

# Programmation structurée

Support de TP et TD

# Parties

<b>Section 1 : Algorithmique.....</b>	<b>13</b>
<b>Partie 1.1 : Introduction à la programmation .....</b>	<b>17</b>
<b>Partie 1.2 : Introduction à l'algorithmique.....</b>	<b>25</b>
<b>Partie 1.3 : Algorithmique .....</b>	<b>33</b>
<b>Section 2 : Langage C.....</b>	<b>42</b>
<b>Partie 2.1 : Introduction au langage C .....</b>	<b>46</b>
<b>Partie 2.2 : Instruction de contrôle.....</b>	<b>58</b>
<b>Partie 2.3 : Tableau et adresse.....</b>	<b>85</b>
<b>Partie 2.4 : Programmation modulaire .....</b>	<b>98</b>
<b>Partie 2.5 : Fonction .....</b>	<b>103</b>
<b>Partie 2.6 : Fichier et Mini-Projet.....</b>	<b>113</b>

# Chapitres

Parties .....	2
Chapitres.....	3
TD .....	5
TP.....	7
Avant-propos.....	10
Introduction a.....	11
Références .....	12
<b>Section 1 : Algorithmique.....</b>	<b>13</b>
<b>Partie 1.1 : Introduction à la programmation .....</b>	<b>17</b>
Chapitre 1.1.1 : Informatique .....	18
Chapitre 1.1.2 : Système Informatique.....	21
Chapitre 1.1.3 : Langages informatiques .....	23
<b>Partie 1.2 : Introduction à l'algorithmique.....</b>	<b>25</b>
Chapitre 1.2.1 : Qu'est-ce que l'algorithmique ?.....	26
Chapitre 1.2.2 : Quatre catégories d'ordre.....	30
Chapitre 1.2.3 : Conception d'un algorithme .....	31
<b>Partie 1.3 : Algorithmique .....</b>	<b>33</b>
Chapitre 1.3.1 : Notion de variable.....	35
Chapitre 1.3.2 : Instruction d'affectation .....	39
<b>Section 2 : Langage C .....</b>	<b>42</b>
<b>Partie 2.1 : Introduction au langage C .....</b>	<b>46</b>
Chapitre 2.1.1 : Bonjour Langage C .....	47
Chapitre 2.1.2 : Types de base .....	49
Chapitre 2.1.3 : entrées-sorties.....	52
Solution des TP.....	56
<b>Partie 2.2 : Instruction de contrôle.....</b>	<b>58</b>
Chapitre 2.2.1 : Condition .....	59

Chapitre 2.2.2 : Boucle.....	64
Chapitre : 2.2.3 Opérateurs et expressions.....	74
<b>Partie 2.3 : Tableau et adresse.....</b>	<b>85</b>
Chapitre 2.3.1 : Tableaux.....	86
Chapitre 2.3.2 : Chaînes de caractères.....	90
<b>Partie 2.4 : Programmation modulaire.....</b>	<b>98</b>
Chapitre 2.4.1 : Structures.....	99
<b>Partie 2.5 : Fonction.....</b>	<b>103</b>
Chapitre 2.5.1 Fonction.....	104
Chapitre 2.5.2 : Variable & Adresse.....	109
<b>Partie 2.6 : Fichier et Mini-Projet.....</b>	<b>113</b>
Chapitre 6.1 : Fichier.....	114
Mini-Projet : Gestion des personnes.....	124

# TD

<b>Section 1 : Algorithmique</b>	13
<b>Partie 1.1 : Introduction à la programmation</b>	17
TD 0 - Proposition des phrases pour les 3 instructions	22
<b>Partie 1.2 : Introduction à l'algorithmique</b>	25
TD 1 : Calcule de temps d'exécution d'un traitement manuel	28
TD 2 : Lire et afficher un nombre	30
TD 3 : échange de deux valeurs - Technique 1	31
TD 4 : Besoin des techniques en programmation structurée	31
TD 5 : Besoin en technique de la programmation structurée	32
<b>Partie 1.3 : Algorithmique</b>	33
TD 6 : Algorithme de Karim – Notion du variable	35
Exercice 1 : Affectation	40
TD 7 : Technique1 : échange entre deux variables	40
TD 8 : Concaténation	40
<b>Section 2 : Langage C</b>	42
<b>Partie 2.1 : Introduction au langage C</b>	46
<b>Partie 2.2 : Instruction de contrôle</b>	58
<b>Partie 2.3 : Tableau et adresse</b>	85
<b>Partie 2.4 : Programmation modulaire</b>	98
TP 63 : Déclaration d'une structure	100
<b>Partie 2.5 : Fonction</b>	103
TP 71 : Fonction du menu principal	105
TP 73 : Exemple d'utilisation d'une variable globale	107
TP 74 :	109
TP 75	109
TP 77 Fonction Lire et Ecrire un tableau	110
<b>Partie 2.6 : Fichier et Mini-Projet</b>	113



# TP

<b>Section 1 : Algorithmique</b>	13
<b>Partie 1.1 : Introduction à la programmation</b>	17
<b>Partie 1.2 : Introduction à l'algorithmique</b>	25
<b>Partie 1.3 : Algorithmique</b>	33
<b>Section 2 : Langage C</b>	42
<b>Partie 2.1 : Introduction au langage C</b>	46
TP 1 : Afficher	47
TP 2 : Lire et écrire un nombre	48
TP 2 : Lire et Ecrire un nombre entier	49
TP 3 : Lire et écrire un nombre flottant	50
TP 4 : Lire et écrire un caractère	50
TP 5 : Possibilités de la fonction « printf »	54
TP 6 : Possibilités de la fonction « scanf »	54
<b>Partie 2.2 : Instruction de contrôle</b>	58
TP 7 : Instruction if	59
TP 8 : étude de cas - Prix TTC	59
TP 9 : Afficher la valeur maximale de deux variables	60
TP 10 : étude de cas – Calcule avec If	60
TP 11 : équation de deuxième degré	60
TP 12: Instruction switch	61
TP 13 : Instruction switch avec default	61
TP 14 : Instruction Switch – rôle de « break »	62
TP 15 : Menu d'application – Calculatrice	62
TP 16 : Boucle For	64
TP 17 : Utilisation de Break dans une boucle	65
TP 18 : Utilisation de continue	65
TP 19 : Affichage de la table de multiplication	66

TP 20 : Utilisation des boucles Niveau 2 .....	67
TP 21 : Utilisation des boucles Niveau 3 .....	67
TP 22 - Cours: Boucle While .....	68
TP 23 : La somme des nombres.....	68
TP 24 : Traduction de for vers while et do..while .....	68
TP 25: Boucle qui s'arrête à une valeur avec calcule .....	70
TP 26 : Déterminer si un nombre est premier .....	70
TP 27 : Boucle - do... while.....	71
TP 28: Menu d'application .....	71
TP 29 : Expression Mixte .....	74
TP 30 : Conversion implicite .....	75
TP 31 : Valeur Booléen.....	76
TP 32 : opérateurs logiques.....	78
TP 33 : Pré-Incrémentation et Post-Incrémentation .....	79
TP 34 : Opérateur d'affectation élargie.....	80
TP 35 : Conversion forcée par une affectation .....	80
TP 36 : Utilisation de Cast .....	81
TP 37 : Opérateur conditionnel .....	82
TP 38 : Opérateur - sizeof .....	82
TP 39 : sur l'opérateur conditionnel.....	83
TP 40 : sur l'opérateur conditionnel.....	83
TP 41 : Exercice sur l'opérateur d'incrémentacion.....	83
<b>Partie 2.3 : Tableau et adresse.....</b>	<b>85</b>
TP 42 : Saisir et Afficher un tableau .....	87
TP 43 : Somme et Moyenne d'un tableau .....	87
TP 44 : La valeur maximale d'un tableau .....	87
TP 45 : échanger la valeur maximale avec la valeur de la fin du tableau.....	87
TP 46 : Boucle avec condition .....	87
TP 47 : Trie par sélection .....	88
TP 48 : Rechercher dans tableau .....	88
TP 49 .....	88



TP 50 .....	88
TP 51 : Lire et écrire un tableau de plusieurs dimensions .....	89
TP 52 : .....	89
TP 53 : Lire et afficher chaîne de caractère .....	91
TP 54 : Exercice.....	92
TP 55 : strcpy .....	92
TP 56 : strcat .....	93
TP 57 : strcmp.....	93
TP 58 : strcpy .....	94
TP 59 : stelen .....	95
TP 60 : Exercice 1.....	95
TP 61 : Exercice 2.....	95
TP 62 : Exercice 3.....	95
<b>Partie 2.4 : Programmation modulaire .....</b>	<b>98</b>
<b>Partie 2.5 : Fonction .....</b>	<b>103</b>
TP 67 : Bonjour fonction .....	104
TP 68 : Fonction Somme .....	104
TP 69 : Le Menu d'application et fonctions.....	105
TP 70 : l'instruction return.....	105
TP 72 : Transmission par valeur.....	106
TP 76.....	110
<b>Partie 2.6 : Fichier et Mini-Projet.....</b>	<b>113</b>
TP : 82 : fopen, fprintf et fscanf .....	115
TP : 83 : fread et fwrite .....	116
TP : 84 : Fichier et Tableau .....	117

# Avant-propos

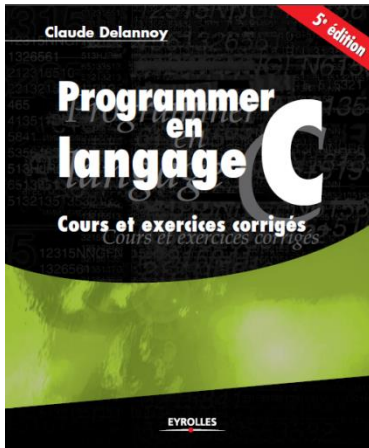
Le langage C a été créé en **1972** par Denis Ritchie avec un objectif relativement limité : écrire un système d'exploitation (UNIX).

Mais ses qualités opérationnelles l'ont très vite fait adopter par une large communauté de programmeurs.

# Introduction a

L'objectif de ce module est d'initier les stagiaires à élaborer des algorithmes pour répondre à des problèmes variés et réaliser des programmes pour résoudre des problèmes informatiques de façon logique et structurée.

# Références



# Section 1 : Algorithmique

<b>Partie 1.1 : Introduction à la programmation</b>	17
TD 0 - Proposition des phrases pour les 3 instructions	22
<b>Partie 1.2 : Introduction à l'algorithmique</b>	25
TD 1 : Calcule de temps d'exécution d'un traitement manuel	28
TD 2 : Lire et afficher un nombre	30
TD 3 : échange de deux valeurs - Technique 1	31
TD 4 : Besoin des techniques en programmation structurée	31
TD 5 : Besoin en technique de la programmation structurée	32
<b>Partie 1.3 : Algorithmique</b>	33
TD 6 : Algorithme de Karim – Notion du variable	35
Exercice 1 : Affectation	40
TD 7 : Technique1 : échange entre deux variables	40
TD 8 : Concaténation	40
<b>Partie 2.1 : Introduction au langage C</b>	46
TP 1 : Afficher	47
TP 2 : Lire et écrire un nombre	48
TP 2 : Lire et Ecrire un nombre entier	49
TP 3 : Lire et écrire un nombre flottant	50
TP 4 : Lire et écrire un caractère	50
TP 5 : Possibilités de la fonction « printf »	54
TP 6 : Possibilités de la fonction « scanf »	54
<b>Partie 2.2 : Instruction de contrôle</b>	58
TP 7 : Instruction if	59
TP 8 : étude de cas - Prix TTC	59
TP 9 : Afficher la valeur maximale de deux variables	60
TP 10 : étude de cas – Calcule avec If	60

TP 11 : équation de deuxième degré .....	60
TP 12: Instruction switch .....	61
TP 13 : Instruction switch avec default .....	61
TP 14 : Instruction Switch – rôle de « break » .....	62
TP 15 : Menu d'application – Calculatrice .....	62
TP 16 : Boucle For.....	64
TP 17 : Utilisation de Break dans une boucle.....	65
TP 18 : Utilisation de continue .....	65
TP 19 : Affichage de la table de multiplication.....	66
TP 20 : Utilisation des boucles Niveau 2 .....	67
TP 21 : Utilisation des boucles Niveau 3 .....	67
TP 22 - Cours: Boucle While.....	68
TP 23 : La somme des nombres .....	68
TP 24 : Traduction de for vers while et do..while.....	68
TP 25: Boucle qui s'arrête à une valeur avec calcule.....	70
TP 26 : Déterminer si un nombre est premier .....	70
TP 27 : Boucle - do... while .....	71
TP 28: Menu d'application.....	71
TP 29 : Expression Mixte .....	74
TP 30 : Conversion implicite.....	75
TP 31 : Valeur Booléen .....	76
TP 32 : opérateurs logiques .....	78
TP 33 : Pré-Incrémentation et Post-Incrémentation.....	79
TP 34 : Opérateur d'affectation élargie .....	80
TP 35 : Conversion forcée par une affectation.....	80
TP 36 : Utilisation de Cast.....	81
TP 37 : Opérateur conditionnel .....	82
TP 38 : Opérateur - sizeof .....	82
TP 39 : sur l'opérateur conditionnel .....	83
TP 40 : sur l'opérateur conditionnel .....	83
TP 41 : Exercice sur l'opérateur d'incrémentation .....	83

<b>Partie 2.3 : Tableau et adresse .....</b>	<b>85</b>
TP 42 : Saisir et Afficher un tableau .....	87
TP 43 : Somme et Moyenne d'un tableau.....	87
TP 44 : La valeur maximale d'un tableau .....	87
TP 45 : échanger la valeur maximale avec la valeur de la fin du tableau .....	87
TP 46 : Boucle avec condition .....	87
TP 47 : Trie par sélection.....	88
TP 48 : Rechercher dans tableau.....	88
TP 49 .....	88
TP 50 .....	88
TP 51 : Lire et écrire un tableau de plusieurs dimensions.....	89
TP 52 : .....	89
TP 53 : Lire et afficher chaine de caractère.....	91
TP 54 : Exercice.....	92
TP 55 : strcpy .....	92
TP 56 : strcat.....	93
TP 57 : strcmp .....	93
TP 58 : strcpy .....	94
TP 59 : stelen .....	95
TP 60 : Exercice 1 .....	95
TP 61 : Exercice 2 .....	95
TP 62 : Exercice 3 .....	95
<b>Partie 2.4 : Programmation modulaire.....</b>	<b>98</b>
TP 63 : Déclaration d'une structure .....	100
<b>Partie 2.5 : Fonction .....</b>	<b>103</b>
TP 67 : Bonjour fonction.....	104
TP 68 : Fonction Somme .....	104
TP 69 : Le Menu d'application et fonctions.....	105
TP 70 : l'instruction return .....	105
TP 71 : Fonction du menu principal .....	105
TP 72 : Transmission par valeur .....	106

