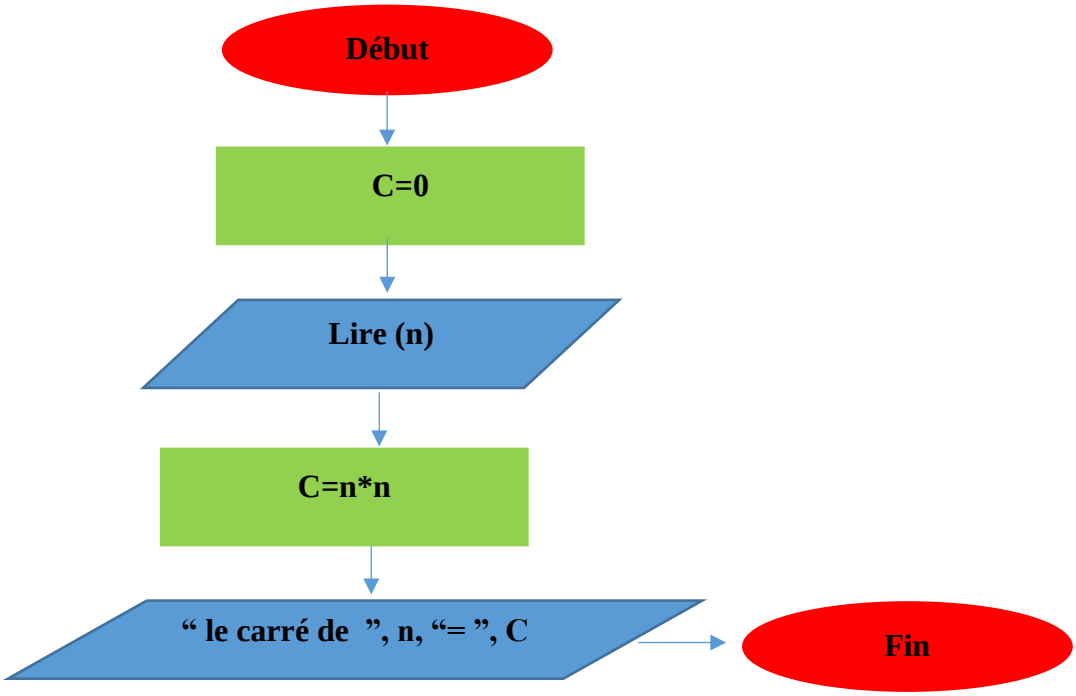
   Nous cherchons à écrire un algorithme qui demande un nombre entier à l'utilisateur et qui affiche par la suite le carré de ce nombre.

   Pour résoudre ce problème nous aurons besoins de deux variables de type entier, la première variable *n* sera lue à partir du clavier, la seconde est C aidera à stocker le résultat.

3. *Ecrire l'algorithme*

Algorithme Calcul_carre ;
Variables : n, C : **entier** ;
Début
C <- 0 ; // Initialisation
Lire (n)
C <- n*n ;
Ecrire (" le carré de ", n, "= ", C) ;
Fin

4. *La représentation par un organigramme*



5. *En langage C*

```
#include <stdio.h>
main()
{
    int n;
    int C=0 ;
    printf("donner un nombre \n");
    scanf("%d",&n);
    C=n*n;
    printf("le carrée de nombre %d = %d",n,C);}
```

9. Exercices d'application

Exercice n° 1 : Ecrire un algorithme qui déclare deux variable i et j et applique par la suite sur ces variables les instructions suivantes : i=2; j=(i*3)+5; i=j;

Algorithme Premier_Algorithme ;
Variables i,j :entier ;
Début
i ← 2;
j ← (i*3)+5;
i ←j;
Fin

```
main( )
{
int i, j;
i=2;
j=(i*3)+5;
i=j
}
```

Exercice n° 2 : Ecrire un algorithme qui calcule est affiche le double d'un nombre entier introduit par l'utilisateur.

Algorithme Calcul_double ;
Variables x,Y :entier ;
Début
Y←0 ;
Ecrire (" donner un nombre entier") ;
Lire (x)
Y←x*2 ;
Ecrire (" le double de" ,x, " =", Y) ;
Fin

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

```
    int x,y;
    Y=0 ;
    printf("donner un nombre\n");
    scanf("%d",&x);
    Y=2*x;
    printf("le duoble de ce nombre  %d = %d" ,x,Y);
}
```

Exercice n° 3 : Ecrire un algorithme qui permet de convertir en degrés Celsius (C) une température entrée au clavier exprimée en degrés Fahrenheit (F) en utilisant la formule suivante:

$$C = 5/9 * (F - 32)$$

Algorithme temperature_fahrenheit_celsius ;
Variables celsius, fahrenheit : Réel ;
Début
Ecrire (”Entrez une temperature en degres Fahrenheit : ”);
Lire (fahrenheit);
celsius ← 0.55556 *(fahrenheit − 32.0) ;
Ecrire(”Temperature en degre Celsius ”, celsius);
Fin

```
#include <stdio.h>
main ( void )
{
float celsius , fahrenheit ;
printf (" Entrez une température en degré Fahrenheit : ") ;
scanf ("%f " , &fahrenheit ) ;
celsius = 0.55556 * (fahrenheit - 32.0) ;
printf ("Temperature en degré Celsius %f . \ n" , celsius) ;
}
```

TD1 : Algorithme séquentiel simple

Objectifs pédagogiques

- Comprendre la manière de décomposition d'un problème en étapes élémentaires ;
- Traduire ces étapes en opérations afin de les représenter sous forme d'un organigramme ;
- Saisir la structure générale et écrire un algorithme simple ;
- Apprendre à exécuter un algorithme en suivant les changements et les traces de ses variables (simuler le scénario d'exécution) ;
- Maitriser les principales instructions : l'affectation, lecture (entrée) et écriture (sortie).

Exercice n°1

Quelles seront les valeurs des variables A et B après l'exécution des instructions suivantes ?

```
Algorithme Instruction_Affectation1 ;
Variables : A, B: Entier ;
Début
A ← 1;
B ← A+8;
A ← B-5;
Fin
```

Exercice n°2

Quelles seront les valeurs des variables A et B après l'exécution des instructions suivantes ?

```
Algorithme Instruction_Affectation1 ;
Variables : A, B: Entier ;
Début
B ← -4;
A ← 5+B;
Lire (B) ;
A ← B-1;
Fin
```

Exercice n°3

Quelles seront les valeurs des variables A, B et C après l'exécution des instructions suivantes ?

```
Algorithme Instruction_Affectation2 ;
Variables : A, B, C : Entier ;
Début
A ← 10;
B ← -4;
C ← A * B;
A ← C+ 4;
B ← B – A;
Fin
```

Exercice n°4

Ecrire un algorithme qui demande deux nombres à l'utilisateur, puis calcule et affiche la somme, la multiplication et la soustraction de ces deux nombres. Tracer l'organigramme approprié.

Exercice n°5

Écrire un algorithme qui déclare trois entiers, deux réels, un booléen et deux caractères.
Affecter une valeur à chaque variable et afficher le résultat. Tracer l'organigramme approprié

Exercice n°6

Ecrire un algorithme qui lit le prix d'achat Hors Taxe d'un article HT, le nombre d'articles NB et la taxe TVA (Taxe pour Valeur Ajoutée), et qui fournit par la suite le prix total TTC correspondant. Sachant que : *TTC = NB*HT*(1+TVA)*. Tracer l'organigramme approprié

Exercice n°7

Lors d'une vente promotionnelle, un magasin annonce une réduction de 20% sur tous les articles. Écrire un algorithme qui lit le prix d'un article entré au clavier et affiche le nouveau prix avec la réduction.

Exercice n°8

Écrire un algorithme permettant de permuter les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.
Exemple : A = 5 et B = 3, le résultat de la permutation est A = 3 et B = 5.

a. Première solution : en utilisant une variable intermédiaire ;
b. Deuxième solution : en utilisant les opérations d'addition et de soustraction.

Exercice n°9

Soit une fonction mathématique *f* définie par : $f(x) = (2x + 3) / (3x^2 + 2)$.
Écrire un algorithme qui calcule l'image de *f* avec un nombre *x* donné par l'utilisateur.

Tracer l'organigramme approprié

TP1 : Les premiers programmes en C

Objectifs pédagogiques

- Familiariser avec l'environnement de développement Turbo C ;
- Ecrire les premiers programmes à partir des algorithmes (transformer un algorithme en programme C) ;
- Comprendre les concepts de compilation et exécution.
- Faire sortir les aspects fondamentaux du langage C (structure générale, la fonction main, déclarations, erreur syntaxique, …)

Exercice n°1

Écrire un programme qui affiche « Bonjour tout le monde ! ».

Modifier le programme précédent de façon à obtenir le même résultat sur l'écran en utilisant plusieurs fois la fonction **printf()**,

Exercice n°2

Traduire l'algorithme suivant en programme C.

Algorithme Exemple_C ;
Variables A, B, C : entier ;
Début
Lire (A) ;
B← 3 ;
C← (A * B) + (A-B) ;
A← 2;
C ← (B – A) *A ;
Ecrire (A) ;
Ecrire (B) ;
Ecrire (C) ;
Fin

Exercice n°3

Écrire un programme C qui calcule la somme de deux nombres entiers **A** et **B**.

Exercice n°4

Écrire un programme C qui demande la longueur et la largeur d'un rectangle, puis il calcule et affiche sa surface et son périmètre.

Exercice n°5

Ecrire un programme C qui permute et affiche les valeurs de trois variables A, B, C de type entier données par l'utilisateur :

$$A \leftarrow C , B \leftarrow A , C \leftarrow B$$

Exercice n°6

Ecrire un programme C qui calcule la somme, la soustraction et la multiplication de quatre nombres du type entier donnés par l'utilisateur.

Exercice n°7

Ecrire un programme C qui demande à l'utilisateur trois réels et affiche par la suite leur moyenne en précisant deux chiffres après la virgule.