TP 73 : Exemple d'utilisation d'une variable globale .....	107
TP 74 : .....	109
TP 75 .....	109
TP 76 .....	110
TP 77 Fonction Lire et Ecrire un tableau .....	110
<b>Partie 2.6 : Fichier et Mini-Projet</b> .....	113
TP : 82 : fopen, fprintf et fscanf .....	115
TP : 83 : fread et fwrite .....	116
TP : 84 : Fichier et Tableau .....	117



# **Partie 1.1 : Introduction à la programmation**

TD 0 - Proposition des phrases pour les 3 instructions.....	22
---	----

# Chapitre 1.1.1 : Informatique

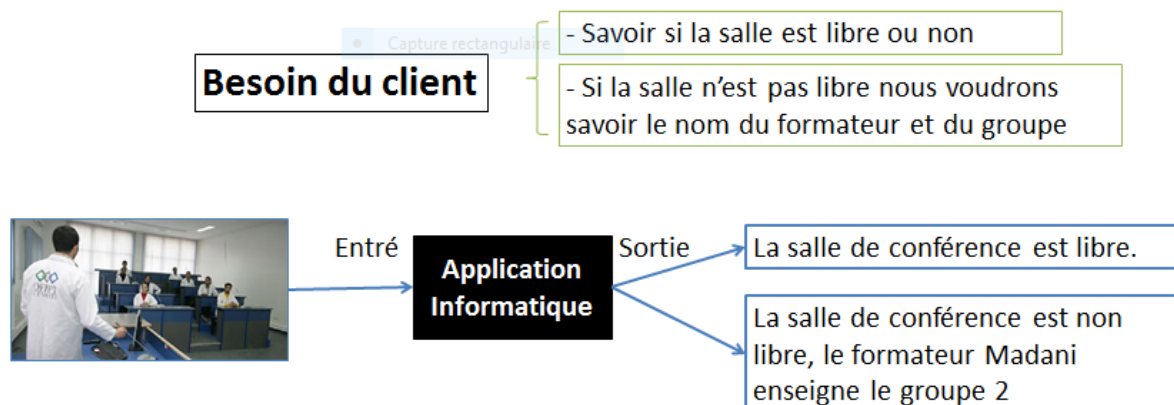
**Informatique = Traitement automatique de l'information**

## Information



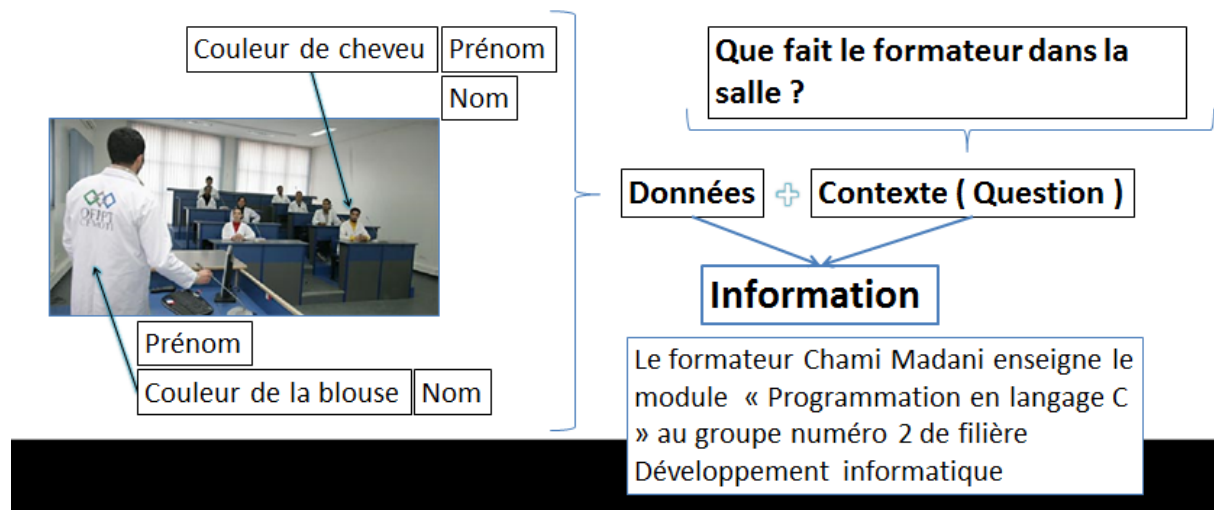
- Le formateur Chami Madani enseigne le module « Programmation en langage C » au groupe numéro 2 de filière « Développement informatique »
- Le groupe 2 contient 9 stagiaires
- Le formateur enseigne le groupe 2 dans la salle de conférence

## Traitement automatique de l'information



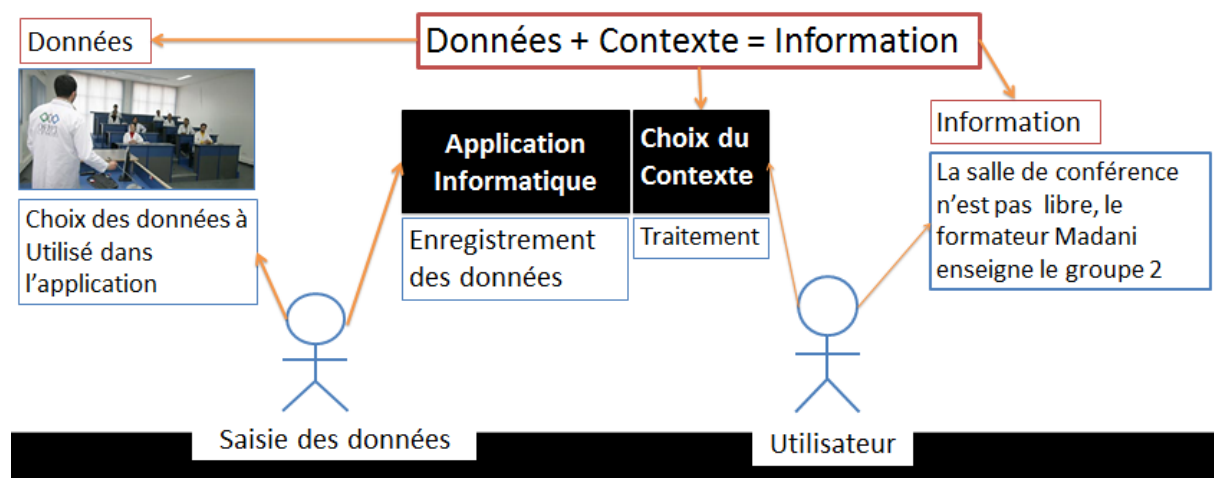
# Données

Information = Données + Contexte (Question, Utilisation des données)

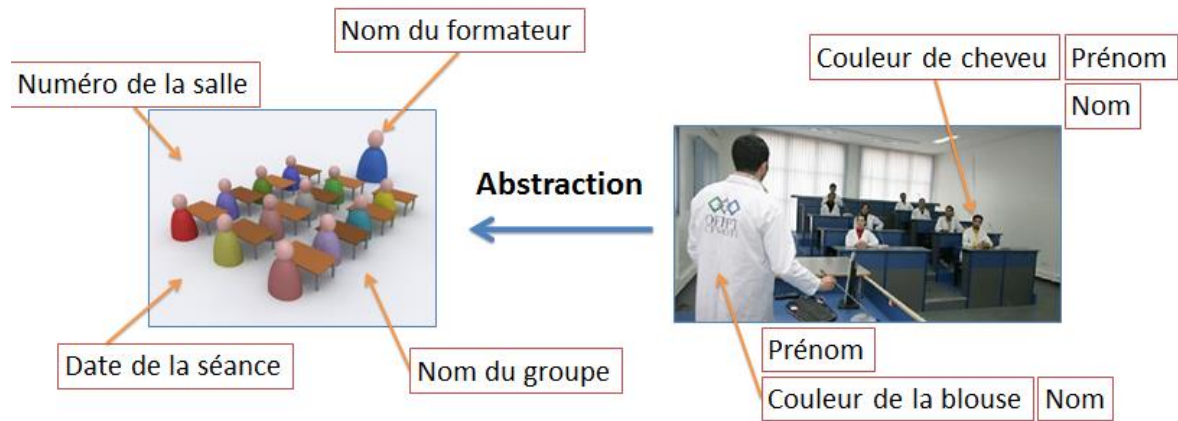


# Traitement

## Comment traiter automatiquement l'information ?



## Comment choisir les données à utiliser dans l'application ?



## Chapitre 1.1.2 : Système Informatique

Techniques du traitement **automatique** de l'**information** au moyen des ordinateurs

Système informatique = ordinateur + Application

### Matériel: Principaux éléments d'un ordinateur

- Unité centrale (le boîtier)
- Processeur ou CPU (*Central Processing Unit*)
  - Mémoire centrale
  - Unité de calcul
  - Unité de commande
- Périphériques
  - Moniteur (l'écran), clavier, souris
  - Modem, imprimante, scanner, ...

### Programme d'ordinateur?

Pour faire marcher un ordinateur il faut lui fournir un programme

Ordinateur = matériel + programme(s)

Un programme est une suite **d'instructions d'ordinateur**

Une instruction est un **ordre** compris par l'ordinateur et qui lui fait exécuté une **action**, c-à-d une modification de son **environnement**

## Les catégories d'ordres

Les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que **cinq** catégories d'ordres (en programmation, on n'emploiera pas le terme d'ordre, mais plutôt celui **d'instructions**). Ces quatre familles d'instructions sont :

- Déclaration d'une variable
- l'affectation de variables
- la lecture / écriture
- les tests
- les boucles

## TD 0 - Proposition des phrases pour les 3 instructions

Proposer des phrases en français pour les ordres suivant :

- 1 - Déclaration d'une variable : i
- 2 – Affectation de la valeur 5 à i
- 3 – Lecture 5 **et** affecter à i
- 4 – Ecriture 5

## Chapitre 1.1.3 : Langages informatiques

Un langage informatique est un outil permettant de donner des **ordres** (**instructions**) à la machine

- A chaque instruction correspond une action du processeur

Intérêt : écrire des programmes (suite consécutive d'instructions) destinés à effectuer une tâche donnée

- Exemple: un programme de gestion des stagiaires

Contrainte: être compréhensible par la machine

### Langage machine

- **Langage binaire**: l'information est exprimée et manipulée sous forme d'une suite de bits
- Un **bit** (*binary digit*) = 0 ou 1 (2 états électriques)
- Une combinaison de 8 bits = 1 **Octet** →  $2^8 = 256$  possibilités qui permettent de coder tous les caractères alphabétiques, numériques, et symboles tels que ?, \*, &, ...
  - Le code **ASCII** (*American Standard Code for Information Interchange*) donne les correspondances entre les caractères alphanumériques et leurs représentation binaire, Ex. A = 01000001, ? = 00111111
- Les opérations logiques et arithmétiques de base (addition, multiplication, ... ) sont effectuées en binaire





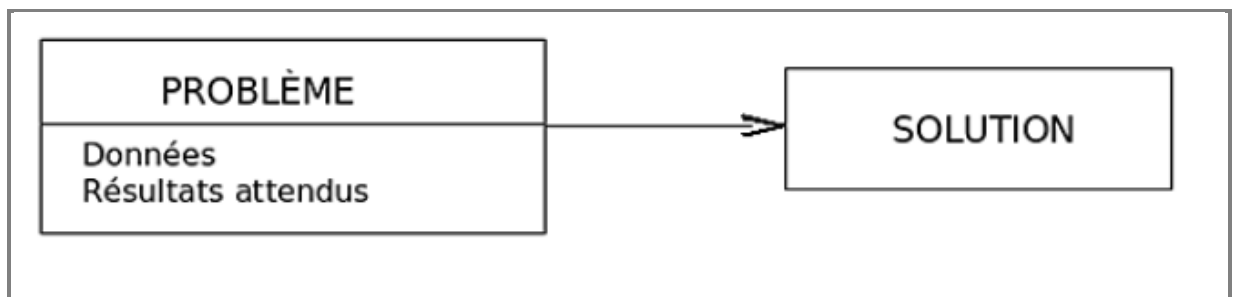
# Partie 1.2 : Introduction à l'algorithmique

TD 1 : Calcule de temps d'exécution d'un traitement manuel.....	28
TD 2 : Lire et afficher nombre .....	30
TD 2 : échange de deux valeurs - Technique 1 .....	31
TD 3 : Besoin des techniques en programmation structurée .....	31
TD 4 : Besoin en technique de la programmation structurée .....	32

## Chapitre 1.2.1 : Qu'est-ce que l'algorithmique ?

On s'intéresse à l'activité de programmation d'un ordinateur qui permet de résoudre des problèmes d'une façon automatique. On se place à un niveau conceptuel dans lequel un problème quelconque, informatique ou non, est caractérisé par un ensemble de données d'entrée et des résultats attendus.


On peut donc représenter ce niveau d'analyse par le schéma de la figure ci-dessous.



**Figure 1 : Rôle d'un algorithme**

Exemple 1 : Préparation d'une omelette

Référence : [http://www.marmiton.org/recettes/recette\\_omelette-nature\\_21255.aspx](http://www.marmiton.org/recettes/recette_omelette-nature_21255.aspx)



Temps  
**15 min**

Personnes  
**4**


Facile

Bon marché

Ajouter

## Ingrédients

Nombre de personnes - 4 +

 7 oeufs

 50 g de beurre

 0 Sel

 0 Poivre

## Préparation

imprimer 

TEMPS TOTAL : 15 MIN

Préparation : 5 min

Cuisson : 10 min

### Etape 1

Battez les oeufs à la fourchette, salez et poivrez.

### Etape 2

Faites chauffer le beurre, versez-en un peu dans les oeufs et mélangez. Versez les oeufs dans la poêle à feu vif, baissez le feu et laissez cuire doucement en ramenant les bords de l'omelette au centre au fur et à mesure qu'ils prennent.

### Etape 3

Secouez un peu la poêle pour éviter que l'omelette n'attache, vérifiez la texture baveuse ou bien prise.

### Etape 4

Pliez l'omelette en deux et servez.

## TD 1 : Calcule de temps d'exécution d'un traitement manuel

Calculez le temps d'exécution de l'algorithme suivant :

Echange les valeurs de deux listes de 7 million d'éléments, avec la condition suivante :

On échange si l'une des valeurs est inférieure à 5

Exemple

1	7
8	6
5	3

7	1
8	6
3	5

Exemple 2 : Additionner deux nombres

Tout élève apprend à additionner deux nombres entiers positifs en classe primaire. Pour calculer la somme de 127 et 35, il apprend à « poser » l'addition, en plaçant en colonnes les chiffres des deux nombres, les unités du second sous les unités du premier, les dizaines du second sous les dizaines du premier, etc. Ceci fait, il utilise un répertoire de connaissances acquises préalablement, pour additionner les deux nombres chiffre à chiffre. Il doit donc savoir que 7+5 font 12, 2+3 font 5, etc. Il apprend aussi à reporter la retenue lorsque l'addition de deux chiffres donne une somme supérieure à 9. Il est aussi possible de détailler les étapes du calcul de la façon suivante :

1. poser 127 ;
2. poser 35 sous 127, avec 5 sous 7, 3 sous 2 ;
3. additionner 7 et 5 qui font 12 : poser 2 pour les unités du résultat et 1 de retenue pour les dizaines ;

4. additionner 2 et 3 qui font 5 pour les dizaines et augmenter le résultat de la retenue précédente : poser 6 ;

5. additionner les centaines 1 à 0 : poser 1 ;

6. le résultat est 162.

Cette description est comparable à la recette précédente. On considère les ingrédients que sont les nombres à additionner, 127 et 35. À partir des tables d'addition, dont la connaissance est un préalable, on applique un ensemble de règles qui transforme les données, pour obtenir un résultat, ici leur somme 162.

## Chapitre 1.2.2 : Quatre catégories d'ordre

Les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que **quatre catégories d'ordres** (en programmation, on n'emploiera pas le terme d'ordre, mais plutôt celui d'**instructions**). Ces quatre familles d'instructions sont :

- Les variables et leur affectation
- la lecture / écriture
- les tests
- les boucles

### TD 2 : Lire et afficher un nombre

Ecrire un algorithme qui lire et affiche un nombre

# Chapitre 1.2.3 : Conception d'un algorithme

## TD 3 : échange de deux valeurs - Technique 1

Ecrire un programme qui permet de

- Lire a et b
- Echanger la valeur de a par b, et la valeur de b par la valeur de a
- Afficher la valeur de a et b

## TD 4 : Besoin des techniques en programmation structurée

### La pomme la plus grande

Vous avez 100 pommes dans un panier, on vous demande d'expliquer une méthode pratique pour trouver la plus grand pomme



## TD 5 : Besoin en technique de la programmation structurée

### Le nombre existe dans deux listes

Trouver le nombre existant dans les deux listes suivant :

Liste 1	Liste 2
100	106
102	102
500	501
653	560
...	...
...	...



## Partie 1.3 : Algorithmique

TD 6 : Algorithme de Karim – Notion du variable .....	<b>Erreur ! Signet non défini.</b>
Exercice 1 : Affectation .....	<b>Erreur ! Signet non défini.</b>
TD 7 : Technique1 : échange entre deux variables .....	<b>Erreur ! Signet non défini.</b>
TD 8 : Concaténation .....	<b>Erreur ! Signet non défini.</b>
TD 9 : Module et Puissance .....	<b>Erreur ! Signet non défini.</b>
TP Algo-1 : Lire et écrire un nombre .....	<b>Erreur ! Signet non défini.</b>
TP Algo-Exercice 1 : Affectation .....	<b>Erreur ! Signet non défini.</b>
TP Algo-TD7 : échange deux variables .....	<b>Erreur ! Signet non défini.</b>
TD 10 : Lire et Ecrire .....	<b>Erreur ! Signet non défini.</b>
TP Algo-TD 10 : Lire et Ecrire .....	<b>Erreur ! Signet non défini.</b>
TD 11 : Concaténation de deux chaînes de caractère ....	<b>Erreur ! Signet non défini.</b>
TP Algo-5 : Concaténation de deux chaînes de caractère .....	<b>Erreur ! Signet non défini.</b>
TD : Erreurs .....	<b>Erreur ! Signet non défini.</b>
TD 12 : Construction d'un algorithme simple .....	<b>Erreur ! Signet non défini.</b>
TP Algo-6 - Construction d'un algorithme simple .....	<b>Erreur ! Signet non défini.</b>
TD 13 : Si .. Alors .. sinon .....	<b>Erreur ! Signet non défini.</b>
TD 14: .....	<b>Erreur ! Signet non défini.</b>
TD : Tests imbriqués .....	<b>Erreur ! Signet non défini.</b>
TD -Exercice : Tests imbriqués .....	<b>Erreur ! Signet non défini.</b>
TD : TD préparatoire à la boucle Pour .....	<b>Erreur ! Signet non défini.</b>
TD : .....	<b>Erreur ! Signet non défini.</b>
TD : La somme de N nombre avec Pour .....	<b>Erreur ! Signet non défini.</b>
TD : .....	<b>Erreur ! Signet non défini.</b>
TD .....	<b>Erreur ! Signet non défini.</b>
TD : Boucle Tant que : exemple simple .....	<b>Erreur ! Signet non défini.</b>

TD : Menu d'application ..... **Erreur ! Signet non défini.**

TD : Algorithme d'Euclide..... **Erreur ! Signet non défini.**

## Chapitre 1.3.1 : Notion de variable

### TD 6 : Algorithme de Karim – Notion du variable

Karim possède 3 seaux : un seau en plastique d'une contenance de 10 litres, un seau en bois d'une contenance de 7 litres et un seau en fer d'une contenance de 9 litres.

- 10h00 : karim vide ses 3 seaux
- 10h05 : karim va rendre visite a Nabil, celui-ci met 6 litres dans le seau en bois de Karim
- 10h10 : karim transverse le contenu de son seau en bois dans le seau en fer
- 10h15 : karim revient vers nabil remplir à ras bord son seau en plastique
- 10h20 : karim déverse la moitié de son seau en plastique à l'égout
- 10h25 : karim transvase le contenu de son seau en plastique dans celui en bois
- 10h30 : karim transvase 2 litres de son seau en bois dans celui en fer
- 10h35 : karim informe Asmae du nombre de litres contenu dans ses seaux en plastique, en bois, en fer.

Question 1 : Donner un schéma qui représente le premier paragraphe

Question 2 : Quelles sont les quantités des trois seaux que « Asmae » a reçues?

Question 3 : Remplir le tableau suivant

seau en plastique 10 L	seau en bois 7 L	seau en fer 9 L

## Notion fondamentale

- **Notion d'algorithme** : si les huit phrases sont bien exécutées par Karim, alors l'histoire est un algorithme
- **Notion d'instruction** : chacune des huit phrases est une instruction (un ordre)
- **Notion de valeur** : { 0, 3, 5, 6, 8, 10 }
- **Notion de mémoire** : elle est matérialisée par les seaux qui « mémorisent » les quantités de liquide
- **Notion de variable** : une variable est un emplacement mémoire, ici on a trois variables (le seau en plastique, le seau en bois et le seau en fer)
- **Notion d'environnement** : c'est l'ensemble des objets, informations, personnes qui ont une existence hors de l'histoire mais qui interviennent dans son déroulement.
- **Notion des valeurs d'entrée et de sortie** : ce sont les valeurs que le processeur reçoit de l'environnement et celles qu'il donne à l'environnement durant l'exécution. Valeurs en entrée : {6, 10}      Valeurs en sortie = {0, 3, 8}

## Notion de variable

Dans les langages de programmation une **variable** sert à stocker la valeur d'une donnée

Une variable désigne en fait **un emplacement mémoire** dont le contenu peut changer au cours d'un programme (d'où le nom variable)

Règle : Les variables doivent être **déclarées** avant d'être utilisées, elles doivent être caractérisées par :

- un nom (**Identificateur**)
- un **type** (entier, réel, caractère, chaîne de caractères, ...)

## Choix des identificateurs

Le choix des noms de variables est soumis à quelques règles qui **varient selon le langage**, mais en général :

Un nom doit commencer par une lettre alphabétique

- exemple valide : B1
- exemple invalide : 1B

Doit être constitué uniquement de lettres, de chiffres et du soulignement \_ (Éviter les caractères de ponctuation et les espaces)

- valides : TangerMaroc, Tanger\_Maroc
- invalides : Tanger Maroc, Tanger-Maroc, Tanger;Maroc

Doit être différent des mots réservés du langage (par exemple en Langage C: **int**, **float**, **else**, **case**, **for**, **main**, **return**, ...)

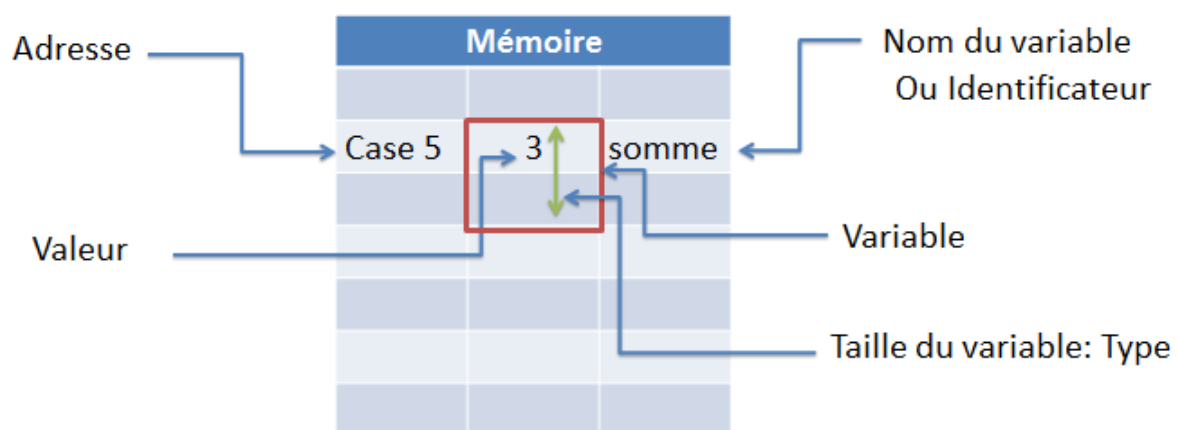
La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

Conseil : pour la lisibilité du code choisir des noms significatifs qui décrivent les données manipulées

**Exemples : TotalVentes, Prix\_TTC, Prix\_HT, CIN**

Remarque : en pseudo-code algorithmique, on va respecter les règles citées, même si on est libre dans la syntaxe.

## Adresse et Nom du variable



## Types des variables

Le type d'une variable détermine **l'ensemble des valeurs** qu'elle peut prendre.

Les types offerts par la plus part des langages sont :

- Type numérique (entier ou réel)
  - **Byte** (codé sur 1octet): de 0 à 255
  - **Entier court** (codé sur 2 octets) : -32 768 à 32 767
  - **Entier long** (codé sur 4 ou 8 octets)
  - **Réel simple précision** (codé sur 4 octets)
  - **Réel double précision** (codé sur 8 octets)
- Type logique ou booléen:

- **Booléen** : deux valeurs VRAI ou FAUX
- Type caractère :
  - **Caractère** : lettres majuscules, minuscules, chiffres, symboles, ...
  - exemples : 'A', 'a', '1', '?', ...
- Type chaîne de caractère :
  - **Chaîne de caractères** : toute suite de caractères, **exemples** : "Madani Ali", "900", "a11" ...

## Déclaration des variables

Rappel : toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration préalable

En pseudo-code, on va adopter la forme suivante pour la déclaration de variables

Syntaxe :

Variables      liste d'identificateurs : type
---

Exemple:

Variables i,j,k : entier x, y : réel OK: booléen ch1, ch2 : chaîne de caractères
---

## Chapitre 1.3.2 : Instruction d'affectation

**L'affectation** consiste à attribuer une valeur à une variable (ça consiste en fait à remplir ou à modifier le contenu d'une zone mémoire)

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

Var1← e : attribue la valeur de e à la variable Var

- e peut être une valeur, une autre variable ou une expression
- Var et e doivent être de même type ou de types compatibles
- l'affectation ne modifie que ce qui est à gauche de la flèche

**Ex valides :**

```
i←1
j←i
k←i+j
x←10.3
OK←FAUX
ch1←"Tanger"
ch2←ch1
x←4
x←j
```

**Non valides :**

```
i←10.3
OK←"Tanger"
j ←x
```

## Quelques remarques

Beaucoup de langages de programmation (C/C++, Java, ...) utilisent le signe égal = pour l'affectation ←. Attention aux confusions:

- l'affectation n'est pas commutative : A=B est différente de B=A
- l'affectation est différente d'une équation mathématique :
  - A=A+1 a un sens en langages de programmation
  - A+1=2 n'est pas possible en langages de programmation et n'est pas équivalente à A=1

- Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable **d'initialiser les variables** déclarées

## Exercice 1 : Affectation

Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

**Variables A, B, C: Entier**

**Début**

$A \leftarrow 3$

$B \leftarrow 7$

$A \leftarrow B$

$B \leftarrow A+5$

$C \leftarrow A + B$

$C \leftarrow B - A$

**Fin**

## TD 7 : Technique1 : échange entre deux variables

Ecrire un algorithme permettant de demander et d'échanger les valeurs de deux variables A et B

## TD 8 : Concaténation

Que produit l'algorithme suivant ?