Exercice n°8

Ecrire un programme C qui permet de calculer la distance entre deux points du plan dont les coordonnées sont données par A (XA, YA), B (XB, YB). La distance D entre ces deux points est donnée par la formule suivante :

$$D = \sqrt{(XA - YA)^2 + (XB - YB)^2}$$

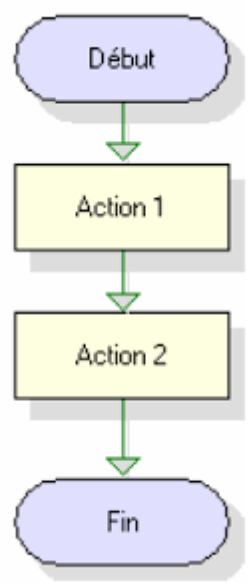


Figure 9 : Le principe d'un algorithme séquentiel simple.

Chapitre 3 « Les structures conditionnelles»

Chapitre 3 : Les structures conditionnelles

En langage algorithmique et en C

Dans notre vie quotidienne, nous tombons toujours dans des situations où nous devons choisir et prendre une décision. Exemple, devant un feu de signalisation si le feu est rouge alors je dois m'arrêter, si le feu est vert alors c'est les piétons qui traversent la route. Nous avons donc deux actions différentes selon l'alternative prise. Dans ce chapitre, nous allons présenter les instructions conditionnelles qui peuvent utiliser pour exprimer l'alternative, c'est-à-dire une situation dans laquelle on est amené à choisir entre deux solutions possibles.

Les objectifs pédagogiques de ce cours sont :

→ Comprendre le rôle d'une structure conditionnelle ou de contrôle.

→ Apprendre les différentes structures conditionnelles utilisées en algorithmique.

→ Ecrire des algorithmes pour résoudre des problèmes nécessitant l'utilisation des structures conditionnelles.

1. Structure conditionnelle simple

L'algorithme doit prendre une décision en fonction d'une condition, cette dernière est une expression booléenne, elle ne peut prendre que deux valeurs : vrai ou faux. Dans la structure conditionnelle simple (alternative réduite), seule la situation qui vérifie la condition (vrai) sera exécutée (exécution des instructions correspondantes), l'autre situation conduisant systématiquement à la sortie de la structure.

Syntaxe

```
Si (condition) alors
    Bloc d'instructions
Fin si
```

Syntaxe en C

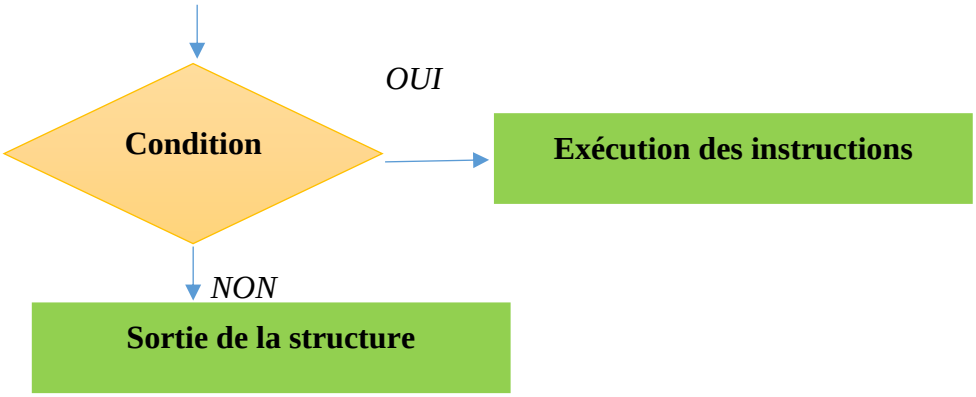
```
if ( )
{
}
```

Exemple

Ecrire un algorithme qui permet d'afficher la mention « admis » d'un étudiant selon son moyenne.

```
Algorithme alternative_reduite;
Variable moy : réel ;
Début
Ecrire (" donner la moyenne de l'étudiant") ;
Lire (moy) ;
Si (moy >=10) alors
Ecrire (" Admis") ;
Fin si
Fin
```

Organigramme



2. Structure conditionnelle alternative

Cette structure nommée aussi l'alternative complète, elle permet de sélectionner sur la base d'une condition un bloc d'instructions à exécuter parmi deux blocs.

Syntaxe

```
Si Condition alors
   Bloc d'instructions 1;
Sinon
   Bloc d'instructions 2;
Fin si
```

Syntaxe en C

```
if ( )
{
        }
else
{
       }
```

Scénario d'exécution

Commence par l'évaluation (calcul) de la condition :

→ Si la condition est vérifiée (= Vrai), c'est le Bloc d'instructions 1 qui va exécuter. Le Bloc 2 va ignorer.

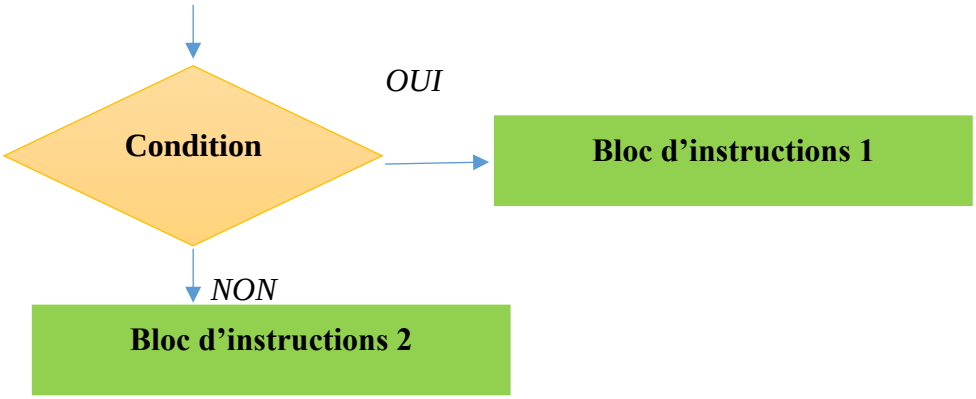
→ Si la condition est fausse (= Faux), c'est le Bloc d'instructions 2 qui va exécuter. Le Bloc 1 va ignorer.

Exemple

Ecrire un algorithme qui permet d'afficher la mention « admis ou ajourné » d'un étudiant selon son moyenne.

```
Algorithme alternative_complete ;
Variable moy : réel ;
Début
  Ecrire (" donner la moyenne de l'étudiant") ;
 Lire (moy) ;
 Si (moy >=10) alors
     Ecrire (" Admis") ;
Sinon
    Ecrire ("Ajourné") ;
Fin si
Fin
```

Organigramme



3. Structure conditionnelle imbriquée

Les instructions conditionnelles peuvent être imbriquées afin d'exprimer plusieurs choix multiples. Dans ce cas, l'évaluation des conditions s'arrêtera à la première condition fausse rencontrée.

Syntaxe

```
Si condition 1 alors
   Bloc d'instruction1 ;
Sinon si condition 2 alors
    Bloc d'instruction2 ;
Sinon
    Bloc d'instruction3 ;
Fin si
```

Exemple

Ecrire un algorithme qui permet de donner l'état de l'eau selon sa température : solide, liquide ou gazeux.

→ Première solution avec des structures conditionnelles simples

```
Algorithme  etat_deau ;
Variable Temp : entier
Début
Ecrire ("Entrer la température de l'eau :") ;
```

```
Lire (Temp) ;
Si (Temp =< 0 ) alors
    Ecrire ("Etat solide") ;
Fin Si
Si (Temp > 0) et (Temp < 100)[3] alors
   Ecrire ("Etat liquide") ;
Fin si
Si (Temp >= 100) alors
    Ecrire( "Etat gazeux") ;
Fin si
Fin
```

→ Deuxième solution avec une structure conditionnelle imbriquée

```
Algorithme  etat_deau ;
Variable Temp : entier
Début
Ecrire ("Entrer la température de l'eau :") ;
Lire (Temp) ;
Si (Temp =< 0 ) alors
  Ecrire ("Etat solide") ;
Sinon
     Si (Temp < 100) alors
         Ecrire (" Etat liquide") ;
    Sinon
         Ecrire ("Etat gazeux") ;
     Fin si
Fin si
Fin
```

[3] Une condition composée (ou complexe) formée de plusieurs (deux) conditions simples reliées par des opérateurs logiques (ET, OU)

4. Structure conditionnelle à choix multiple

L'instruction ***selon (ou suivant)*** permet de faire un choix parmi plusieurs cas possibles en fonction de la valeur d'une variable.

Cette instruction est utilisée lorsqu'on a un choix multiple sur une variable, cependant si le nombre de choix possibles est petit l'instruction conditionnelle imbriquée est possible.

Syntaxe

Selon (valeur ou expression) ***faire***
Cas valeur1 : instruction1 ;
Cas valeur2 : instruction2 ;
…
Cas $valeur_n$: $instruction_n$;

Autres cas : $instruction_{n+1}$;
Fin selon ;

Syntaxe en C

```
switch ( valeur ou expression ) {
case 1 :
Bloc d'instruction1 ;
break ;
case 2 :
Bloc d'instruction2 ;
break ;
case 3:
Bloc d'instruction3 ;
break ;      …….
default :
Bloc d'instruction(n+1) ;
}
```

Scénario d'exécution

L'exécution de cette structure conditionnelle commence par l'évaluation de l'expression. Seulement le bloc d'instructions qui contient dans sa liste la valeur de l'expression sera exécuté. Si la valeur de l'expression ne figure pas dans aucune liste, c'est le bloc d'instructions (n+1) qui sera exécuté (commence par le mot clé **Autres**).

Exemple

Ecrire un algorithme qui demande le numéro de mois et affiche par la suite la saison correspondante.

```
Algorithme saison ;
Variable mois : entier ;
Début
Lire (mois) ;
Selon (mois) faire
   Cas 12 : Ecrire ("C'est l'hiver") ;
  Cas 1 : Ecrire ("C'est l'hiver") ;
  Cas 2 : Ecrire ("C'est l'hiver") ;
  Cas  3 : Ecrire("C'est le printemps") ;
  Cas  4 : Ecrire("C'est le printemps") ;
  Cas  5 : Ecrire("C'est le printemps") ;
  Cas  6 : Ecrire("C'est l'été") ;
  Cas  7 : Ecrire("C'est l'été") ;
  Cas  8 : Ecrire("C'est l'été") ;
   Cas 9 : Ecrire("C'est l'automne") ;
  Cas  10: Ecrire("C'est l'automne") ;
  Cas  11 : Ecrire("C'est l'automne") ;
    Autres cas : Ecrire ("Ce n'est pas un mois de l'année")
FinSelon
```

5. Exercice d'application

Écrire un algorithme qui demande un entier et affiche s'il est pair ou impair.