Variables A, B, C en Caractères

Début

$A \leftarrow "423"$

$B \leftarrow "12"$

$C \leftarrow A + B$

Afficher ( C )

Fin





# Section 2 : Langage C

<b>Partie 2.1 : Introduction au langage C</b>	46
Chapitre 2.1.1 : Bonjour Langage C	47
TP 1 : Afficher	47
TP 2 : Lire et écrire un nombre	48
Chapitre 2.1.2 : Types de base	49
TP 2 : Lire et Ecrire un nombre entier	49
TP 3 : Lire et écrire un nombre flottant	50
TP 4 : Lire et écrire un caractère	50
Chapitre 2.1.3 : entrées-sorties	52
TP 5 : Possibilités de la fonction « printf »	54
TP 6 : Possibilités de la fonction « scanf »	54
Solution des TP	56
<b>Partie 2.2 : Instruction de contrôle</b>	58
Chapitre 2.2.1 : Condition	59
TP 7 : Instruction if	59
TP 8 : étude de cas - Prix TTC	59
TP 9 : Afficher la valeur maximale de deux variables	60
TP 10 : étude de cas – Calcule avec If	60
TP 11 : équation de deuxième degré	60
TP 12: Instruction switch	61
TP 13 : Instruction switch avec default	61
TP 14 : Instruction Switch – rôle de « break »	62
TP 15 : Menu d'application – Calculatrice	62
Chapitre 2.2.2 : Boucle	64
TP 16 : Boucle For	64
TP 17 : Utilisation de Break dans une boucle	65

TP 18 : Utilisation de continue .....	65
TP 19 : Affichage de la table de multiplication .....	66
TP 20 : Utilisation des boucles Niveau 2 .....	67
TP 21 : Utilisation des boucles Niveau 3 .....	67
TP 22 - Cours: Boucle While .....	68
TP 23 : La somme des nombres.....	68
TP 24 : Traduction de for vers while et do..while .....	68
TP 25: Boucle qui s'arrête à une valeur avec calcule .....	70
TP 26 : Déterminer si un nombre est premier .....	70
TP 27 : Boucle - do... while.....	71
TP 28: Menu d'application .....	71
Chapitre : 2.2.3 Opérateurs et expressions .....	74
TP 29 : Expression Mixte .....	74
TP 30 : Conversion implicite .....	75
TP 31 : Valeur Booléen.....	76
TP 32 : opérateurs logiques.....	78
TP 33 : Pré-Incrémentation et Post-Incrémentation .....	79
TP 34 : Opérateur d'affectation élargie.....	80
TP 35 : Conversion forcée par une affectation .....	80
TP 36 : Utilisation de Cast .....	81
TP 37 : Opérateur conditionnel .....	82
TP 38 : Opérateur - sizeof .....	82
TP 39 : sur l'opérateur conditionnel.....	83
TP 40 : sur l'opérateur conditionnel.....	83
TP 41 : Exercice sur l'opérateur d'incrémentation.....	83
<b>Partie 2.3 : Tableau et adresse .....</b>	<b>85</b>
Chapitre 2.3.1 : Tableaux .....	86
TP 42 : Saisir et Afficher un tableau .....	87
TP 43 : Somme et Moyenne d'un tableau .....	87
TP 44 : La valeur maximale d'un tableau .....	87
TP 45 : échanger la valeur maximale avec la valeur de la fin du tableau.....	87

TP 46 : Boucle avec condition .....	87
TP 47 : Trie par sélection .....	88
TP 48 : Rechercher dans tableau .....	88
TP 49 .....	88
TP 50 .....	88
TP 51 : Lire et écrire un tableau de plusieurs dimensions .....	89
TP 52 : .....	89
Chapitre 2.3.2 : Chaînes de caractères .....	90
TP 53 : Lire et afficher chaîne de caractère .....	91
TP 54 : Exercice .....	92
TP 55 : strcpy .....	92
TP 56 : strcat .....	93
TP 57 : strcmp .....	93
TP 58 : strcpy .....	94
TP 59 : stelen .....	95
TP 60 : Exercice 1 .....	95
TP 61 : Exercice 2 .....	95
TP 62 : Exercice 3 .....	95
<b>Partie 2.4 : Programmation modulaire</b> .....	98
Chapitre 2.4.1 : Structures .....	99
TP 63 : Déclaration d'une structure .....	100
<b>Partie 2.5 : Fonction</b> .....	103
Chapitre 2.5.1 Fonction .....	104
TP 67 : Bonjour fonction .....	104
TP 68 : Fonction Somme .....	104
TP 69 : Le Menu d'application et fonctions .....	105
TP 70 : l'instruction return .....	105
TP 71 : Fonction du menu principal .....	105
TP 72 : Transmission par valeur .....	106
TP 73 : Exemple d'utilisation d'une variable globale .....	107
Chapitre 2.5.2 : Variable & Adresse .....	109

TP 74 : .....	109
TP 75 .....	109
TP 76 .....	110
TP 77 Fonction Lire et Ecrire un tableau .....	110
<b>Partie 2.6 : Fichier et Mini-Projet</b> .....	113
Chapitre 6.1 : Fichier .....	114
TP : 82 : fopen, fprintf et fscanf .....	115
TP : 83 : fread et fwrite .....	116
TP : 84 : Fichier et Tableau .....	117
Mini-Projet : Gestion des personnes .....	124

## **Partie 2.1 : Introduction au langage C**

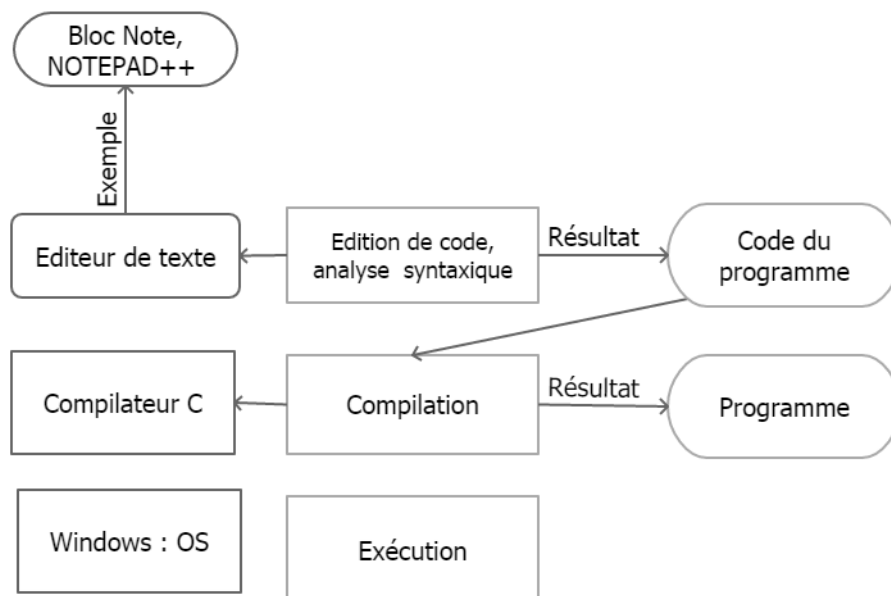
# Chapitre 2.1.1 : Bonjour Langage C

## Installation du compilateur

- DevC++

<https://www.youtube.com/watch?v=Rsuz5XkkoiA>

## Compilation



## TP 1 : Afficher

Ecrire un programme en langage C qui affiche le message « Bonjour langage C »

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{

    printf("Bonjour \n");
    system("PAUSE");
    return 0;
}
```

## **TP 2 : Lire et écrire un nombre**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int nombre; // Déclaration d'une variable
    printf("Donnez un nombre : ");
    scanf("%d",&nombre); // Lire (nombre)
    printf("Le nombre que vous avez saisi est : %d",nombre);
    system("PAUSE");
    return 0;
}
```



## Chapitre 2.1.2 : Types de base

### Notion de type

La **mémoire centrale** est un ensemble de positions binaires nommées **bits**. Les bits sont regroupés en **octets** (8 bits), et chaque octet est repéré par ce qu'on nomme son adresse.

Les types de base du langage C se répartissent en trois grandes catégories en fonction de la nature des informations qu'ils permettent de représenter :

- **nombres entiers** (mot-clé `int`),
- **nombres flottants** (mot-clé `float` ou `double`),
- **caractères** (mot-clé `char`) ; nous verrons qu'en fait `char` apparaît (en C) comme un cas particulier de `int`.  
— Chaîne de caractère

### Types entiers

Le langage C prévoit que, sur une machine donnée, on puisse trouver jusqu'à trois tailles différentes d'entiers, désignées par les mots-clés suivants :

- **short int** (qu'on peut abrégé en `short`),
- **int** (c'est celui que nous avons rencontré dans le chapitre précédent),
- **long int** (qu'on peut abrégé en `long`).

Chaque taille impose naturellement ses limites. Toutefois, ces dernières dépendent, non seulement du mot-clé considéré, mais également de la machine utilisée : tous les `int` n'ont pas la même taille sur toutes les machines ! Fréquemment, deux des trois mots-clés correspondent à une même taille (par exemple, sur PC, `short` et `int` correspondent à 16 bits, tandis que `long` correspond à 32 bits).

À titre indicatif, avec 16 bits, on représente des entiers s'étendant de  $-2^{15}$  à  $2^{15}-1$  ; avec 32 bits, on peut couvrir les valeurs allant de  $-2^{31}$  à  $2^{31}-1$ .

### TP 2 : Lire et Ecrire un nombre entier

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
```

```
{
    int nombre;
    printf("Donnez un nombre : ");
    scanf("%d",&nombre); //Lire(nombre)
    printf("Le nombre que vous avez saisi est : %d",nombre);
    system("PAUSE");
    return 0;
}
```

## Types flottants

Les types flottants permettent de représenter, **de manière approchée**, une partie des nombres réels.

```
float nombre ;
scanf("%f",&nombre);
printf("Le nombre que vous avez saisi est : %f",nombre);
```

### TP 3 : Lire et écrire un nombre flottant

Ecrire un programme qui lire et écrire un nombre flottants

## Types caractères

### TP 4 : Lire et écrire un caractère

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    char unCaractere ;

    printf("Ecrire un caractère : ");
    scanf("%c",&unCaractere);
    printf("Le caractère que vous avez saisi est : %c \n",unCaractere);

    system("PAUSE");
    return 0;
}
```

Caractères disposant d'une notation spéciale

NOTATION EN C	CODE ASCII (hexadécimal)	ABRÉVIATION USUELLE	SIGNIFICATION
\a	07	BEL	cloche ou bip (alert ou audible bell)
\b	08	BS	Retour arrière (Backspace)
\f	0C	FF	Saut de page (Form Feed)
\n	0A	LF	Saut de ligne (Line Feed)
\r	0D	CR	Retour chariot (Carriage Return)
\t	09	HT	Tabulation horizontale (Horizontal Tab)
\v	0B	VT	Tabulation verticale (Vertical Tab)
\\	5C	\	
\'	2C	'	
\"	22	"	
\?	3F	?	

## Chapitre 2.1.3 : entrées-sorties

Jusqu'ici, nous avons utilisé de façon intuitive les fonctions `printf` et `scanf` pour afficher des informations à l'écran ou pour en lire au clavier. Nous vous proposons maintenant étudier en détail les différentes possibilités de ces fonctions,

### Les possibilités de la fonction `printf`

#### Les principaux codes de conversion

- c**     `char` : caractère affiché « en clair » (convient aussi à `short` ou à `int` compte tenu des conversions systématiques)
- d**     `int` (convient aussi à `char` ou à `int`, compte tenu des conversions systématiques)
- u**     `unsigned int` (convient aussi à `unsigned char` ou à `unsigned short`, compte tenu des conversions systématiques)
- ld**    `long`
- lu**    `unsigned long`
- f**     `double` ou `float` (compte tenu des conversions systématiques `float` -> `double`) écrit en notation décimale avec six chiffres après le point (par exemple : 1.234500 ou 123.456789)
- e**     `double` ou `float` (compte tenu des conversions systématiques `float` -> `double`) écrit en notation exponentielle (mantisse entre 1 inclus et 10 exclu) avec six chiffres après le point décimal, sous la forme `x.xxxxxxe+yyy` ou `x.xxxxxx-yyy` pour les nombres positifs et `-x.xxxxxxe+yyy` ou `-x.xxxxxx-yyy` pour les nombres négatifs
- s**     chaîne de caractères dont on fournit l'adresse (notion qui sera étudiée ultérieurement)

### La macro `putchar`

L'expression :

`putchar (a)`

joue le même rôle que :

`printf ("%c", a)`

# Les possibilités de la fonction `scanf`

## Les principaux codes de conversion de `scanf`

<b>c</b>	char
<b>d</b>	int
<b>u</b>	unsigned int
<b>hd</b>	short int
<b>hu</b>	unsigned short
<b>ld</b>	long int
<b>lu</b>	unsigned long
<b>f</b> ou <b>e</b>	float écrit indifféremment dans l'une des deux notations : décimale (éventuellement sans point, c'est-à-dire comme un entier) ou exponentielle (avec la lettre e ou E)
<b>lf</b> ou <b>le</b>	double avec la même présentation que ci-dessus
<b>s</b>	chaîne de caractères dont on fournit l'adresse (notion qui sera étudiée ultérieurement)

Contrairement à ce qui se passait pour `printf`, il ne peut plus y avoir ici de conversion automatique puisque l'argument transmis à `scanf` est l'adresse d'un emplacement mémoire. C'est ce qui justifie l'existence d'un code `hd` par exemple pour le type `short` ou encore celle des codes `lf` et `le` pour le type `double`.

## La macro `getchar`

L'expression :

```
a = getchar()
```

joue le même rôle que :

```
scanf ("%c", &a)
```

## **TP 5 : Possibilités de la fonction « printf »**

Quels seront les résultats fournis par ce programme

```
#include <stdio.h>
main ()
{   int n = 543 ;
    int p = 5 ;
    float x = 34.5678;
    printf ("A : %d %f\n", n, x) ;
    printf ("B : %4d %10f\n", n, x) ;
    printf ("C : %2d %3f\n", n, x) ;
    printf ("D : %10.3f %10.3e\n", x, x) ;
    printf ("E : %*d\n", p, n) ;
    printf ("F : %*.*f\n", 12, 5, x) ;
}
```

## **TP 6 : Possibilités de la fonction « scanf »**

Quelles seront les valeurs lues dans les variables *n* et *p* (de type *int*), par l'instruction suivante ?

```
scanf ("%4d %2d", &n, &p) ;
```

Lorsqu'on lui fournit les données suivantes (le symbole ^ représente un espace et le symbole @ représente une fin de ligne, c'est-à-dire une validation) ?

- a) 12^45@
- b) 123456@
- c) 123456^7@
- d) 1^458@
- e) ^^4567^^8912@



# Solution des TP

## TP1 : Affichage de message bonjour

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    printf("Bonjour \n");
    system("PAUSE");
    return 0;
}
```

## TP2 : Lire et écrire un nombre entier

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int nombre;
    printf("Donnez un nombre : ");
    scanf("%d",&nombre);

    printf("Le nombre que vous avez saisi est : %d",nombre);

    system("PAUSE");
    return 0;
}
```



## TP3 : Lire et écrire un nombre flottante

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    float nombre;
    printf("Donnez un nombre : ");
    scanf("%f",&nombre);

    printf("Le nombre que vous avez saisi est : %f",nombre);

    system("PAUSE");
    return 0;
}
```

## **Partie 2.2 : Instruction de contrôle**

## Chapitre 2.2.1 : Condition

### Rappel :

#### Valeur booléen

#### Opérateur booléen

### L'instruction `if`

#### TP 7 : Instruction if

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int unNombre ;

    printf("Donnez un nombre : ");
    scanf("%d",&unNombre);

    if ( unNombre > 0 ){
        printf("Le nombre que vous avez saisi est positive \n");
    } else {
        printf("Le nombre que vous avez saisi est négative \n");
    }

    system("PAUSE");
    return 0;
}
```

#### TP 8 : étude de cas - Prix TTC

Ecrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement

## **TP 9 : Afficher la valeur maximale de deux variables**

Ecrire un programme qui demande deux valeurs et affiche la valeur maximale de

- Deux nombre
- Deux float
- Deux caractères

## **TP 10 : étude de cas – Calcule avec If**

Le prix de photocopies dans une reprographie varie selon le nombre demandé: 0,5 DH la copie pour un nombre de copies inférieur à 10, 0,4DH pour un nombre compris entre 10 et 20 et 0,3DH au-delà.

## **TP 11 : équation de deuxième degré**

### **Question 1**

```
int main(int argc, char *argv[]) {  
    int x = sqrt(9);  
    printf("%d", x);  
    return 0;  
}
```

### **Question 2 :**

Écrire l'algorithme du traitement qui calcule le discriminant DELTA d'un trinôme du second degré  $AX^2 + BX + C$  et qui, en fonction de son signe, calcule la ou les racines réelles du trinôme ou affiche, si besoin est qu'il n'y a pas de racine réelle.