Algorithme pair_impair ;
Variable n : entier ;
Début
Ecrire (" donner un nombre") ;
Lire (n) ;
Si (n Mod 2= 0) alors
Ecrire (" le nombre est pair") ;
Sinon
Ecrire ("le nombre est impair") ;
Fin si
Fin

En langage C

```
#include <stdio.h>
main ( )
{
int n ;
printf ( " entrer un nombre\n" ) ;
scanf ( "%d" ,&n) ;
if (n%2==0)
printf ( " le nombre est pair \n" ) ;
else
printf ( " le nombre est impair \n" ) ;
}
```

TD2 : Les structures conditionnelles

Objectifs pédagogiques

- L'exécution manuelle (déroulement) des algorithmes non séquentiels afin de comprendre le fonctionnement de ces derniers ;
- Maîtriser les différents types de structures conditionnelles (de contrôle) : simples, alternatives, imbriquées, multiples ;
- Traduire quelques algorithmes en programmes C pour saisir la syntaxe générale des programmes C.

Exercice n°1

Dérouler (exécution manuelle) l'algorithme suivant avec plusieurs exemples et donner les résultats à chaque fois.

```
Algorithme Exemple_deroulement ;
Variables X, Y : Entier ;
Début
Ecrire ("Donner des valeurs pour les variables X et Y") ;
Lire (X, Y) ;
Si (X> Y) alors
    X ←  X-Y ;
    Ecrire ("La nouvelle valeur de X = " X) ;
Sinon
    Y ←  Y-X ;
    Ecrire ("La nouvelle valeur de Y = " Y) ;
Fin si
Fin
```

Exercice n°2

Ecrire un algorithme qui calcule la taxe sur le chiffre d'affaire (CA) sachant qu'elle est de :

- 10% si le CA<5000 DA
- 20% si le CA >=5000 DA

Exercice n°3

Écrire un algorithme qui lit deux variables au clavier et les affiche en ordre croissant.
Modifier l'algorithme pour prendre en considération le cas de l'égalité en utilisant une structure simple et une structure alternative.

Exercice n°4

Écrire un algorithme qui lit trois variables au clavier et affiche le maximum des trois. Traduire l'algorithme en programme C.

Exercice n°5

Ecrire un algorithme qui permet de résoudre une équation de seconde degré $ax^2+bx+c=0$ /a non nul.

NB : pour pouvoir résoudre une telle équation, il faut tout d'abord calculer le discriminant Δ/

$\Delta = b^2 - 4*a*c$

Si $\Delta < 0$, il n'y a pas de solution.

Si $\Delta = 0$, il y a une seule solution à l'équation : x= -b/(2*a).

Si $\Delta > 0$ il y a deux solutions qui sont : x1 = (-b-√Δ)/(2*a) x2= (-b+√Δ)/(2*a).

Exercice n°6

Ecrire un algorithme qui lit deux valeurs entières (A et B) au clavier et qui affiche le signe du produit de A et B sans faire la multiplication. Traduire l'algorithme en programme C.

Exercice n°7

Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :

- « Poussin » de 6 à 7 ans
- « Pupille » de 8 à 9 ans
- « Minime » de 10 à 11 ans
- « Cadet » après 12 ans

Traduire l'algorithme en programme C.

Exercice n° 8

Une librairie facture **5 DA** les dix premières photocopies, **4 DA** les vingt suivantes et **3 DA** au-delà. Écrire un algorithme qui demande à l'utilisateur le nombre **NB** de photocopies effectuées et qui affiche la facture correspondante.

Exercice n°9

Ecrire un algorithme qui permet de tester si une année est bissextile ou non.

Une année est bissextile si elle est divisible par 4 et pas par 100 ou si elle est divisible par 400.

Exercice n°10

Ecrire un algorithme qui permet de saisir un nombre puis détermine s'il appartient à un intervalle donné ou non, sachant que les extrémités de cet intervalle sont fixées par l'utilisateur.

Exercice n°11

Ecrire un algorithme qui affiche le jour de semaine selon la valeur d'une variable *jour* de type caractère $\in$ {S,D,L,M ,m,J,V}. Traduire l'algorithme en programme C.

TP2 : Les structures conditionnelles en C

Objectifs pédagogiques

- Comprendre le fonctionnement et maîtriser la syntaxe des différents types des instructions conditionnelles ;

Exercice n°1 :

Ecrire un programme C qui permet de vérifier si un nombre entier donné par l'utilisateur est positif ou négatif.

Exercice n°2 :

Ecrire un programme C qui permet de vérifier si un nombre est divisible par 5 et 15 ou non.

Exercice n°3 :

Ecrire un programme C qui demande à l'utilisateur trois valeurs et affiche par la suite la valeur minimale entre ces trois valeurs (en utilisant des conditions simples).

Exercice n°4 :

Ecrire un programme C qui permet de résoudre une équation du 1er degré : **ax + b = 0**

Exercice n°5 :

Ecrire un programme C pour entrer le numéro du mois entre (1-12) et afficher le nombre de jours de ce mois en utilisant la structure conditionnelle imbriquée. En utilisant le tableau suivant :

Mois	Nombre de jours
Janvier, Mars, Mai, Juillet, Aout, Octobre, Décembre	31 jours
Février	28/29 jours
Avril, Juin, Septembre, Novembre	30 jours

Exercice n°6 :

Écrire un programme C qui lit la moyenne générale obtenue par un(e) étudiant(e) et affiche la mention. En utilisant le tableau suivant :

Moyenne	Mention
<10	INSUFFISANT
<12	PASSABLE
<14	ASSEZ BIEN
<16	BIEN
<18	TRES BIEN
>=18	EXCELLENT

Exercice n°7

Ecrire un programme qui permet de vérifier si un alphabet est une voyelle ou une consonne. Les lettres a, e, i, o et u en minuscules et en majuscules sont appelées voyelles. Les alphabets autres que les voyelles sont appelés consonnes.

Exercice n°8 :

Ecrire un programme qui demande à l'utilisateur de donner un chiffre (entre 0 et 9) et qui affiche ensuite le chiffre donné en lettres. Par exemple, si l'utilisateur tape le chiffre 9, le programme affichera neuf.

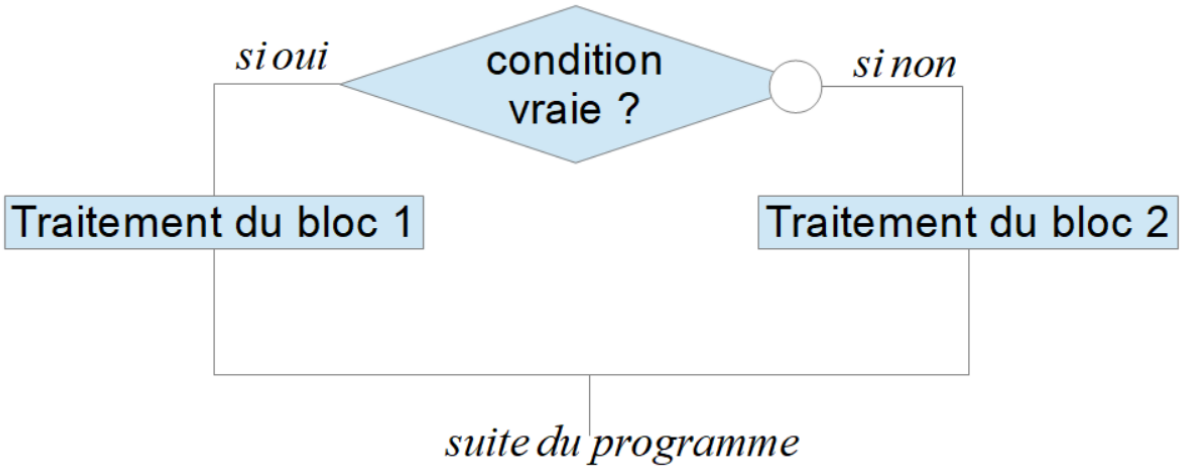


Figure 10 : Le principe de l'instruction conditionnelle

Chapitre 4 « Les structures alternatives»

Chapitre 4 : Les structures itératives (les boucles)

En langage algorithmique et en C

Les instructions itératives nommées aussi les boucles, leur principe consiste à faire répéter **un certain nombre de fois** l'exécution d'une instruction ou d'un bloc d'instructions. Le nombre de répétitions ou itérations est contrôlé soit par un compteur (le nombre de répétitions est connu) soit par une condition booléenne.

→ La boucle où le nombre de répétitions est connu est la boucle pour.

→ Les boucles où le nombre de répétitions est inconnu sont : Tant que et répéter.

Les objectifs pédagogiques de ce cours sont :

→ Comprendre le rôle des boucles et apprendre les différentes structures itératives utilisées en algorithmique.

→ Ecrire des algorithmes pour résoudre des problèmes nécessitant l'utilisation des structures itératives.

1. La boucle Pour

La boucle ***Pour*** permet d'utiliser un compteur (indice) pour contrôler le nombre d'itérations ou de répétitions d'un bloc d'instructions. Le compteur ou l'indice est caractérisé par :

→ sa valeur initiale (V_{init}),

→ sa valeur finale (V_{fin}),

→ son pas de variation :

- incrémentation (++) : si la valeur initiale est **inférieure** à la valeur finale,
- décrémentation (--) : si la valeur initiale est **supérieure** à la valeur finale.

Syntaxe

Pour (compteur ← V_{init} à V_{fin}) ***faire***
    Bloc d'instructions ;
Finpour

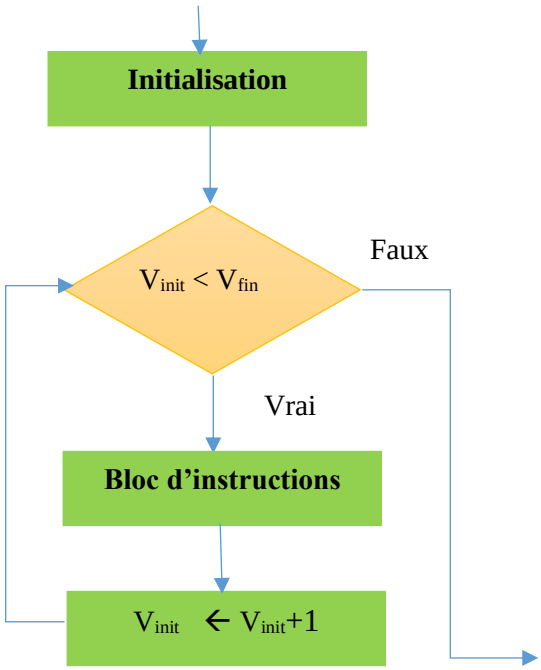
Syntaxe en C

```
for  (compteur = Vinit; compteur = Vfin; compteur=compteur+pas)
{
Bloc d'instructions ;
}
```

Scénario d'exécution

Pour compteur allant de valeur initiale jusqu'à la valeur finale, les instructions sont exécutées itérativement.

Organigramme



Exemple

En utilisant la boucle ***Pour***, écrire un algorithme qui permet de calculer la somme S = 1+ 2 + 3 + … + n, la valeur de n donné par l'utilisateur.

Algorithme Somme ;
Variables n, i, S : entier ;
Début
Ecrire (" Donner un nombre ") ;
Lire (n) ;
S ← 0 ; /* initialisation*/

Pour i ← 1 à n ***faire***
S ← S + i ;
Finpour
Ecrire ("La somme = ", S) ;
Fin

2. La boucle Tant que

Cette boucle permet de répéter un traitement tant qu'une condition est satisfaite. Dans cette structure itérative, on commence, premièrement, par l'évaluation d'une condition, si elle est vérifiée, le traitement est exécuté, sinon le programme quitte la boucle.

Syntaxe

Tant que *(Condition)* ***faire***
Bloc d'instructions ;
Fintque

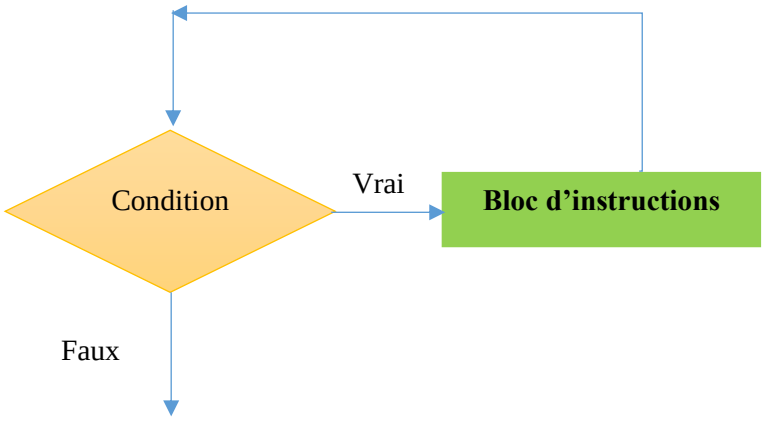
Syntaxe en C

```
while (Condition)
{
   Bloc d'instructions ;
}
```

Scénario d'exécution

Tant que la condition est vérifiée c'est-à-dire vraie l'exécution des instructions de la boucle tant que est répétée, jusqu'à ce que la condition devienne fausse.

Organigramme



Exemple

L'exemple précédent avec la boule Tant que.

```
Algorithme Somme ;
Variables n, i, S : Entier ;
Début
Ecrire (" Donner un nombre ") ;
Lire (n) ;
S ← 0 ;   /* initialisation*/
i ← 1 ;
Tant que (i<=n) faire
S ← S + i ;
i←i+1 ;
Fintque
Ecrire ("La somme = ", S) ;
Fin
```

3. La boucle Répéter

Le principe de cette boucle consiste à répéter l'exécution des instructions jusqu'à ce que la condition soit satisfaite. A la différence de la structure tant que, les instructions de la boucle répéter sont exécutées **au moins une fois** car l'exécution des instructions précède l'évaluation de la condition.

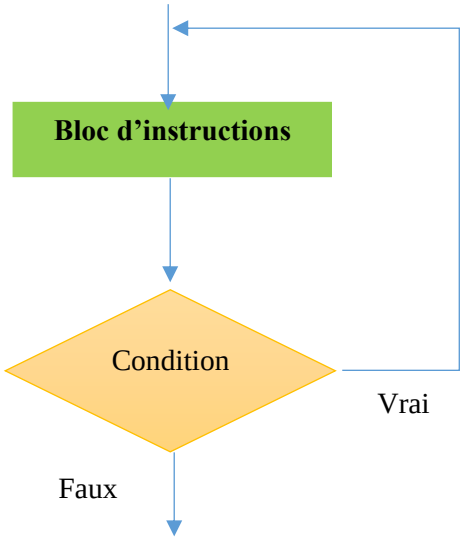
Syntaxe

```
Répéter
Bloc d'instructions ;
Jusqu'à (Condition) ;
```

Syntaxe en C

```
do {
Bloc d'instructions ;
}
while ( Condition ) ;
```

Organigramme



Exemple

L'exemple précédent avec la boule Répéter.

```
Algorithme Somme ;
Variables n, i, S : Entier ;
Début
Ecrire (" Donner un nombre ") ;
Lire (n) ;
S ← 0 ;   /* initialisation*/
i ← 1 ;
Répéter
S ← S + i ;
i←i+1 ;
Jusqu'à (i>n)
Ecrire ("La somme = ", S) ;
Fin
```

Conclusions sur les boucles

→ La boucle ***Pour*** permet d'exprimer les itérations contrôlées par un compteur. Si on connaît le point de départ (valeur initiale) et le point d'arrivée (valeur finale) et le pas de variation qui est constant, il est préférable d'utiliser la boucle Pour.

→ La boucle ***Répéter*** permet d'exprimer les itérations dans lesquelles la première exécution du corps de la boucle (Bloc d'instructions) n'est pas soumise à une condition.

→ La boucle ***Tant que*** permet d'exprimer toute itération. Elle peut remplacer les deux autres boucles ***Pour*** et ***Répéter***.

4. Les boucles imbriquées

Les instructions d'une boucle peuvent être à leur tour des instructions itératives ou boucles, on parle ici de boucles imbriquées, où la boucle contient elle-même une autre boucle.

Exemple

```
Pour (i allant de 1 à 3) faire
  Pour (j allant de 1 à 2) faire
      Ecrire ("i =", i);
     Ecrire ("j =", j);
  finpour
finpour
```

Scénario d'exécution

À chaque étape dans la boucle 1, le programme va exécuter la boucle intérieure (boucle 2) jusqu'à la fin avant de passer à l'étape suivante de la boucle 1, et ainsi de suite jusqu'à la fin des deux boucles. Alors, le résultat de l'exemple précédent est :

Boucle 1 : i= 1 ; *Boucle 2* : j=1 ; j=2 ;

Boucle 1 : i=2 ; *Boucle 2* : j=1 ; j=2 ;

Boucle 1 : i=3 ; *Boucle 2* : j=1 ; j=2 ;

5. L'utilisation de la notion de compteur dans les boucles

Généralement, le compteur est une variable dont la valeur est incrémentée par 1 à chaque étape de la boucle. Il permet donc de compter le nombre des itérations c'est-à-dire le nombre de fois que la boucle est exécutée.

Dans la boucle « ***pour »***, l'indice joue le rôle d'un compteur, tandis que, dans les structures itératives « ***répéter…jusqu'à*** » et « ***tant que …faire*** », nous pouvons l'utiliser comme suit :

compteur ← 0 ; /*initialisation*/ ***Tant que*** **(**Condition) ***faire*** Bloc d'instructions ; compteur ← compteur +1 ; ***Fintque***	compteur ← 0 ; /*initialisation*/ **Répéter** Bloc d'instructions ; compteur ← compteur +1 ; **Jusqu'à (**Condition)

N.B : il faut toujours initialiser le compteur avant l'utiliser.

6. Exercice d'application

En utilisant les trois boucles, écrire un algorithme pour calculer le factorielle d'un nombre donné par l'utilisateur.

Algorithme	Programme C
Pour	

***Algorithme* factorielle ;**
***Variables* i ,n, F : entier ;**
Début
***Ecrire* ("Donner un nombre ") ;**
***Lire* (n) ;**
F← 1 ;
Pour* i←1 à n *faire
F← F*i ;
Finpour
***Ecrire* (n, "! =", F) ;**
Fin
	#include <stdio.h>
	main()
	{
	int n, i, F;
	printf ("Donner un nombre \n");
	scanf ("%d",&n);
	F=1;
	For (i=1;i<=n;i++)
	{
	F=F*i;
	}
	printf("%d!=%d",n,F);
	}

Tant que	

***Algorithme* factorielle ;**
***Variables* i ,n,F : entier ;**
Début
***Ecrire* ("Donner un nombre ") ;**
***Lire* (n) ;**
F← 1 ;
i←1 ;
Tant que (*i<=n) *faire
F← F*i ;
i← i+1 ;
Fintque
***Ecrire* (n, "! =", F) ;**
Fin
	#include <stdio.h>
	main()
	{
	int n,i,F;
	printf("Donner un nombre \n");
	scanf("%d",&n);
	F=1; i=1;
	while(i<=n)
	{
	F=F*i;
	i++;
	}
	printf("%d!=%d",n,F); }

Répéter	

***Algorithme* factorielle ;**
***Variables* i ,n,F : entier ;**
Début
***Ecrire* ("Donner un nombre ") ;**
***Lire* (n) ;**
F← 1 ;
i←1 ;
Répéter
F← F*i ;
i← i+1 ;
***Jusqu'à* (i>n)**
***Ecrire* (n, "! =", F) ;**
Fin
	#include <stdio.h>
	main()
	{
	int n,i,F;
	printf("Donner un nombre \n");
	scanf("%d",&n);
	F=1;
	i=1;
	do
	{
	F=F*i;
	i++;
	}while(i<=n);
	printf("%d!=%d",n, F);
	}

TD3 : Les structures itératives

Objectifs pédagogiques

- Comprendre et maîtriser les différents types de structures itératives : la boucle pour, la boucle tante que et la boucle répéter ;
- Découvrir quel type de boucle faut –il utiliser pour résoudre un problème donné.