Les trois coefficients A, B et C seront saisis au clavier avant traitement.

# Instruction switch

## TP 12: Instruction switch

```
#include<stdio.h>
#include<stdlib.h>

int main(){

    int n ;
    printf ("donnez un entier : ") ;
    scanf ("%d", &n) ;
    switch (n)
    {
        case 0 : printf ("zéro\n") ;
                break ;
        case 1 : printf ("un\n") ;
                break ;
        case 2 : printf ("deux\n") ;
                break ;
    }
    system("pause");
    return 0;
}
```

## TP 13 : Instruction switch avec default

```
#include<stdio.h>
#include<stdlib.h>

int main(){

    int n ;
    printf ("donnez un entier : ") ;
    scanf ("%d", &n) ;
    switch (n)
    {
        case 0 : printf ("zéro\n") ;
                break ;
        case 1 : printf ("un\n") ;
                break ;
        case 2 : printf ("deux\n") ;
                break ;
        default : printf ("grand\n") ;
    }
    system("pause");
    return 0;
}
```

## **TP 14 : Instruction Switch – rôle de « break »**

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

    int n ;
    printf ("donnez un entier : ") ;
    scanf ("%d", &n) ;
    switch (n)
    {
        case 0 : printf ("nul\n") ;
                break ;
        case 1 :
        case 2 : printf ("petit\n") ;
        case 3 :
        case 4 :
        case 5 : printf ("moyen\n") ;
                break ;
        default : printf ("grand\n") ;
    }
    system("pause");
    return 0;
}
```

## **TP 15 : Menu d'application – Calculatrice**

### **Question 1 :**

Ecrire un programme qui calcule la somme, la soustraction, la division et le Modulo entre deux variables données par l'utilisateur.

### **Question 2 :**

Ajouter à votre programme un menu d'utilisateur lui permet de choisir l'opération arithmétique à réaliser.

- a – en utilisant l'instruction « If »
- b – en utilisant l'instruction « Switch »

### **Exemple d'exécution**

```
Les opérations possibles
+ : Addition
+ : Soustraction
+ : Multiplication
+ : Division
+ : Module
Préciser l'opération à réaliser : +
Donnez la valeur 1 : 4
Donnez la valeur 2 : 2
```

La somme de 4 et 2 égale : 6

## Chapitre 2.2.2 : Boucle

### L'instruction `for`

Étudions maintenant la dernière instruction permettant de réaliser des boucles, à savoir l'instruction `for`.

#### TP 16 : Boucle For

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i ;
    for ( i=1 ; i<=5 ; i++ ) {
        printf ("bonjour " ) ;
        printf ("%d fois\n", i) ;
    }
    system("PAUSE");
    return 0;
}
```

### L'instruction `break`

Nous avons déjà vu le rôle de `break` au sein du bloc régi par une instruction `switch`. Le langage C autorise également l'emploi de cette instruction dans une boucle. Dans ce cas, elle sert à interrompre le déroulement de la boucle, en passant à l'instruction qui suit cette boucle.



## **TP 17 : Utilisation de Break dans une boucle**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i ;
    for ( i=1 ; i<=10 ; i++ ){
        printf ("début tour %d\n", i) ;
        printf ("bonjour\n");
        if ( i==3 ) break ;
        printf ("fin tour %d\n", i) ;
    }

    printf ("après la boucle") ;

    system("PAUSE");
    return 0;
}
```

## **L'instruction continue**

L'instruction continue, quant à elle, permet de passer prématurément au tour de boucle suivant.

## **TP 18 : Utilisation de continue**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i ;
    for ( i=1 ; i<=5 ; i++ ){
        printf ("début tour %d\n", i) ;
        if (i<4) continue ;
        printf ("bonjour\n") ;
    }

    system("PAUSE");
    return 0;
}
```

## **TP 19 : Affichage de la table de multiplication**

### **Question 1 :**

Écrire un programme qui affiche la table de multiplication de 9

```
1 * 9 = 1
2 * 9 = 18
3 * 9 = 27
4 * 9 = 36
5 * 9 = 45
6 * 9 = 54
7 * 9 = 63
8 * 9 = 72
9 * 9 = 81
```

### **Question 2 :**

Écrire un programme qui affiche la table de multiplication des nombres de 1 à 10, sous la forme suivante :

	I	1	2	3	4	5	6	7	8	9	10
1	I	1	2	3	4	5	6	7	8	9	10
2	I	2	4	6	8	10	12	14	16	18	20
3	I	3	6	9	12	15	18	21	24	27	30
4	I	4	8	12	16	20	24	28	32	36	40
5	I	5	10	15	20	25	30	35	40	45	50
6	I	6	12	18	24	30	36	42	48	54	60
7	I	7	14	21	28	35	42	49	56	63	70
8	I	8	16	24	32	40	48	56	64	72	80
9	I	9	18	27	36	45	54	63	72	81	90
10	I	10	20	30	40	50	60	70	80	90	100

## **TP 20 : Utilisation des boucles Niveau 2**

Afficher un triangle rempli d'étoiles, s'étendant sur un nombre de lignes fourni en donnée et se présentant comme dans cet exemple

```
*  
**  
***  
****  
*****
```

## **TP 21 : Utilisation des boucles Niveau 3**

Afficher un triangle rempli d'étoiles, s'étendant sur un nombre de lignes fourni en donnée et se présentant comme dans cet exemple

```
*  
***  
*****  
*****
```

## L'instruction `while`

Voyons maintenant la deuxième façon de réaliser une boucle conditionnelle, à savoir l'instruction « `while` ».

### TP 22 - Cours: Boucle While

Question 1 : que fait l'instruction `n+=2`

```
int n ;
n = 2 ;
n = n + 2 ;
printf(''%d'',n) ;
n += 2 ;
printf(''%d'',n) ;
```

Question 2 :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n, somme ;
    somme = 0 ;
    while (somme <100) {
        printf ("donnez un nombre : ") ;
        scanf ("%d", &n) ;
        somme += n ;
    }
    printf ("somme obtenue : %d", somme) ;

    system("PAUSE");
    return 0;
}
```

### TP 23 : La somme des nombres

Ecrire un programme qui affiche la somme des nombres saisies par l'utilisateur.

La somme ne doit pas dépasser la valeur 100.

### TP 24 : Traduction de `for` vers `while` et `do..while`

Soit le petit programme suivant

```
#include <stdio.h>
main()
{
    int i, n, som ;
    som = 0 ;
    for (i=0 ; i<4 ; i++)
        { printf ("donnez un entier ") ;
          scanf ("%d", &n) ;
          som += n ;
        }
    printf ("Somme : %d\n", som) ;
}
```

Écrire un programme réalisant exactement la même chose, en employant, à la place de l'instruction for :

- une instruction while,
- une instruction do... while.

## **TP 25: Boucle qui s'arrête à une valeur avec calcule**

Calculer la moyenne de notes fournies au clavier avec un dialogue de ce type :

note 1 : 12

note 2 : 15.25

note 3 : 13.5

note 4 : 8.75

note 5 : -1

moyenne de ces 4 notes : 12.37

Le nombre de notes n'est pas connu a priori et l'utilisateur peut en fournir autant qu'il le désire. Pour signaler qu'il a terminé, on convient qu'il fournira une note fictive négative. Celle-ci ne devra naturellement pas être prise en compte dans le calcul de la moyenne.

## **TP 26 : Déterminer si un nombre est premier**

Déterminer si un nombre est un nombre premier ou non.

NB. Un nombre premier est divisible sur 1 et lui même

Question 1

Donnez un nombre : 8

Le nombre 8 n'est pas un nombre premier

Question 2

Donnez un nombre premier : 8

Le nombre 8 n'est pas un nombre premier

Donnez un nombre premier : 7

Oui, 7 est un nombre premier

# L'instruction `do... while`

## TP 27 : Boucle - `do... while`

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n ;
    do {
        printf ("donnez un nb >0 : ") ;
        scanf ("%d", &n) ;
        printf ("vous avez fourni %d\n", n) ;
    } while (n<=0) ;
    printf ("réponse correcte") ;

    system("PAUSE");
    return 0;
}
```

Répète l'instruction qu'elle contient (ici un bloc) tant que la condition mentionnée ( $n \leq 0$ ) est vraie. On ne sait pas a priori combien de fois une telle boucle sera répétée. Toutefois elle est toujours parcourue au moins une fois. En effet, la condition qui régit cette boucle n'est examinée qu'à la fin de chaque répétition.

## TP 28: Menu d'application

Question 1 : Ecrire un programme qui fait l'addition

Donnez X : 10

Donnez Y : 3

Solution de X+Y : 13

Question 2 : Ecrire un programme qui fait la soustraction

Question 3 : Ecrire un programme qui fait les opération arithmétique (+,-,\*,/)

Calculatrice

1 – Addition

2 – Soustraction

3 – Division

4 – Multiplication

Donnez votre choix : 2

Donnez X : 10

Donnez Y : 3

Solution de  $X - Y$  : 7





## Chapitre : 2.2.3 Opérateurs et expressions

### Les conversions implicites pouvant intervenir dans un calcul d'expression

#### Notion d'expression mixte

#### TP 29 : Expression Mixte

Question 1 : Exécuter le code suivant

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int nombre_entier;
    float nombre_float ;

    nombre_entier = 1;
    nombre_float = 2.2;

    printf(" Le résultat est : %f \n", nombre_entier + nombre_float );
    printf(" Le résultat est : %d \n", nombre_entier + nombre_float );

    system("PAUSE");
    return 0;
}
```

Question 2 : Quel est le type de résultat de l'expression : nombre\_entier + nombre\_float

#### Les conversions d'ajustement de type

Une conversion telle que **int -> float** se nomme une « conversion d'ajustement de type ».

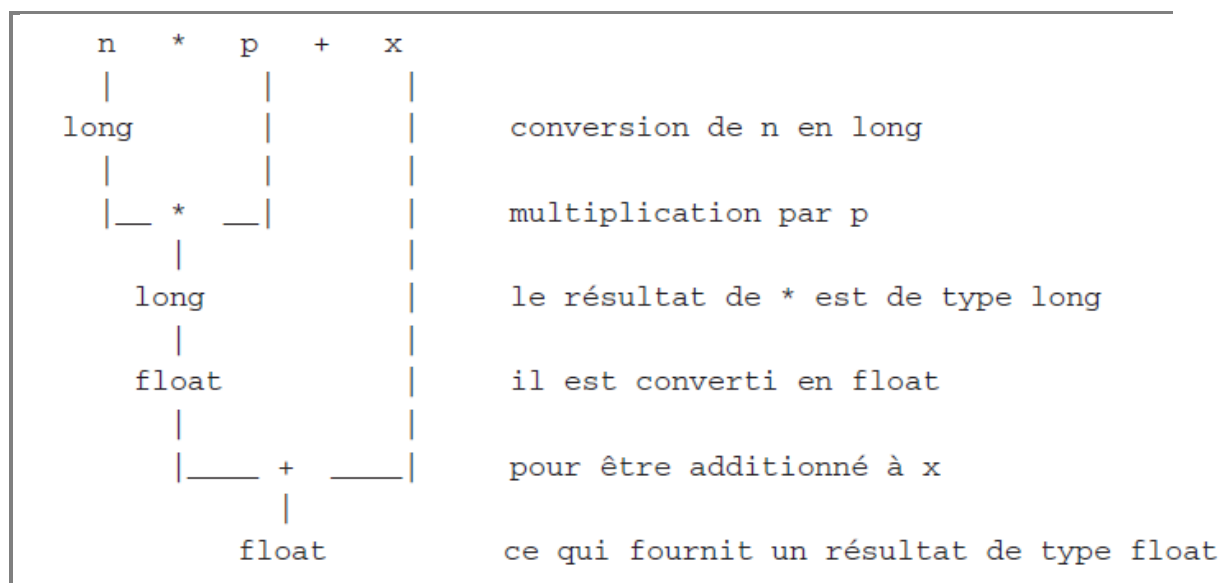
Une telle conversion ne peut se faire que suivant une hiérarchie qui permet de ne pas dénaturer la valeur initiale, à savoir :

`int -> long -> float -> double -> long double`

On peut bien sûr convertir directement un **int** en **double** ; en revanche, on ne pourra pas convertir un **double** en **float** ou en **int**.

Notez que le choix des conversions à mettre en oeuvre est effectué en considérant un à un les opérandes concernés et non pas l'expression de façon globale. Par exemple, si `n` est de type `int`, `p` de type `long` et `x` de type `float`, l'expression :

`n * p + x`



## TP 30 : Conversion implicite

Soit les déclarations suivantes :

```
int n = 10, p = 4 ;
long q = 2 ;
float x = 1.75 ;
```

Question 1 : Donner le type et la valeur de chacune des expressions suivantes :

- a) `n + q`
- b) `n + x`
- c) `n % p + q`
- d) `n < p`
- e) `n >= p`
- f) `n > q`
- g) `q + 3 * (n > p)`

Question 2 : Ecrire un programme qui permet de vérifier les réponse de la question 1

## Les promotions numériques

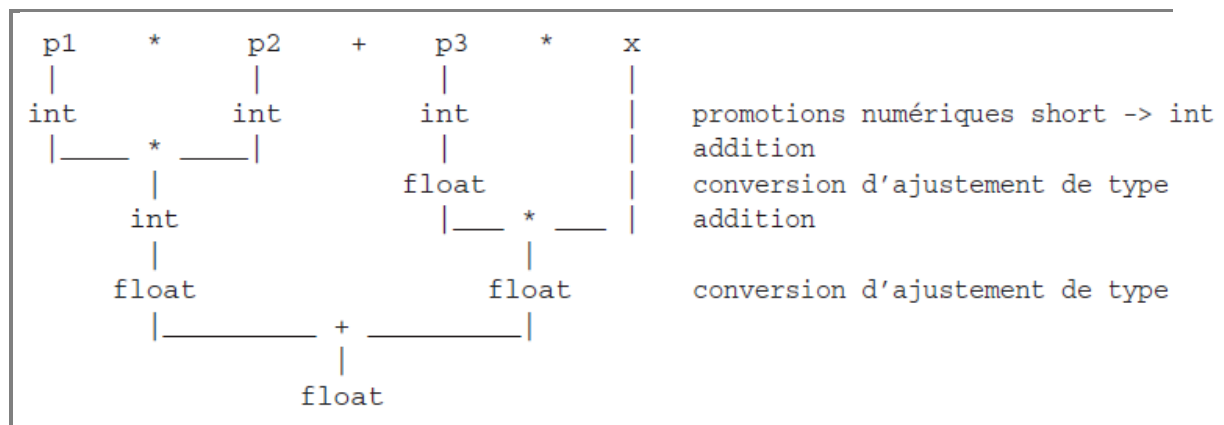
Les conversions d'ajustement de type ne suffisent pas à régler tous les cas. En effet, comme nous l'avons déjà dit, les opérateurs numériques ne sont pas définis pour les types **char** et **short**.

En fait, le langage C prévoit tout simplement que toute valeur de l'un de ces deux types apparaissant dans une expression est d'abord convertie en **int**.

Par exemple, si `p1`, `p2` et `p3` sont de type `short` et `x` de type `float`, l'expression :

```
p1 * p2 + p3 * x
```

est évaluée comme l'indique le schéma ci-après :



## Les opérateurs relationnels

### TP 31 : Valeur Booléen

Exécuter le code suivant en remarquant la valeur des deux expression suivant :

- `a > b`
- `b > a`

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int a,b;
    a = 1;
    b = 2;
    printf("la valeur de a > b = %d \n",a > b );
    printf("la valeur de b > a = %d \n",b > a );
    system("PAUSE");
    return 0;
}
```

Les différents opérateurs relationnels :

OPÉRATEUR	SIGNIFICATION
<	inférieur à
<=	inférieur ou égal à
>	supérieur à
>=	supérieur ou égal à
==	égal à
!=	différent de

Le résultat de la comparaison de  $a > b$  est, non pas une valeur booléenne (on dit aussi logique) prenant l'une des deux valeurs *vrai* ou *faux*, mais un entier valant :

- 0 si le résultat de la comparaison est faux,
- 1 si le résultat de la comparaison est vrai.