Exercice n°1

Ecrire un algorithme qui demande un nombre de départ, et qui affiche par la suite les dix nombres suivants. Par exemple, si l'utilisateur donne le nombre 15, le programme affichera les nombres de 16 à 25.

Exercice n°2

Ecrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise par la suite quel était le plus grand parmi les 20 nombres donnés et à quelle position a été saisi.

Exercice n°3

Réécrire l'algorithme précédent, mais cette fois, on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

Exercice n°4

Ecrire un algorithme qui permet de saisir un entier et qui l'affiche à l'envers. Par exemple, si l'utilisateur donne le nombre 123456, le programme affichera 654321. Pour cela, il faut utiliser les fonctions Div et Mod.

Exercice n°5

Ecrire un algorithme qui permet de convertir un entier N écrit sous forme binaire en sa valeur décimale. *Exemple* : N $=(10111010)_2 = (186)_{10}$

Exercice n°6

Ecrire un algorithme qui affiche tous les diviseurs d'un nombre entier positif N donné par l'utilisateur.

Exercice n°7

En utilisant les trois types de boucles, écrire un algorithme qui calcule la somme des nombres pairs compris dans l'intervalle [1, N], N un entier positif donné par l'utilisateur.
Somme = 2 + 4 + 6 + 8 +…….+ N (si N est pair ou (N-1) si N est impair).

Exercice n°8

Ecrire un algorithme qui calcule la somme des factorielles. S= 1 !+2 !+3 !.....

Exercice n°9

Ecrire un algorithme qui permet de calculer la somme S= 1+2-3+4+5-6+7+8-9……

TP3 : Les structures itératives en C

Objectifs pédagogiques

- Comprendre le fonctionnement et maîtriser la syntaxe des différents types des instructions itératives (les boucles) en C ;
- Savoir choisir la boucle la plus naturelle pour résoudre un problème donné.

Exercice n°1

Ecrire un programme C qui génère la table de multiplication d'un nombre entier positif N donné par l'utilisateur.

Exercice n°2

- En utilisant les boucles while, do-while et for, écrire un programme C qui lit N nombres entiers au clavier et qui affiche leur somme, leur produit et leur moyenne.
- Laquelle des trois boucles est la plus naturelle pour ce problème ?

Exercice n°3

Ecrire un programme C qui permet de vérifier si un entier positif N est parfait (égal à la somme de ses diviseurs) ou pas. Exemple : 28= 1+2+4+7+14, 28 est un nombre parfait.

Exercice n°4

Ecrire un programme C qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

Exercice n°5

Ecrire un programme C qui permet de calculer la valeur approchée :

$e^x \cong 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$ (x : un nombre réel, n : un entier positif).

Exercice n°6

Ecrire un programme C qui calcule le PGCD (Plus Grand Commun Diviseur) de deux entiers positifs A et B.

Exercice n°7

Ecrire un programme C qui permet de calculer la somme des chiffres d'un nombre entier N donné par l'utilisateur. Exemple : Somme_chiffres (123456)= 21.

Pour résoudre ce problème, on doit utiliser les opérations *N % 10* et *N / 10* jusqu'à ce que N devienne nul.

Exercice n°8

Ecrire un programme C qui permet de calculer la valeur de l'expression suivante :

E = (1+2)*(1+2+3)*(1+2+3+4)*…*(1+2+3+…+(N-2)+(N-1)+N) / (N≥2).

Exercice n°9

Modifier le programme précédent pour afficher les tables de multiplication entre un intervalle [N, 10] (N<10) en utilisant la boucle while. Exemple : si N=5 le programme va afficher les tables de multiplication de 5, 6, 7, 8, 9 et 10.

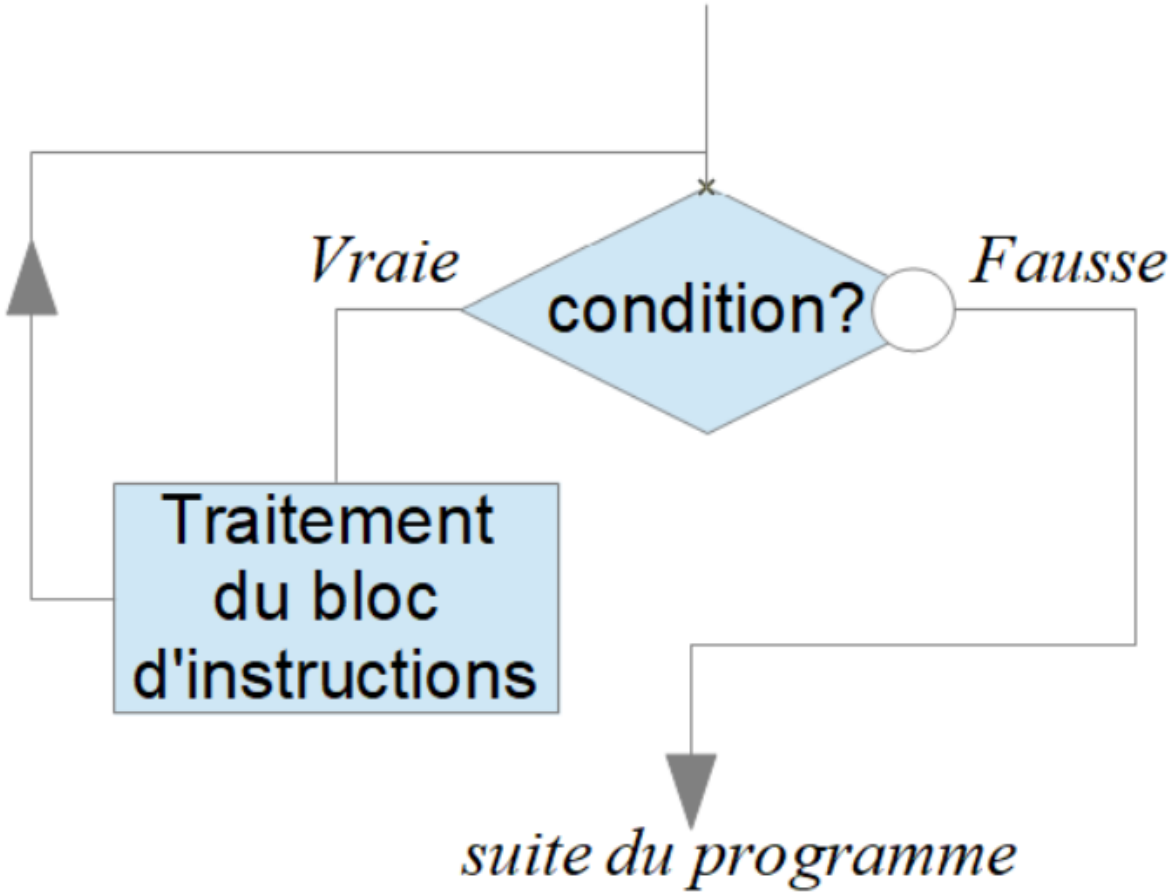


Figure 11 : Le principe de l'instruction itérative

Chapitre 5 « Les tableaux et les chaînes de caractères»

Chapitre 5 : Les tableaux et les chaînes de caractères

Pourquoi les tableaux ?

Si on veut calculer les moyennes de 30 étudiants et faire leur classement. Donc, la première solution proposée est l'utilisation d'une boucle pour calculer les moyennes, comme suit :

Pour i ← 1 à 30 ***faire***
Ecrire (" Donner les notes de l'étudiant N°", i) ;
Lire (Notes) ;
Moyenne ← $\frac{\sum notes}{nombre\_matières}$;
Finpour

Le problème ici !! On ne peut pas effectuer le classement, parce qu'on ne garde pas les moyennes précédentes et la variable moyenne contient uniquement la dernière valeur (la moyenne du dernier étudiant).

La solution !!! L'utilisation d'une structure de données qui permet de stocker toutes les moyennes calculées afin de les réutiliser pour faire le classement,

Le tableau comme une collection homogène de données représente la solution la plus convenable.

Les objectifs pédagogiques de ce cours sont :

→ Comprendre le rôle des tableaux, la différence entre les tableaux et les matrices et saisir les différentes opérations sur les tableaux.

→ Ecrire des algorithmes pour résoudre des problèmes nécessitant l'utilisation des tableaux/matrices.

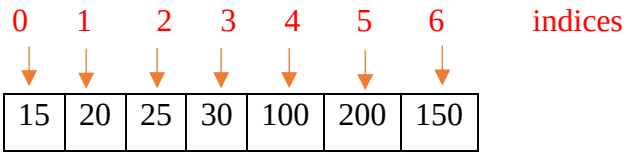
1. Le type tableau

Une variable simple peut stocker une seule valeur de type simple (entier , réel…), par contre, le tableau est une structure composée homogène T permet de stocker dans une zone mémoire contigüe plusieurs valeurs T[i] repérés par un index i. Les tableaux vérifient généralement les propriétés suivantes :

→ tous les éléments ont le même type de données ;

→ le nombre d'éléments stockés est fixé.

Exemple



→ 15 est l'élément d'indice 0, il est noté par T[0].

→ 100 est l'élément d'indice 4, il est noté par T[4].

Syntaxe

< *identificateur* >: *tableau*[*Taille*] < *Type* >;

- L'identificateur représente le nom du tableau ;
- Taille est le nombre des éléments du tableau ;
- Type est le type des éléments de ce tableau.

Exemple

Tab : *tableau* [1..10] *entier* ;

Syntaxe en C

```
< type >< identificateur > [taille] ;
```

Exemple

```
int tab1 [20] ;
float tab2 [10]
```

1.1.Accès aux éléments d'un tableau

→ L'accès à un élément d'une variable Tab de type tableau se fait par une variable indicée qui s'écrit : Tab[Expression]

Expression : est une expression de type entier qui détermine l'indice (le rang, la position) de l'élément sélectionné dans le tableau. Exemple **:** Moyenne [8] : désigne l'élément de rang 8 dans le tableau *Moyenne*.