## Les opérateurs logiques

C dispose de trois opérateurs logiques classiques : **et** (noté `&&`), **ou** (noté `||`) et **non** (noté `!`).

Par exemple :

$(a < b) \ \&\& \ (c < d)$

prend la valeur 1 (vrai) si les deux expressions  $a < b$  et  $c < d$  sont toutes deux vraies (de valeur 1), et prend la valeur 0 (faux) dans le cas contraire.

$(a < b) \ || \ (c < d)$

prend la valeur 1 (vrai) si l'une au moins des deux conditions  $a < b$  et  $c < d$  est vraie (de valeur 1), et prend la valeur 0 (faux) dans le cas contraire

! (a<b) prend la valeur 1 (vrai) si la condition a<b est fausse (de valeur 0) et prend la valeur 0 (faux) dans le cas contraire. Cette expression est équivalente à : a>=b.

## **TP 32 : opérateurs logiques**

Exécuter le code suivant en remarquant

- le type de la valeur « a > b && d > c »
- la condition de la structure de contrôle « If »
- Le bloc exécuté de la structure de la structure de contrôle « If » ou « else » et pourquoi ?
- La possibilité d'affectation « resultat\_logique = a > b && d > c »

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int a,b,c,d;
    int resultat_logique;
    a = 1; b = 2; c = 3; d = 4;

    printf(" a = %d \n b = %d \n c = %d \n d = %d \n",a,b,c,d);

    printf("la valeur de a > b = %d \n", a > b);
    printf("la valeur de d > c = %d \n", d > c);
    printf("la valeur de a > b && d > c = %d \n", a > b && d > c);

    resultat_logique = a > b && d > c;
    printf("la valeur de a > b && d > c = %d \n", a > b && d > c);

    if (a > b && c > d)
        printf("la valeur de 'a > b && c > d' est différent à 0");
    else
        printf("la valeur de 'a > b && c > d' est égale à 0 \n") ;

    system("PAUSE");
    return 0;
}
```

## L'opérateur d'affectation ordinaire

## Les opérateurs d'incrémentation et de décrémentation

### TP 33 : Pré-Incrémentation et Post-Incrémentation

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Nombre1 = 0,
        Nombre2=0,
        pre_incrementation = 0,
        post_incrementation = 0;

    printf("Nombre1 = %d \n",Nombre1);
    printf("Nombre2 = %d \n\n",Nombre2);

    pre_incrementation = ++Nombre1;
    post_incrementation = Nombre2++;
    printf("pre_incrementation = %d \n",pre_incrementation);
    printf("post_incrementation = %d \n\n",post_incrementation);

    printf("Nombre1 = %d \n",Nombre1);
    printf("Nombre2 = %d \n\n",Nombre2);

    system("PAUSE");
    return 0;
}
```

## Les opérateurs d'affectation élargie

`+=`    `-=`    `*=`    `/=`    `%=`

## **TP 34 : Opérateur d'affectation élargie**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Nombre1 = 1,
        Nombre2= 2,
        pre_incrementation = 0,
        post_incrementation = 0;

    printf("Nombre1 = %d \n",Nombre1);
    printf("Nombre2 = %d \n\n",Nombre2);

    Nombre1 += Nombre2;

    printf("Nombre1 = %d \n",Nombre1);
    printf("Nombre2 = %d \n\n",Nombre2);

    system("PAUSE");
    return 0;
}
```

## **Les conversions forcées par une affectation**

## **TP 35 : Conversion forcée par une affectation**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int NombreInt = 1;
    float  NombreFloat= 2.2;

    printf("NombreInt = %d \n",NombreInt);
    printf("Nombre2 = %f \n\n",NombreFloat);

    NombreInt = NombreFloat + 5;

    printf("Nombre1 = %d \n",NombreInt);
    printf("Nombre2 = %f \n\n",NombreFloat);

    system("PAUSE");
    return 0;
}
```



## L'opérateur de cast

S'il le souhaite, le programmeur peut forcer la conversion d'une expression quelconque dans un type de son choix, à l'aide d'un opérateur un peu particulier nommé en anglais « cast ».

### TP 36 : Utilisation de Cast

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int NombreInt = 1;
    float  NombreFloat= 2.6;

    printf("NombreInt = %d \n",NombreInt);
    printf("Nombre2 = %f \n\n",NombreFloat);

    NombreInt =  NombreFloat + 5.6;
    printf("NombreFloat + 5.6 = %d \n",NombreInt);

    NombreInt = (int) NombreFloat + 5.6;
    printf("Nombre(int) NombreFloatInt = %d \n",NombreInt);

    system("PAUSE");
    return 0;
}
```

## L'opérateur conditionnel

Considérons l'instruction suivante :

```
if ( a>b )
    max = a ;
else
    max = b ;
```

Elle attribue à la variable `max` la plus grande des deux valeurs de `a` et de `b`. La valeur de `max`

Pourrait être définie par cette phrase :

Si `a>b` alors `a` sinon `b`

En langage C, il est possible, grâce à l'aide de l'**opérateur conditionnel**, de traduire presque littéralement la phrase ci-dessus de la manière suivante :

```
max = a > b ? a : b
```

## TP 37 : Opérateur conditionnel

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    float NombreInt = 2;
    float  NombreFloat = 2.6;
    float max = 0;

    max = NombreFloat > NombreInt ? NombreFloat:NombreInt;

    printf("max = %f \n",max);

    system("PAUSE");
    return 0;
}
```

## L'opérateur sizeof

L'opérateur `sizeof`, dont l'emploi ressemble à celui d'une fonction, fournit la taille en octets

(n'oubliez pas que l'octet est, en fait, la plus petite partie adressable de la mémoire).

## TP 38 : Opérateur - sizeof

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int nombreInt = 2;
    long nombreLong = 200000000;
    float nombreFloat = 2.2;
    double nombreDouble = 2.00000002;
    char caractere = 'A' ;

    printf("sizeof (nombreInt) = %d \n", sizeof(nombreInt));
    printf("sizeof (nombreLong) = %d \n", sizeof(nombreLong));
    printf("sizeof (nombreFloat) = %d \n", sizeof(nombreFloat));
    printf("sizeof (nombreDouble) = %d \n", sizeof(nombreDouble));
    printf("sizeof (caractere) = %d \n", sizeof(caractere));

    system("PAUSE");
    return 0;
}
```

## **TP 39 : sur l'opérateur conditionnel**

Écrire plus simplement l'instruction suivante

```
z = (a > b ? a : b) + (a <= b ? a : b) ;
```

## **TP 40 : sur l'opérateur conditionnel**

**n** étant de type **int**, écrire une expression qui prend la valeur :

- -1 si n est négatif,
- 0 si n est 0,
- 1 si n est positif.

## **TP 41 : Exercice sur l'opérateur d'incrément**

Quels résultats fournit le programme suivant ?

```
#include <stdio.h>
main()
{
    int n=10, p=5, q=10, r ;
    r = n == (p = q) ;
    printf ("A : n = %d  p = %d  q = %d  r = %d\n", n, p, q, r) ;
    n = p = q = 5 ;
    n += p += q ;
    printf ("B : n = %d  p = %d  q = %d\n", n, p, q) ;
    q = n < p ? n++ : p++ ;
    printf ("C : n = %d  p = %d  q = %d\n", n, p, q) ;
    q = n > p ? n++ : p++ ;
    printf ("D : n = %d  p = %d  q = %d\n", n, p, q) ;
}
```



## **Partie 2.3 : Tableau et adresse**

## Chapitre 2.3.1 : Tableaux

TP 42 : Saisir et Afficher un tableau

TP 43 : Somme et Moyenne d'un tableau

TP 44 : La valeur maximale d'un tableau

TP 45 : échanger la valeur maximale avec la valeur de la fin du tableau

TP 46 : Boucle avec condition

TP 47 : Trie par sélection

TP 48 : Rechercher dans tableau

TP 49

TP 50

TP 51 : Lire et écrire un tableau de plusieurs dimensions

TP 52 :

TP 53 : Lire et afficher chaîne de caractère

TP 54 : Exercice

TP 55 : strcpy

TP 56 : strcat

TP 57 : strcmp

TP 58 : strcpy

TP 59 : stelen

TP 60 : Exercice 1

TP 61 : Exercice 2

TP 62 : Exercice 3

## Les tableaux à une indice

Exemple d'utilisation d'un tableau en C

## **TP 42 : Saisir et Afficher un tableau**

Question 1 : écrire un programme qui lire 10 note de module 1 de 10 stagiaires

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    float module1[10];
    float module2[10];
    int numero_stagiaire;

    printf("Note du module 1 \n");
    for(numero_stagiaire = 1; numero_stagiaire <= 10; numero_stagiaire++){
        printf("Donnez la note %d :",note);
        scanf("%d",&module1[numero_stagiaire - 1]);
    }

    system("PAUSE");
    return 0;
}
```

Question 2 : afficher le tableau que vous avez saisi

## **TP 43 : Somme et Moyenne d'un tableau**

Question 1 : Afficher la somme des notes du module 1

Question 2 : Afficher la moyenne des notes du module 1

## **TP 44 : La valeur maximale d'un tableau**

Question 1 : Afficher la note maximale

## **TP 45 : échanger la valeur maximale avec la valeur de la fin du tableau**

Échanger la valeur maximale avec la valeur de la fin du tableau

## **TP 46 : Boucle avec condition**

Ecrire un programme qui lire les note de 30 stagiaire

Avec la condition  $0 < \text{note} < 20$

## **TP 47 : Trie par sélection**

Ecrire un programme qui trie un tableau de 10 notes

## **TP 48 : Rechercher dans tableau**

Ecrire un programme qui affiche la position d'une note dans le tableau donnée par l'utilisateur

Exemple

11	12	13	7	17	9	6	13	15	14
----	----	----	---	----	---	---	----	----	----

```
Donner un nombre : 17
```

```
Sa position dans le tableau est : 4
```

## **TP 49**

Donner un programme qui demande un nombre et recherche le nombre d'occurrence (d'existante) dans le tableau

Exemple :

11      12      13      7      17      9      6      13      15      14

```
Donnez un nombre : 13
```

```
Le nombre 13 existe : 2 fois
```

## **TP 50**

Donnez un programme qui affiche la valeur de milieu d'un tableau.

Exemple :

11      12      13      7      17      9      6      13      15      14

```
La valeur de milieu du tableau : 17
```



# Les tableaux à plusieurs indices

## TP 51 : Lire et écrire un tableau de plusieurs dimensions

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    // Colonne 1 : Numéro , Colonne 2 : Age, Colonne 3 : Numéro de tel
    int liste_stagiaires[30][3];
    int indice = 0;

    for(indice = 0; indice < 4; indice++){
        printf("Donnez le numero du stagiaire : ");
        scanf("%d",&liste_stagiaires[indice][0]);
        printf("Donnez l'age du stagiaire : ");
        scanf("%d",&liste_stagiaires[indice][1]);
        printf("Donnez le numeo du Tel ");
        scanf("%d",&liste_stagiaires[indice][2]);
    }

    printf(" \n\t Liste des stagiaires \n");
    for(indice = 0; indice < 4; indice++){
        printf("Stagiaire %d, Age = %d , Tel : %d \n",
               liste_stagiaires[indice][0],
               liste_stagiaires[indice][1],
               liste_stagiaires[indice][2] );
    }
    system("PAUSE");
    return 0;
}
```

## TP 52:

Ecrire un programme qui lire et écrire un matrice de 40 ligne et 2 colonne.

## Chapitre 2.3.2 : Chaînes de caractères

Les chaînes de caractères sont des tableaux de caractères dont le dernier élément est ‘\0’.

### Déclaration

```
char nom_variable[taille] ;
```

#### Exemple :

```
char nom[20] ;  
char prenom[20] ;
```

### Initialisation

On peut initialiser une chaîne de caractères en utilisant l’une des méthodes suivantes :

```
char B[14]= "un deux troix" ;// la taille de cette chaîne de caractères est 14  
char B[]= "un deux troix" ;  
char B[]= "un" "deux" "troix" ;  
char d[10]="x" ; // les restes vont être initialisés par 0  
char d[10]={‘x’,’\0’} ;  
char c[5]={‘a’,’b’,’c’,’d’,’\0’} ;
```

## Manipulation des chaînes de caractères

### La bibliothèque <stdio.h>

- Les fonctions « printf » et « scanf »

#### Exemple :

```
char phrase[10] ;  
printf("saisir une phrase \n ") ; // affichage du message « saisir une phrase »  
scanf("%s",phrase) ; // lecture d’une phrase  
printf("%s",phrase) ; // affichage de la phrase saisie
```

- **Les fonctions « puts » et « gets »**

**Exemple :**

```
char texte [200] ;  
puts("saisir un texte\n ") ; //affichage du message « saisir un texte »  
gets(texte) ; //lecture d'un texte
```

**Remarque :**

La fonction « scanf » permet de lire un texte qui ne contient pas d'espaces par contre la fonction « gets » peut lire un texte de plusieurs lignes contenant des espaces.

## **TP 53 : Lire et afficher chaîne de caractère**

Ecrire un programme qui lire et affiche le nom et le prénom d'un stagiaires

### **La bibliothèque <string.h>**

- **La fonction « strlen »**

C'est une fonction qui retourne la longueur d'une chaîne de caractères sans compter le caractère '\0'

**Exemple 1:**

```
int nb ;  
nb=strlen("bonjour") ;  
printf("la longueur de la chaîne tapée est %d ",nb) ;
```

**Exemple 2 :**