→ L'affectation d'une valeur à un élément d'un tableau se fait par l'instruction suivante :

< *identificateur* > *[Indice]* ← < *expression* >*;*

Exemple : Moyenne[8] ← 12.5 ;

1.2. Manipulation des tableaux

Chaque élément dans le tableau se comporte comme une variable, peut être lu à partir du clavier, affiché à l'écran, utilisé dans une comparaison ou dans une affectation.

→ La lecture d'un tableau consiste à affecter une valeur pour chaque élément à partir d'un clavier, afin de compléter cette tâche et parcourir tous les éléments d'un tableau, nous utilisons les boucles.

Syntaxe

```
Pour i ←1 à n faire
Lire(T[i]) ;
finpour
```

→ L'écriture d'un tableau consiste à afficher le contenu des éléments d'un tableau.

Syntaxe

```
Pour i ←1 à n faire
Ecrire(T[i]) ;
finpour
```

Exemple **:** Ecrire un algorithme qui calcule la moyenne de *30* étudiants.

```
Algorithme Moyenne ;
Variables nombre_matieres, i, j : entier ;
          Note, somme : réel ;
          Moyenne : tableau [1..30] réel ;
Début
Ecrire (" Donner le nombre des matières") ;
Lire (nombre_matieres) ;
Pour i ←1 à 30 faire
/* nous allons utiliser les boucles imbriquées*/
Somme ← 0 ;
Pour j ←1 à nombre_matieres  faire
Ecrire (" Donner les notes de l'étudiant N°",i) ;
Lire ( Note) ;
Somme ← Somme +Note;
```

```
Finpour
Moyenne [i] ← Somme/ nombre_matieres ;
Ecrire (" La moyenne de l'étudiant N°",i, "est", Moyenne [i]) ;
Finpour
```

2. Les tableaux multidimensionnels (matrices)

Lorsqu'un traitement utilise plusieurs tableaux à une dimension ont le même nombre d'éléments (même dimension) et subissent le même traitement, dans ce cas, nous pouvons utiliser un seul tableau à deux dimensions appelé **matrice**.

La matrice possède un identifiant unique. Chaque élément est identifié par deux indices :

- → indice de ligne.
- → indice de colonne.

Syntaxe

```
< identificateur >: tableau[nb_l, nb_c] < Type >;
```

Exemple

```
Mat : Tableau [1..4, 1..4] réel
```

Syntaxe en C

```
< type >< identificateur > [taille1][taille2] ;
```

Exemple

```
int Mat1 [6] [6];
float Mat2 [8] [11] ;
```

2.1. Accès aux éléments d'une matrice

L'accès à un élément d'une variable Mat de type matrice (tableau à deux dimensions) se fait par une variable indicée dont la syntaxe est : Mat [Exp1, Exp2]. Exp1 et Exp2 sont deux expressions, de type entier, qui déterminent le numéro de ligne et numéro de colonne de l'élément sélectionné dans le tableau.

Exemple : Notes [5, 3] : l'élément qui se trouve à la 5ième ligne et la 3ième colonne dans la matrice Notes. Cet élément désigne, par exemple, la note du 5ième étudiant à la 3ième matière.

2.2. Manipulation des matrices

Pour lire ou écrire les éléments d'une matrice, nous pouvons utiliser deux boucles imbriquées afin de parcourir tous les éléments. L'une des deux boucles est utilisée pour parcourir les lignes et l'autre pour les colonnes.

Syntaxe

```
/* n : nombre de lignes, m : nombre de colonnes */
Pour i ←1 à n faire
  Pour j ← 1 à m faire
      Lire(Mat[i, j]);
  Finpour
Finpour
```

```
Pour i ←1 à n faire
  Pour j ← 1 à m faire
      Ecrire(Mat[i, j]);
  Finpour
Finpour
```

Exemple : En utilisant les matrices, réécrire l'algorithme de l'exemple précédent avec nombre_matières =10 ;

```
Algorithme Moyenne ;
Variables   Notes : Tableau [ 1..30, 1..10 ] réel ;
            i, j : entier ;
            Somme, Moyenne : réel ;
Début
Pour i ← 1 à 30 faire
  Pour j ← 1 à 10 faire
      Ecrire ("Donner la note de l'étudiant N° ", i, " à la matière ", j )
      Lire ( Notes[ i, j ] )
  Finpour
Finpour
```

```
/* Calcul des moyennes*/
Pour i ← 1 à 30 faire
   Somme ← 0
  /*Calcul de la somme des notes de l'étudiant i*/
   Pour j ← 1 à 10 faire
     Somme ← Somme + Notes[ i, j ]
  Finpour
Moyenne ← Somme / 10 ;
Ecrire ("Moyenne de l'étudiant ", i , " = ", Moyenne) ;
Finpour
Fin
```

3. Les chaînes de caractères

Une chaîne de caractères est une suite ordonnée de caractères, entre guillemets (""), elle est représentée en algorithmique par le mot-clé chaîne. Cependant, en C, la chaîne de caractères est représentée par un tableau de caractères (char), on peut accéder à chaque caractère par son indice c'est-à-dire sa position au sein de la chaîne.

Syntaxe

```
< identificateur > : chaîne;
```

Syntaxe en C

```
char < identificateur > [< Longueur >];
```

En informatique un caractère est représenté par son code ASCII ("American Standard Coding for Information Interchange").

3.1. La manipulation des chaînes de caractères

→ *Longueur*

La longueur d'une chaîne de caractères est le nombre de caractères qu'elle contient. Exemple : Long ("Salut") =5.

→ *La concaténation*

La concaténation permet de combiner (assembler) deux chaînes de caractères, c'est-à-dire placer une chaîne de caractère à la suite d'une autre, en utilisant l'opérateur '+'.

Exemple

- Chn1 <- "Turbo" ;

- Chn2 <- "C++" ;
- Chn3 <- Chn1+"+Chn2;

La variable Chn3 va contenir "Turbo C++"

→ *La comparaison* (=, <>, <, > <= , >=)

Il est possible d'effectuer une comparaison entre deux chaînes de caractères, le résultat est du type booléen. La comparaison se fait caractère par caractère, de gauche à droite selon le code ASCII.

Exemples :

- 'baobab' < 'sapin' car le code ASCII de 'b' est inférieur au code ASCII de 's'.
- 'baobab' > 'banania' car le code ASCII de '0' est supérieur au code ASCII de 'n', les deux premiers caractères sont identiques.
- '1999' > '1998' car le code ASCII de '9'est supérieur au code ASCII de '8', les trois premiers caractères sont identiques.

  La table suivante représente le code ASCII pour les différents caractères

3.2. La table de code ASCII (7bits)

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

4. Exercices d'application

Exercice n° 1:

Ecrire un algorithme qui permet de calculer la somme, le produit et la moyenne des éléments d'un tableau.

```
Algorithme somme_produit_moyenne;
Variables
        Tab   : tableau [1..100] réel ;
        N, i   :  entier ;
        S, P, Moy :  réel ;
Début
   Ecrire ("Donner la taille du tableau <=100 ") ;
   Lire (N) ;
        Si N=0 alors
           Ecrire ("le tableau est vide ") ;
        Sinon
         Ecrire ("Enter les éléments du tableau ") ;
        Pour i ← 1 à N faire
          Lire (Tab [i])
       Finpour
            S ← 0 ; /* initialisation par l'élément neutre*/
            P ← 1 ;
            Pour i ← 1 à N faire
                S ←  S+Tab[i] ;
                P ←  P * Tab[i] ;
             Finpour
         Moy ←    S/N ;
        Ecrire ("la somme des éléments du tableau est : ",S);
        Ecrire ("le produit des éléments du tableau est : ",P);
        Ecrire ("la  moyenne des éléments du tableau est : ",M);
      Finsi
Fin
```

Exercice n° 2 :

Ecrire un algorithme qui permet de consulter un élément d'un tableau.

```
Algorithme consultation ;
Variables
        Tab   : tableau [1..100] réel ;
        N, Pos   :  entier ;
Début
Ecrire ("Donner la taille du tableau <=100 ") ;

  Lire (N) ;
        Si N=0 alors
            Ecrire ("le tableau est vide ") ;
        Sinon
          Ecrire ("Enter les éléments du tableau ") ;
         Pour i ← 1 à N faire
          Lire (Tab [i])
        Finpour
            Ecrire ("entrer l'indice de l'élément à consulter : ") ;
            Lire (Pos) ;
            Si (Pos<1) ou (Pos>N) alors
                Ecrire ("Position hors limites du tableau ")
            Sinon
                Ecrire ("l'élément à consulter est : ", Tab[Pos]) ;
             Finsi
        Finsi
Fin
```

TD4 : Les tableaux & les matrices

Objectifs pédagogiques

- Manipuler les tableaux, les matrices et les chaînes de caractères ;
- Comprendre l'utilité des tableaux (unis et multidimensionnels), et découvrir quel est le type le plus approprié pour résoudre un problème donné.

Exercice n°1

Ecrire un algorithme qui permet de chercher toutes les occurrences d'un élément dans un tableau qui contient des éléments de type entier.

Exercice n°2

Ecrire un algorithme qui permet de calculer la somme des éléments d'un tableau.

Exercice n°3

Ecrire un algorithme qui permet de trier par ordre croissant les éléments d'un tableau du type réel.

Exercice n°4

Ecrire un algorithme qui permet d'inverser les éléments, de type caractère, d'un tableau.

Exercice n°5

Ecrire un algorithme qui permet d'afficher le nombre de valeurs négatives et le nombre de valeurs positives dans un tableau donné par l'utilisateur.

Exercice n°6

Ecrire un algorithme qui permet d'éclater un tableau des entiers T en deux tableaux T1 et T2 contenant respectivement les nombres pairs et impairs de T.

Exercice n°7

Ecrire un algorithme qui détermine si un mot est un palindrome. Un mot palindrome se lit de gauche à droite et de droite à gauche, exemple : RADAR, ELLE, ICI.

Exercice n°8

Soit une matrice Mat(n, m) d'entiers avec $n \leq 20$ et $m \leq 30$, écrire un algorithme qui permet de:

- Calculer et sauvegarder la somme de chaque colonne,
- Déterminer la position Jmin de la somme minimale et la position Jmax de la somme maximale,
- Permuter les deux colonnes d'indices Jmin et Jmax de la matrice si Jmin > Jmax.

Exercice n°9

Soit une matrice Mat (n, m) de caractères ($n \leq 30$ et $m \leq 50$). Ecrire un algorithme qui :

- Recherche un élément (caractère) dans la matrice Mat.
- Calcule le nombre de voyelles appartenant à la matrice Mat.

TP4 : Les tableaux & les matrices en C

Objectifs pédagogiques

- Manipuler les tableaux, les matrices et les chaînes de caractères en langage C, en matière de déclaration et traitement.

Exercice n°1 :

Ecrire un programme C qui lit et remplit un tableau Tab du type entier avec une dimension maximale= 100 éléments ;
En utilisant deux autres tableaux, copier tous les éléments strictement positifs dans le premier, nommé Tab_POS, et tous les éléments strictement négatifs dans le deuxième tableau, nommé Tab_NEG.

Exercice n°2

Écrire un programme C qui demande 20 nombres entiers à l'utilisateur, les range dans un tableau, ensuite, il détermine les éléments le plus grand et le plus petit. Afficher le résultat.

Exercice n°3

Ecrire un programme C qui permet de saisir un mot et l'afficher dans l'ordre inversé.
Rappel : un tableau de caractères (chaîne de caractères), se termine par le caractère spécial : '\0'.

Exercice n°4

On imagine une visite médicale en deux parties. Premièrement, tous les patients se présentent pour mesurer leur taille, puis repassent, dans le même ordre, au pesage. Ecrire un programme C qui permet de :

a. Enregistrer dans un tableau la taille t en mètres de tous les patients qui se présentent, l'arrêt se fait par la saisie d'un nombre <= 0.
b. Afficher le nombre total de patients.
c. Pour chaque patient précédemment mesuré, enregistrer la masse m en kilogrammes dans un deuxième tableau.
d. Calculer et afficher la taille moyenne et le poids moyen des patients.
e. Pour chaque patient, calculer l'indice de masse corporelle : IMC = m/t^2 et afficher :
    - si IMC < 18.5, "pas assez" ;
    - si IMC > 25, "trop" ;
    - sinon, "normal".

Exercice n°5

Ecrire un programme C qui permet de :
Lire et remplir un tableau Tab du type entier avec une dimension maximale= 100.

Effacer toutes les occurrences de la valeur 0 dans le tableau Tab et tasser les éléments restants. Afficher le tableau résultant.

Exercice n°6

Ecrire un programme C qui demande à l'utilisateur de saisir son nom, ensuite compte les caractères et affiche leur nombre.

Exercice n°7

Soit une matrice carrée Mat (n, n) d'entiers (n≤50). Ecrire un programme C qui :

1- Calcule la trace de la matrice Mat. (*La trace est la somme des éléments de la diagonale principale).*

2- Détermine la valeur maximale dans la diagonale.

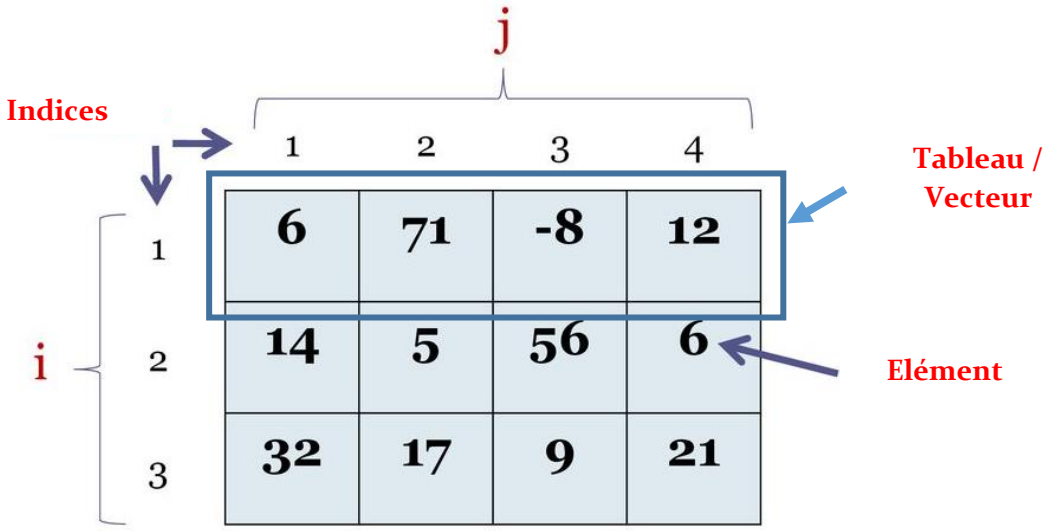


Figure 12 : Tableau vs. Matrice

Chapitre 6 « Les types personnalisés»

Chapitre 6 : Les types personnalisés

Pourquoi les types personnalisés ?

Dans certains problèmes nous pouvons trouver des données fortement liées les unes avec les autres, le traitement de l'une doit être influé directement sur celui de l'autre. Il convient donc de regrouper ces données dans un ensemble indivisible d'informations.

Exemple : on veut enregistrer des informations sur les clients pour les réutiliser ultérieurement. Pour chaque client, les informations sont : le nom, le prénom et l'âge. Au lieu d'enregistrer chaque information indépendamment, on peut les regrouper dans une seule structure comme suit :

Structure de données

Type Client
  Nom: chaine
  Prenom: chaine
  Age: entier
FinType

Dans ce cours, nous nous intéressons principalement aux deux types personnalisés : énumération et enregistrement.

1. Enumération (Type énuméré)

Les énumérations permettent de définir un type par la liste des valeurs qu'il peut prendre.

Le type énuméré permettant de représenter des objets qui peuvent prendre leur valeur dans une liste finie et ordonnée, autrement dit, le type énuméré permet d'associer à un type un ensemble de valeurs ordonnées suivant leur ordre de déclaration.

1.1. Déclaration d'un type énuméré

Pour déclarer un type énuméré, on utilise un nom pour l'identification de l'objet (identificateur), suivi par un ensemble de valeurs entre parenthèse, comme suit :

Syntaxe :

Type
Identificateur = (Val_1, Val_2, Val_3, ..., Val_N) ;

- *Identificateur* : le nom de l'énumération ;
- *Val_1, Val_2, Val_3, ..., Val_N* , sont des valeurs que peut prendre *Identificateur* .
- *Val_1, Val_2, Val_3, ..., Val_N*, sont ordonnées, c.-à-d. $Val_1 < Val_2 < Val_3 < \ldots.. < Val_N$

Syntaxe en C :

```
enum identificateur {Val1, Val2, Val3, …..., ValN };
```

Exemples

- **Type** Semaine = (dimanche, lundi, mardi, mercredi, jeudi, vendredi, samedi) ;
- **Type** Couleur = (rouge, vert, bleu) ;
- **Type** Sexe = (masculin, feminin);
- **Type** Voyelle = (A, E, I, O, U) ;

1.2. Utilisation d'un type énuméré

On utilise le type énuméré pour déclarer des variables de ce type, cette variable ne peut prendre qu'une des valeurs données entre parenthèses.

Syntaxe

Variable Nom_Var : *Nom_Type_Enumere ;*

Syntaxe en C

```
enum Nom_Type_Enumere  Nom_Var;
```

Exemple

- **Variable** jour : Semaine ;
- **Variable** C : couleur ;
- **Variable** S : Sexe ;
- **Variable** V : Voyelle ;

Exemple en C

```
enum semaine {Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi,
Samedi};
main()
{
    enum semaine jour;
    jour = Mercredi;
    printf("Le jour %d",jour+1);
 }
```

Le programme affichera « *Le jours 4 »*

2. Enregistrements (Structures)

Nous avons vu dans le chapitre précèdent que le type tableau permet de représenter (enregistrer) plusieurs valeurs de même type. Tandis que, l'enregistrement (structure) est une structure de données permettant de regrouper dans une seule entité un ensemble de données de types différents associées à un même et seul objet.

L'enregistrement est identifié par un nom et un ensemble de propriétés appelées **champs**. Chaque champ est identifié par un nom qui permet d'y accéder directement et un type. Le type peut être simple ou structuré.

2.1. Déclaration d'un enregistrement

Pour déclarer une structure nous pouvons utiliser la syntaxe suivante :

```
TYPE Nom_enregistrement = Enregistrement
        Nom_champ1 : Type1
       Nom_champ2 : Type2
         …
      Nom_champN : TypeN
FinEnregistrement
```

NB : S'il y a des champs ont le même type, on peut les déclarer ensemble en les séparant par des virgules.

Pour déclarer une variable de type enregistrement, on peut utiliser la déclaration suivante :

```
Variable Nom_variable : Nom_enregistrement
```

Syntaxe en C

```
typedef struct Nom_enregistrement
 {Type1 Champ1 ;
   Type2 Champ2 ;
   …….
 TypeN ChampN ;
 } Nom_enregistrement;
```

Exemple

Les informations d'un étudiant : nom, prénom, age, sexe, Moyenne du BAC peuvent être regroupées dans un enregistrement

```
Type Etudiant = Enregistrement
        Nom : chaine
        Prenom : chaine
       Age :entier
       Sexe : caractère // 'M' : Masculin, 'F' : Féminin
        MoyenneBac : réel
FinEnregistrement
```

```
Variables  Etudiant1, Etudiant2 : Etudiant // Deux variables de type Etudiant
           e1, e2, e3 : Etudiant // Trois variables de type Etudiant
```

2.2.Accès aux champs d'un enregistrement

On peut manipuler les champs d'un enregistrement champs par champs. L'accès aux champs d'un enregistrement ce fait par la précision du nom de la variable de type enregistrement suivie du nom du champ séparé par un point(**.**) :

Nom_variable. Nom_champs ;

Exemple : pour modifier l'âge de l'Etudiant1 en utilisant l'affectation on doit écrire :

Etudiant1.Age ←22 ;

Le point indique le chemin d'accès : On accède d'abord à la variable Etudiant1 puis on sélectionne le champ Age.