```
char texte[10] ;  
int nb ;  
puts("entrer une chaîne \n") ;  
gets(texte) ;  
nb=strlen(texte) ;  
printf("la longueur de la chaîne tapée est %d ", nb) ;
```

## **TP 54 : Exercice**

Ecrire un programme qui lise et affiche le nom d'un stagiaire en affichant le nombre des caractères.

### ▪ La fonction « strcpy »

**strcpy**(<s>,<t>);

La fonction **strcpy** sert à copier la chaîne t dans la chaîne s.

#### **Exemple :**

```
char mot1[20], mot2[20] ;  
puts("mot1 = ") ;  
gets(mot1) ;  
puts("mot2 = ") ;  
gets(mot2) ;  
strcpy(mot1, mot2) ;  
puts("%s", mot1) ;
```

## **TP 55 : strcpy**

Ecrire un programme qui lise et affiche le nom et le prénom d'un stagiaire. Ensuite échanger le nom avec le prénom.

### ▪ La fonction « strcat »

**strcat**(<s>,<t>);

La fonction **strcat** sert à ajouter la chaîne t à la fin de la chaîne s.

#### **Exemple :**

```
char mot1[20], mot2[20] ;  
puts("mot1 = ") ;  
gets(mot1) ;  
puts("mot2 = ") ;  
gets(mot2) ;  
strcat(mot1, mot2) ;  
puts("%s", mot1) ;
```

## **TP 56 : strcat**

Ecrire un programme qui lise et affiche le nom et le prénom d'un stagiaires. Ensuite enregistrez le nom et le prénom dans une autre chaîne de caractère avec un espace entre le nom et le prénom.

### ▪ La fonction « strcmp »

**strcmp(<s>,<t>) ;**

Cette fonction compare lexicographiquement deux chaînes de caractères.

- Si strcmp(<s>,<t>) <0 alors la chaîne s précède la chaîne t.
- Si strcmp(<s>,<t>) = 0 alors la chaîne s égale la chaîne t.
- Si strcmp(<s>,<t>) >0 alors la chaîne t précède la chaîne s.

### **Exemple :**

```
char mot1[20], mot2[20] ;  
puts("mot1 = "); gets(mot1) ;  
puts("mot2 = "); gets(mot2) ;  
if (strcmp(mot1, mot2) < 0)  
    printf("%s précède %s ", mot1, mot2) ;  
else  
    if (strcmp(mot1, mot2) > 0)  
        printf("%s précède %s ", mot2, mot1) ;  
    else  
        printf("%s égale à %s", mot1, mot2) ;
```

## **TP 57 : strncpy**

Ecrire un programme qui lise et affiche le nom de deux ville du Maroc et vérifier si les deux villes sont égale.

### ▪ La fonction « strncpy »

La fonction **strncpy** sert à copier n caractères de la chaîne t dans la chaîne s.

**strncpy(<s>,<t>,<n>);**

- **La fonction « strncat »**

Cette fonction permet d'ajouter <n> caractères de la chaîne <t> à la fin de la chaîne <s>.

```
strncat(<s>,<t>,<n>);
```

## **TP 58 : strcpy**

Ecrire un programme qui lire le nom et le prénom d'un stagiaires ensuite ajoutez les trois première lettre du nom et prénom dans une autre chaîne de caractère.

Exemple

Nom = Madani , Prénom = Ali , Nom\_Prenom = MadAli

## **La bibliothèque <ctype.h>**

- **La fonction « isupper »**

Cette fonction permet de vérifier si le caractère saisi comme paramètre est un caractère majuscule ou non.

**isupper(<c>)** <> 0 si c est majuscule.

- **La fonction « islower »**

Cette fonction permet de vérifier si le caractère saisi comme paramètre est un caractère minuscule ou non.

**islower(<c>)** <> 0 si c est minuscule.

- **La fonction « isdigit »**

Cette fonction permet de vérifier si le caractère saisi comme paramètre est un entier compris entre 0 et 9.

**isdigit(<c>)** <> 0 si c est un entier.

- **La fonction « isspace »**

Cette fonction permet de vérifier si le caractère saisi comme paramètre est un signe d'espace (' ', \t, \n, \f, \r)

**isspace(<c>)** <> 0 si **c** est un signe d'espace.

#### ▪ La fonction « tolower »

Cette fonction permet de convertir le caractère saisi comme paramètre à un caractère minuscule s'il est majuscule.

**tolower(<c>)** retourne **c** en minuscule s'il est majuscule.

#### ▪ La fonction « toupper »

Cette fonction permet de convertir le caractère saisi comme paramètre à un caractère majuscule s'il est minuscule.

**toupper(<c>)** retourne **c** en majuscule s'il est minuscule.

## **TP 59 : stelen**

Ecrire un programme qui lit une ligne de texte (ne dépassant pas 200 caractères) la mémorise dans une variable TXT et affiche ensuite:

- a) la longueur de la chaîne.
- b) le nombre de lettre 'e' contenus dans le texte.

## **TP 60 : Exercice 1**

Ecrire un programme qui lit un texte TXT (de moins de 200 caractères) et qui enlève toutes les apparitions du caractère 'a' en tassant les éléments restants. Les modifications se feront dans la même variable TXT.

## **TP 61 : Exercice 2**

Ecrire un programme qui lit deux chaînes de caractères CH1 et CH2, les compare alphabétiquement et affiche le résultat.

## **TP 62 : Exercice 3**

Ecrire un programme qui lit deux chaînes de caractères CH1 et CH2 et qui copie la première moitié de CH1 et la première moitié de CH2 dans une troisième chaîne CH3. Afficher le résultat.







## Partie 2.4 : Programmation modulaire

Comme tous les langages, C permet de découper un programme en plusieurs parties nommées souvent « modules ». Cette programmation dite modulaire se justifie pour de multiples raisons :

- Un programme écrit d'un seul tenant devient difficile à comprendre dès qu'il dépasse une ou deux pages de texte.
  - Une écriture modulaire permet de le scinder en plusieurs parties et de regrouper dans le programme principal les instructions en décrivant les enchaînements.
  - Chacune de ces parties peut d'ailleurs, si nécessaire, être décomposée à son tour en modules plus élémentaires
- La programmation modulaire permet d'éviter des séquences d'instructions répétitives, et cela d'autant plus que la notion d'argument permet de paramétrer certains modules.
- La programmation modulaire permet le partage d'outils communs qu'il suffit d'avoir écrits et mis au point une seule fois.

## Chapitre 2.4.1 : Structures

### TP 63 : Déclaration d'une structure

#### 1. Introduction

Il est souvent nécessaire en programmation de stocker des informations complexes. Par exemple une liste des fiches des stagiaires.

Chaque fiche rassemble les informations caractéristiques d'une **entité** : nom, prénom, adresse, date de naissance, téléphone.

Chaque champ d'une fiche est un **attribut**, exactement défini par son **type** (les valeurs acceptables, espace mémoire occupé), et sa valeur.

L'information est structurée, puisque chaque fiche est organisée d'une manière figée. Le répertoire des fiches des stagiaires lui-même est une collection de fiches de même **structure**.

Entité : Stagiaire					
					Identificateur
attribut	Nom	Prénom	Adresse	Date de nais Date	Téléphone
[TAILLE]	20 car.	20 car.	40 car.		10 num.
Valeur 1	Madani	Ali	Tanger, Maroc	01/01/2009	0600000001
Valeur 2	Madani	Mouad	Rabat, Maroc	01/02/2008	0600000002

Déclaration

## TP 63 : Déclaration d'une structure

Question 1 : exécuter le code suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct str_stagiaire {
    char nom[20];
    char prenom[20];
    char tel[10];
};

typedef struct str_stagiaire stagiaire;

int main(int argc, char *argv[])
{
    struct str_stagiaire stagiaire1;
    strcpy(stagiaire1.nom, "Madani");
    strcpy(stagiaire1.prenom, "Ali");

    puts(stagiaire1.nom);
    puts(stagiaire1.prenom);

    stagiaire stagiaire2;
    strcpy(stagiaire2.nom, "Madani");
    strcpy(stagiaire2.prenom, "Mouad");
    puts(stagiaire2.nom);
    puts(stagiaire2.prenom);

    system("PAUSE");
    return 0;
}
```

Question 2 : que fait l'instruction « typedef »

Question 3 : Tester le même code avec les instruction suivant :

```
typedef struct str_stagiaire {
    char nom[20];
    char prenom[20];
    char tel[10];
} stagiaire ;
```

Question 4 : Est-ce que on peut supprimer le nom de la structure dans l'instruction précédent

```
typedef struct {
    char nom[20];
    char prenom[20];
    char tel[10];
} stagiaire ;
```

On peut aussi inclure dans la définition d'une structure la référence à d'autres structures, par une sorte d'imbrication. Par exemple après avoir défini un type de données date plus approprié, s'y référer dans la définition d'un type personne.

**typedef struct**

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

```
Typedef struct  
{  
    char nom[20];  
    char prenom[10];  
    char adresse[40];  
    date date_niassance;  
    char tel[10];  
} stagiaire;
```

Un stagiaire est alors une variable occupant 86 octets en mémoire (20+10+40+6+10).

La mise à jour de la fiche du Madani Ali. est réalisée par les instructions :

```
stagiaire stagiaire1  
strcpy(stagiaire1.nom, "Madani");  
strcpy(stagiaire1.prenom, "Ali");  
strcpy(stagiaire1.adresse, "Maroc, Tanger");  
stagiaire1.date_niassance.jour=1;  
stagiaire1.date_niassance.mois=1;  
stagiaire1.date_niassance.annee=2001;  
strcpy(stagiaire1.tel, "0600000001");
```

## TP 64 : Exercice 1

Donnez un programme qui permet de saisir et afficher un stagiaire avec ( Nom, Prénom, Adresse, Tel, DateNaissance)

## Structures et tableaux

On peut définir un tableau de structures (mais ceci est assez peu utilisé).

**Exemple :** (à partir de la structure définie précédemment).

### 2.1.Déclaration

Stagiaire T[30];     /\* on déclare un tableau de 30 stagiaires\*/

### 2.2.Utilisation

```
strcpy(T[i].nom,"Madani")     /* pour un indice i quelconque */  
strcpy(f[i].prenom,"Ali");
```

## TP 65 : Tableau de structure

```
Créer une structure   point {  
int num;  
float x;  
float y;  
}
```

Saisir 4 points, les ranger dans un tableau puis les afficher.

## TP 66 : Exercice

Question 1 : Ecrire un programme qui lire et affiche 30 Stagiaires

Stagiaire(Numéro,Nom,Prénom,DateNaissance(Jour,Mois,Année))

Question 2 : Afficher le stagiaire le plus âgée

## **Partie 2.5 : Fonction**

## Chapitre 2.5.1 Fonction

### Exemple de définition et d'utilisation d'une fonction

#### TP 67 : Bonjour fonction

```
#include <stdio.h>
#include <stdlib.h>
int sommeDeMinAMax( int min, int max){
    int somme = 0, i;
    for( i = min; i <= max ; i++) {
        somme += min++ ;
    }
    return somme;
}
int main(int argc, char *argv[])
{
    printf("Somme de 1 à 10 = %d \n", sommeDeMinAMax(1,10));
    printf("Somme de 20 à 30 = %d \n", sommeDeMinAMax(20,30));
    printf("Somme de 1 à 5 = %d \n", sommeDeMinAMax(1,5));

    system("PAUSE");
    return 0;
}
```

#### TP 68 : Fonction Somme

Ecrire une fonction qui calcule la somme de deux nombre.

Utiliser cette fonction pour calculer la somme suivant :

1 + 2

1.2 + 103.5

501.5 + 506.5



## **TP 69 : Le Menu d'application et fonctions**

Question 1 : Ecrire les fonctions de calcul arithmétique : Somme +, Multiplication \*, Soustraction -, Modulo %

Question 2 : Ecrire un programme avec Menu qui permet de choisir la fonction à utiliser.

### **L'instruction `return`**

- L'instruction `return` peut mentionner n'importe quelle expression.
- L'instruction `return` peut apparaître à plusieurs reprises dans une fonction
  - Notez bien que non seulement l'instruction `return` définit la valeur du résultat, mais, en même temps, elle interrompt l'exécution de la fonction
- Si le type de l'expression figurant dans `return` est différent du type du résultat tel qu'il a été déclaré dans l'en-tête, le compilateur mettra automatiquement en place des instructions de conversion.

## **TP 70 : l'instruction `return`**

Donnez le résultat de l'exécution du code suivant :

```
#include <stdio.h>
#include <stdlib.h>

int somme (float nombre1, float nombre2) {
    float somme;
    if (nombre1 > nombre2 )
        return nombre1 + nombre2;
    else if (nombre2 > nombre1 )
        return nombre2 - nombre1 ;
    return 1;
}

int main(int argc, char *argv[])
{

    printf("Somme 1 : %d \n", somme(2.5,1.6));
    printf("Somme 2 : %d \n", somme(1.5,1.6));
    printf("Somme 3 : %d \n", somme(1.5,1.5));

    system("pause");
    return 0;
}
```

## **Cas des fonctions sans valeur de retour**

### **TP 71 : Fonction du menu principal**

Question 1 : Exécuter le code suivant :

```
#include <stdio.h>
#include <stdlib.h>

void menu_principal() {
    printf("-- Calculatrice -- \n\n");
    printf("\t- Somme de a et b \t-- 1 \n");
    printf("\t- Produit de a et b \t-- 2 \n");

    printf("\n \t\t Donnez votre choix : ");
}

int main(int argc, char *argv[])
{
    int choix;
    menu_principal();
    scanf("%d",&choix);
    printf("votre choix est : %d \n",choix);
    system("PAUSE");
    return 0;
}
```

Question 2 : utiliser la fonction « menu\_principale » pour réaliser un programme de calculatrice contenant les quatre opération arithmétique.

## Les arguments sont transmis par valeur

### TP 72 : Transmission par valeur

Donnez l'exécution du code suivant :

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    void echange (int a, int b) ;
    int n=10, p=20 ;
    printf ("avant appel : %d %d\n", n, p) ;
    echange (n, p) ;
    printf ("après appel : %d %d", n, p);

    system("PAUSE");
    return 0;
}

void echange (int a, int b)
{
    int c ;
    printf ("début echange : %d %d\n", a, b) ;
    c = a ;
    a = b ;
    b = c ;
    printf ("fin echange : %d %d\n", a, b) ;
}
```

## Les variables globales

Nous avons vu comment échanger des informations entre différentes fonctions grâce à la transmission d'arguments et à la récupération d'une valeur de retour.

En fait, en C, plusieurs fonctions (dont, bien entendu le programme principal main) peuvent partager des variables communes qu'on qualifie alors de globales.

### TP 73 : Exemple d'utilisation d'une variable globale

```
#include <stdio.h>
#include <stdlib.h>

float pi = 3.14;

float surface_cercle(float rayon){
    return rayon * rayon * pi ;
}

int main(int argc, char *argv[])
{
    printf("%f", surface_cercle(2.2));
    system("PAUSE");
    return 0;
}
```



## Chapitre 2.5.2 : Variable & Adresse

### Variable

Dans le cours de l'algorithmique nous avons expliqué la notion de la variable

### Adresse d'une variable

Nous avons vu que chaque variable a 4 caractéristiques

- Valeur
- Taille : Type
- Nom
- Adresse

### L'opérateur &

#### TP 74 :

Donnez l'affichage du code suivant

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int nombre = 10;
    printf("%d", &nombre);
    system("PAUSE");
    return 0;
}
```

### L'opérateur \*

#### TP 75

Donnez le tableau d'exécution du code suivant

```
Int a ;
Int * b ;
Scanf("%d",&a) ;
a = 20 ;
```

```
b = &a ;  
a++ ;  
(*b) ++ ;  
Printf('%d',a) ;
```

## Passage des paramètres

### Passage par valeur

### Passage par adresse

## TP 76

Ecrire la fonction Décrémenter qui décrémente un nombre entier.