2.3.Structures imbriquées

Un enregistrement peut être imbriqué dans une structure de type tableau ou enregistrement, comme il peut avoir des champs de type structuré quelconque (ex. tableau). La notation utilisée pour sélectionner les champs reste la même, en utilisant le point.

→ Tableaux d'enregistrements

Il est possible de déclarer un tableau dont les éléments sont de type enregistrement. Dans ce cas, on définit d'abord le type enregistrement, puis on déclare un tableau dont les éléments sont de ce type d'enregistrement. Comme suit :

```
Type Nom_enregistrement = Enregistrement
        Nom_champ1 : Type1
        Nom_champ2 : Type2
         ...
        Nom_champN : TypeN
FinEnregistrement
Variable  Nom_tableau : tableau [ 1 .. N ] Nom_enregistrement ;
```

Pour accéder à une case dans le tableau, on utilise les crochets [ ], puis on accède au champ en utilisant le point.

Exemple : pour sélectionner le troisième champ du deuxième élément du tableau on utilise la syntaxe :

Nom_tableau[2] . Nom_champ3

Exemple

On veut déclarer un tableau d'enregistrements pour manipuler les informations de 50 étudiants.

```
Type Etudiant = Enregistrement
        Nom : chaine
        Prenom : chaine
       Age :entier
       Sexe : caractère
        MoyenneBac : réel
FinEnregistrement
Variable Tab : tableau [ 1 .. 50 ] Etudiant ;
```

Pour modifier les champs de l'étudiant 10, on peut écrire:

Tab[10] . Nom ← "Xxxxx"

Tab[10] . Prenom ← "Yyyyy"

Tab[10] . Age ← 17

Tab[10] . Sexe ← 'M'

Tab[10] . MoyenneBac ← 12.05

N.B (manipulation des enregistrements) : Toute opération sur les enregistrements doit être réalisée séparément : La lecture, l'écriture, la comparaison ne peuvent se faire globalement, il faut lire, écrire ou comparer chaque champ individuellement.

Exemple en C

```
typedef struct date {
    int jour;
    int mois;
    int annee ;
    } date;
```

```
Initialisation
    date D = {0, 0,0};
Utilisation
    {  D.jour = 22;
       D.mois = 3;
       D.annee =2004 ;
    }
```

3. Autres possibilités de définition de type : type intervalle

Le type intervalle est un ensemble de valeurs ordonnées définit à partir des types simples et caractérisé par sa valeur minimale (extrémité inférieure) et sa valeur maximale (extrémité supérieure).

Syntaxe de déclaration

Type Nom_Type_Intervalle = [Val_Min .. Val_Max] ;
Variable Nom_Var : Nom_Type_Intervalle ;

Exemples

- **Type** Chiffres = [0 . . 9] ;
- **Type** Lettres = [’C’ . . ’K ’];
- **Type** Note = [0..20] ;
- **Type** Mois = [1..12] ;

→ On peut utiliser le type intervalle pour déclarer une ou plusieurs variables.

Variables Note_Analyse, Note_ASD: Note;

Les variables Note_Analyse et Note_ASD peuvent prendre n’importe quelle valeur de 0 à 20 seulement.

→ Le type intervalle est un sous ensemble de l'ensemble des valeurs d'un type ordinal. Donc, toutes les opérations possibles sur le type de base sont possibles sur le type intervalle.

4. Exercices d'application

Exercice n° 1:

Créer des types intervalles heure, minute et seconde, puis un enregistrement « Temp » qui comporte ces intervalles.

Solution

```
Type Heure = [0..23];
Type Minute = [0..59];
Type Seconde = [0..59];
Type Temp = Enregistrement
        H : Heure;
        M : Minute;
        S: seconde;
FinEnregistrement
```

Exercice n° 2:

Créer un tableau qui contient 100 étudiants. Chaque étudiant s'identifié par le nom, le prénom, la moyenne, ses notes pour 9 matières ainsi que le résultat : "admis", "ajourné".

Solution

```
Type Mention = (admis, ajourné); /* type énuméré*/
Type  Etudiant = Enregistrement
        Nom, prénom :chaine
        Notes : tableau [1 ..9] réel
        Moyenne : réel
        Resultat :Mention
FinEnregistrement
Variable T_PV : tableau [1..100] Etudiant ;
```

TD5 : Les types personnalisés

Objectifs pédagogiques

- Comprendre l'utilité des types personnalisés et les manipuler pour résoudre des différents problèmes.

Exercice n°1

Définir un type TEMPS (enregistrement) qui contient les champs : heure, minute, seconde.

1. Ecrire un algorithme qui permet d'effectuer la somme T de deux durées T1 et T2 de type TEMPS.
2. Ecrire un algorithme qui permet de transformer un temps T de type TEMPS en un entier S qui exprime ce temps en secondes. Exemple : pour T = 2 heures 10 minutes 37 secondes, S = 7837 secondes.

Exercice n°2

Un nombre complexe C est défini par ses parties réelle **a** et imaginaire **b** (C **= a + bi)**.

Ecrire un algorithme qui lit deux nombres complexes C1 et C2 et qui affiche ensuite leur somme et leur produit.

Exercice n°3

Soit un enregistrement Ens défini par deux informations (champs):

- T_Pos un tableau d'entiers pouvant contenir au maximum 50 éléments ;
- N le nombre d'éléments du tableau T_Pos.

Soit une chaîne de caractères Ch, écrire un algorithme qui permet de chercher la chaîne "ab" et mettre dans un enregistrement de type Ens (dans le tableau T_Pos) toutes les positions de la chaîne dans la chaîne M.

Exemple : M = 'faabaababbaabrs'. Positions : 3, 6, 8, 12 => T_Pos [1]=3, T_Pos [2]=6, T_Pos [3]=8, T_Pos [4]=12. Nombre d'éléments : 4 => N=4.

TP5 : Les types personnalisés en C

Objectifs pédagogiques

- Manipuler les types personnalisés en langage C, en matière de déclaration et traitement.

Exercice n°1

Écrire un programme C qui définit trois structures Point (champs : X et Y), Cercle (Champs : X, Y, R) et Rectangle (Log, Lag). Le programme doit lire et afficher les champs respectifs des variables de type structure Point, Cercle et Rectangle (point p1, cercles c1, rectangles R1).

Exercice n°2

Une menuiserie industrielle gère un stock de panneaux de bois. Chaque panneau possède une largeur, une longueur et une épaisseur en millimètres, ainsi que le type de bois qui peut être un pin (code 0), un chêne (code 1) ou un hêtre (code 2).

1. Définir une structure panneau contenant toutes les informations relatives à un panneau de bois ;
2. Ecrire un programme C qui permet de
   - saisir et afficher un panneau de bois (saisie numérique pour le type de bois (ex : 0), affichage en caractères du type de bois (ex : pin) ;
   - Calculer le volume d'un panneau ((épaisseur*largeur*longueur)/10^9).

Exercice n°3

Soit les structures suivantes :

- Date définie par les trois champs : jour, mois et année
- Adresse définie par les champs : Numero, Rue, Commune, Wilaya, CodePostal
- Employé définie par les champs : NomPrenom, Residence, DateNaissance

Ecrire un programme C qui permet de saisir les informations d'un employé à partir du clavier et vérifier :

→ si un employé est né avant une année donnée ;

→ si un employé est résidant dans une wilaya donné.

Références Bibliographiques

[1]. ADECEC Cervioni . (2005). Composants d'un ordinateur. Formation Initiation à l'informatique et à internet.

[2]. AMAD, M. (2016). Algorithmique et Structures de Données, Cours et Travaux Dirigés. Université Abderrahmane Mira de Bejaia.

[3]. BELOUADHA, F.Z. Algorithmes et langage C. Université Mohammed V – Agdal Ecole Mohammadia d'Ingénieurs. Département Informatique

[4]. Berthet, D., & Labatut, V. (2014). Algorithmique & programmation en langage C vol.2 : Sujets de travaux pratiques. Istanbul, Turquie. Université Galatasaray, pp.258.

[5]. Berthet, D., & Labatut, V. (2014). Algorithmique & programmation en langage C-vol. 1. Istanbul, Turquie. Université Galatasaray, pp.232.

[6]. Berthet, D., & Labatut, V. (2014). Algorithmique & programmation en langage C vol.3 : Corrigés de travaux pratiques. Istanbul, Turquie. Université Galatasaray, pp.217.

[7]. BESSAA, B. (2017). Algorithmique, Exercices avec Solutions. https://www.coursehero.com/file/52170520/mi1an-algo-exercices-corriges-1pdf/

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

[9]. Delannoy, C. (1990). Apprendre à programmer en Turbo C.

[10]. Delest, M. (2007). Initiation à l'Algorithmique. Notes de cours. Université Bordeaux 1.

[11]. Helaoui, M. (2011). Travaux Dirigés : Algorithmique et Structure de Données. 10.13140/2.1.3800.9601.

[12]. Helaoui, M. (2019). ASDI 2019 2020 v6.pdf. https://www.researchgate.net/publication/337873900_ASDI_2019_2020_v6pdf

[13]. LANGLOIS, Ph. (2013). Programmation en C – Exercices. Université de Perpignan Via Domitia

[14]. Laurent Poinsot. Chap.I: Architecture de base d'un ordinateur. UniversitéParis13 InstitutGalilée.

[15]. M. DELEST. (2007). Initiation à l'Algorithmique. Notes de cours. Université Bordeaux 1.

[16]. Mohamed, E.M. (2013). Algorithmique. Université Mohammed V-Agdal, Faculté des Sciences –Rabat.

[17]. N'Diaye, L., Algo, C., & Sabbar, A. (2007). Algorithmique et structures de données.

[18]. Nicolas, F. (2011). Cours d'algorithmique BTS SIO première année.

[19]. Parreaux, J. Leçon 927: Exemples de preuves d'algorithmes: correction et terminaison