## TP 77 Fonction Lire et Ecrire un tableau

Question 1 : Ecrire une fonction qui lire un tableau

```
Void LireNoteModule(float* module,int nombre_stagiaire)  
  
void LireNoteModule(float module[100], int nombre_stagiaire){  
    int note;  
    for( note = 1; note < nombre_stagiaire; note++){  
        printf("Donnez la note %d : ",note);  
        scanf("%f",&module[note - 1]);  
    }  
}
```

Question 2 : écrire une fonction qui affiche un tableau des notes

```
void AfficherNoteModule(float module[100], int nombre_stagiaire){  
    int note;  
    printf(" Notes du module \n");  
    for(note = 1; note < nombre_stagiaire; note++){  
        printf(" Note %d = %f \n",note,module[note - 1]);  
    }  
}
```

Question 3 : écrire le **programme principal** qui **teste** les deux fonctions.

```
int main(int argc, char *argv[])  
{  
    float module1[30];  
    LireNoteModule (module1, 5);  
    AfficherNoteModule (module1, 5);  
    system("PAUSE");  
    return 0;  
}
```

## TP 78

Ecrire deux fonctions la première pour lire et l'autre pour afficher un tableau des stagiaires (Id,Nom,Prénom,DateNaissance)

Question 1 : Saisie d'un tableau des stagiaires

Question 2 : Affichage d'un tableau des stagiaires

## TP 79

Ecrire une fonction qui ajouter un stagiaire à la fin d'un tableau des stagiaires.

## TP 80

Ecrire une fonction qui supprimer un stagiaire d'un tableau des stagiaires.

## TP 81:

Ecrire un programme de gestion des stagiaires avec les méthodes suivant :

- AjouterStagiaire : permet d'ajouter un stagiaire à un tableau des stagiaires
- ModifierStagiaire : permet de modifier un stagiaire
- SupprimerStagiaire : permet de supprimer un stagiaire
- RechercheStagiaire : permet de recherche un stagiaire par son nom





## **Partie 2.6 : Fichier et Mini-Projet**

## Chapitre 6.1 : Fichier

Les entrées-sorties conversationnelles apparaîtront comme un cas particulier de la gestion de fichiers

### Type des fichiers

Les fichiers sont soit:

- **binares** (un float sera stocké comme il est codé en mémoire, d'où gain de place mais incompatibilité entre logiciels)
- **formaté ASCII** (un float binaire sera transformé en décimal puis on écrira le caractère correspondant à chaque chiffre).

## Fichier Formaté ASCII

### TP : 82 : fopen, fprintf et fscanf

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  int main(){
4  FILE* f;
5  int n;
6  f = fopen("fichier.txt","w");
7  for(int i = 0;i< 10;i++)
8      fprintf(f,"%d",i);
9
10 fclose(f);
11 f = fopen("fichier.txt","r");
12 for(int i = 0;i< 10;i++){
13     fscanf(f,"%d",&n);
14     printf("%d\n",n);
15 }
16 system("pause");
17 return 0;
18 }
```

## Fichier binaires

### TP : 83 : fread et fwrite

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  int main(){
4  FILE* f;
5  int n;
6  f = fopen("fichierb.txt","wb");
7  for(int i = 0;i< 10;i++)
8      fwrite(&i,sizeof(int),1,f);
9
10 fclose(f);
11 f = fopen("fichierb.txt","rb");
12 while(!feof(f)){
13     fread(&n,sizeof(int),1,f);
14     if(!feof(f)) printf("%d\n",n);
15 }
16 system("pause");
17 return 0;
18 }
```

## TP : 84 : Fichier et Tableau

```
6  int T[100];
7  printf("Tableau");
8  for(int i=0;i<20;i++){
9      T[i] = i*i;
10 }
11
12 f = fopen("fichierb.txt","wb");
13
14 fwrite(T,sizeof(int),20,f);
15 fclose(f);
16
17 f = fopen("fichierb.txt","rb");
18 fread(T,sizeof(int),20,f);
19 |
20 for(int i=0;i<10;i++){
21     printf("%d\n",T[i]);
22 }
23
24
25 system("pause");
26 return 0;
27 }
28
```

## Parcourir

Pour accéder aux données, on le fait soit:

- De manière **séquentielle** (on accède au contenu dans l'ordre du stockage)
- Par un **accès direct** (on peut directement accéder à n'importe quel endroit du fichier)

## Déclaration d'un fichier, Nom Logique et Physique

```
FILE* nom_logique ;
```

Un fichier possède un **nom physique** (connu du système d'exploitation). Pour utiliser un fichier dans un programme, on lui associe un **nom logique** (le nom de la variable de type fichier).

Cette variable ne correspond encore à aucun fichier physique.

Un objet de type FILE \* est appelé flot de données (en anglais, stream). Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une mémoire tampon (buffer), ce qui permet de réduire le nombre d'accès aux périphériques (disque...).

## Ouverture d'un fichier

```
nom_logique = fopen(" nom_physique ", " mode_ouverture ")
```

***nom\_logique*** = ***fopen***(" ***nom\_physique*** ", " ***mode\_ouverture*** ")

Nom de la variable  
de type fichier

Nom physique du  
fichier sur le disque

Mode d'ouverture  
du fichier

- Cette fonction, de type FILE\* ouvre un fichier et lui associe un flot de données.
- Si l'exécution de cette fonction ne se déroule pas normalement, la valeur retournée est le pointeur NULL.
- Le second argument, *mode*, est une chaîne de caractères qui spécifie le mode d'accès au fichier ouverture = positionnement début de fichier (sauf pour 'a+') )

## Mode d'accès au fichier

"r"	ouverture d'un fichier texte en lecture
"w"	ouverture d'un fichier texte en écriture
"a"	ouverture d'un fichier texte en écriture à la fin
"rb"	ouverture d'un fichier binaire en lecture
"wb"	ouverture d'un fichier binaire en écriture
"ab"	ouverture d'un fichier binaire en écriture à la fin
"r+"	ouverture d'un fichier texte en lecture/écriture
"w+"	ouverture d'un fichier texte en lecture/écriture
"a+"	ouverture d'un fichier texte en lecture/écriture à la fin
"r+b"	ouverture d'un fichier binaire en lecture/écriture
"w+b"	ouverture d'un fichier binaire en lecture/écriture
"a+b"	ouverture d'un fichier binaire en lecture/écriture à la fin

- Les spécificateurs de mode d'accès diffèrent suivant le type de fichier considéré ( *fichiers textes, fichiers binaires* )

Ouverture d'un fichier : *Mode d'accès au fichier* :

- Si le mode contient la lettre r, le fichier doit exister.
- Si le mode contient la lettre w, le fichier peut ne pas exister. Dans ce cas, il sera créé.
- Si le fichier existe déjà, son ancien contenu sera perdu.
- Si le mode contient la lettre a, le fichier peut ne pas exister. Dans ce cas, il sera créé.

- Si le fichier existe déjà, les nouvelles données seront ajoutées à la fin du fichier précédent.

## Fermeture

```
fclose(flot)
```

La fonction **fclose** réalise ce que l'on nomme une fermeture de fichier. Elle force l'écriture sur disque du tampon associé au fichier.

Où *flot* est le flot de type FILE\* retourné par la fonction fopen correspondant

La fonction retourne un entier qui vaut zéro si l'opération s'est déroulée normalement et une valeur non nulle en cas d'erreur.

## Fin d'un fichier, feof

```
feof( <FP> );
```

Détection de la fin d'un fichier - feof

**feof** retourne une valeur différente de zéro, si la tête de lecture du fichier référencé par <FP> est arrivée à la fin du fichier;

Sinon la valeur du résultat est zéro.

<FP> est un pointeur du type FILE\* qui est relié au nom du fichier à lire.

## Fichiers textes : Ecriture

```
int fprintf(FILE* fic, char* chaîne_format, liste de données)
```

La fonction fprintf, analogue à printf, permet d'écrire des données dans un fichier

**Exemple :**

```
fprintf(fichier, ``Salut``);  
fprintf(fichier, ``Bonjour %d``, i);
```

Les spécifications de format utilisées pour la fonction fprintf sont les mêmes que pour printf

## Fichiers textes : Lecture

```
int fscanf(FILE* fic, char* format, liste d'adresses)
```

Lit les données en suivant le format dans le fichier fic

**Exemple :**

```
fscanf(fichier, ``%d``, &i);
```

## Fichiers binaires : Lecture

```
int fread(void* p, int taille_bloc, int nb_bloc, FILE* fich)
```

Lit les nb\_blocs de taille taille\_bloc pointés par p dans le fichier fich retourne le nombre de blocs lus et 0 si fin de fichier

Exemple :

```
int tab[3];  
fread(tab, sizeof(int), 3, fichier);
```

## Fichiers binaires : Ecriture

```
fwrite (&n, sizeof(int), 1, sortie) ;
```

fwrite permet de transférer plusieurs blocs consécutifs de même taille à partir d'une adresse donnée.

La fonction fwrite possède quatre arguments précisant :

- l'adresse d'un bloc d'informations (ici &n) ;
- la taille d'un bloc, en octets : ici sizeof(int) ; notez l'emploi de l'opérateur sizeof qui assure la portabilité du programme ;
- le nombre de blocs de cette taille que l'on souhaite transférer dans le fichier (ici 1) ;
- l'adresse de la structure décrivant le fichier (sortie).

La fonction fwrite fournit le nombre de blocs effectivement écrits.

## L'accès direct

```
fseek ( entree, sizeof(int)*(num-1), SEEK_SET) ;
```

- SEEK\_SET (en général 0) : désigne un déplacement (en octets) depuis le **début du fichier**
- SEEK\_CUR (en général 1) : désigne un déplacement exprimé à partir de la **position courante**
- SEEK\_END (en général 2) : le second argument désigne un déplacement depuis la **fin du fichier**

## Taille de Fichier

```
long taille ;  
fseek (entree, 0, SEEK_END) ;  
taille = ftell (entree) ;
```



il suffit de vous positionner en fin de fichier avec `fseek`, puis de faire appel à la fonction `ftell` qui restitue la position courante du pointeur de fichier.

**Exemple : Accès séquentielle d'un fichier**

```
#include <stdio.h>
main()
{
    char nomfich[21] ;
    int n ;
    FILE* sortie ;
    printf ("nom du fichier à créer : ") ;
    scanf ("%20s", nomfich) ;
    sortie = fopen (nomfich, "w") ;
    do {
        printf ("donnez un entier : ") ;
        scanf ("%d", &n) ;
        if (n != 0) fwrite (&n, sizeof(int), 1, sortie) ;
    }
    while (n != 0) ;
    fclose (sortie) ;
}
```

**Exemple : Liste séquentielle d'un fichier**

```
#include <stdio.h>
main()
{
    char nomfich[21] ;
    int n ;
    FILE * entree ;
    printf ("nom du fichier à lister : ") ;
    scanf ("%20s", nomfich) ;
    entree = fopen (nomfich, "r") ;
    while(!feof(entree )){
        int n;
        fread(&n,sizeof(int),1, entree );
        if(!feof(entree )) printf("%d\n",n);
    };
    printf ("\n%d", n) ;
    fclose (entree) ;
}
```

**Exemple : Accès direct**

```
#include <stdio.h>
```

```
main()
{
    char nomfich[21] ;
    int n ;
    long num ;
    FILE * entree ;
    printf ("nom du fichier à consulter : ") ;
    scanf ("%20s", nomfich) ;
    entree = fopen (nomfich, "r") ;
    while (
        printf (" numéro de l'entier recherché : "),
        scanf ("%ld", &num), num )
    {
        fseek (entree, sizeof(int)*(num-1), SEEK_SET) ;
        fread (&n, sizeof(int), 1, entree) ;
        printf (" valeur : %d \n", n) ;
    }
    fclose (entree) ;
}
```

**Exemple : Création d'un fichier en accès direct**

```
#include <stdio.h>
main()
{ char nomfich[21] ;
  FILE * sortie ;
  long num ;
  int n ;
  printf ("nom fichier : ") ;
  scanf ("%20s", nomfich) ;
  sortie = fopen (nomfich, "w") ;
  while (printf ("\nrang de l'entier : "), scanf ("%ld", &num),
    num)
  { printf ("valeur de l'entier : ") ;
    scanf ("%d", &n) ;
    fseek (sortie, sizeof(int)*(num-1), SEEK_SET) ;
    fwrite (&n, sizeof(int), 1, sortie) ;
  }
  fclose(sortie) ;
}
```

**Exercices 1**

Écrire un programme permettant d'afficher le contenu d'un fichier texte en numérotant les lignes.

Ces lignes ne devront jamais comporter plus de 80 caractères.

**Exercice 2 :**

Écrire un programme permettant de créer séquentiellement un fichier « répertoire » comportant

pour chaque personne :

- nom (20 caractères maximum) ;
- prénom (15 caractères maximum) ;
- âge (entier) ;
- numéro de téléphone (11 caractères maximum).

Les informations relatives aux différentes personnes seront lues au clavier.

**Exercice 3 :**

Écrire un programme permettant, à partir du fichier créé par l'exercice précédent, de retrouver les informations correspondant à une personne de nom donné.

**Exercice 4 :**

Écrire un programme permettant, à partir du fichier créé dans l'exercice 2, de retrouver les informations relatives à une personne de rang donné (par accès direct).

Création séquentielle d'un fichier

# Mini-Projet : Gestion des personnes

## Etape 1 : Programme 1

- Saisie de N Personnes
- Enregistrement de N Personnes dans un fichier

## Etape 2 : Programme 2

- Fonction de saisie de N Personnes
- Fonction d'enregistrement de N Personnes
- Programme de Test

## Etape 3 : Programme 3

- Fonction d'ouverture d'un fichier de N Personnes
- Fonction d'affichage de N Personnes

## Etape 4 : Programme 4

- Menu D'application