

ة جنوب تاينقتلا و مولعلا ةيـلـك

Faculté des Sciences et Techniques de Tanger

Département Génie Informatique

# Module : Algorithmique & Programmation 1

Niveau Tronc Commun MIPC

## Cours préparé par :

- Pr. Sanae KHALI ISSA
- Pr. Ayoub AMRANI

## Enseigné par :

- Pr. Ayoub AMRANI
- Email: [a.amrani1@uae.ac.ma](mailto:a.amrani1@uae.ac.ma)

# Déroulement du cours

- Généralités
- Les bases de l'algorithmique
- Les structures alternatives
  - La structure alternative **simple**
  - La structure alternative **complète**
  - La structure alternative **imbriquée**
  - La structure alternative à **choix multiple**
- Les structures de répétition
  - La boucle **Pour ... Faire ... FinPour**
  - La boucle **Répéter ... Jusqu'à**
  - La boucle **Tantque ... FinTantque**
- Les tableaux

# Chapitre 1 : Généralités

# Chapitre 1 : Généralités

**Informatique** : c'est la science du **traitement** automatique et rationnel de **l'information**

**Information** : c'est est un ensemble des **données** portant une **connaissance**.

Une **information** peut être de type :



Texte



Image



Vidéo



Son



Symbol

# Chapitre 1 : Généralités

**Exemple 1 :** Le site web de la FST contient un ensemble d'informations (Listes, Résultats, etc.)

The screenshot shows the homepage of the Faculté des Sciences et Techniques de Tanger (FST) website. At the top, there is a banner for the installation of a 2.268 kW photovoltaic system. Below the banner, several news items are listed:

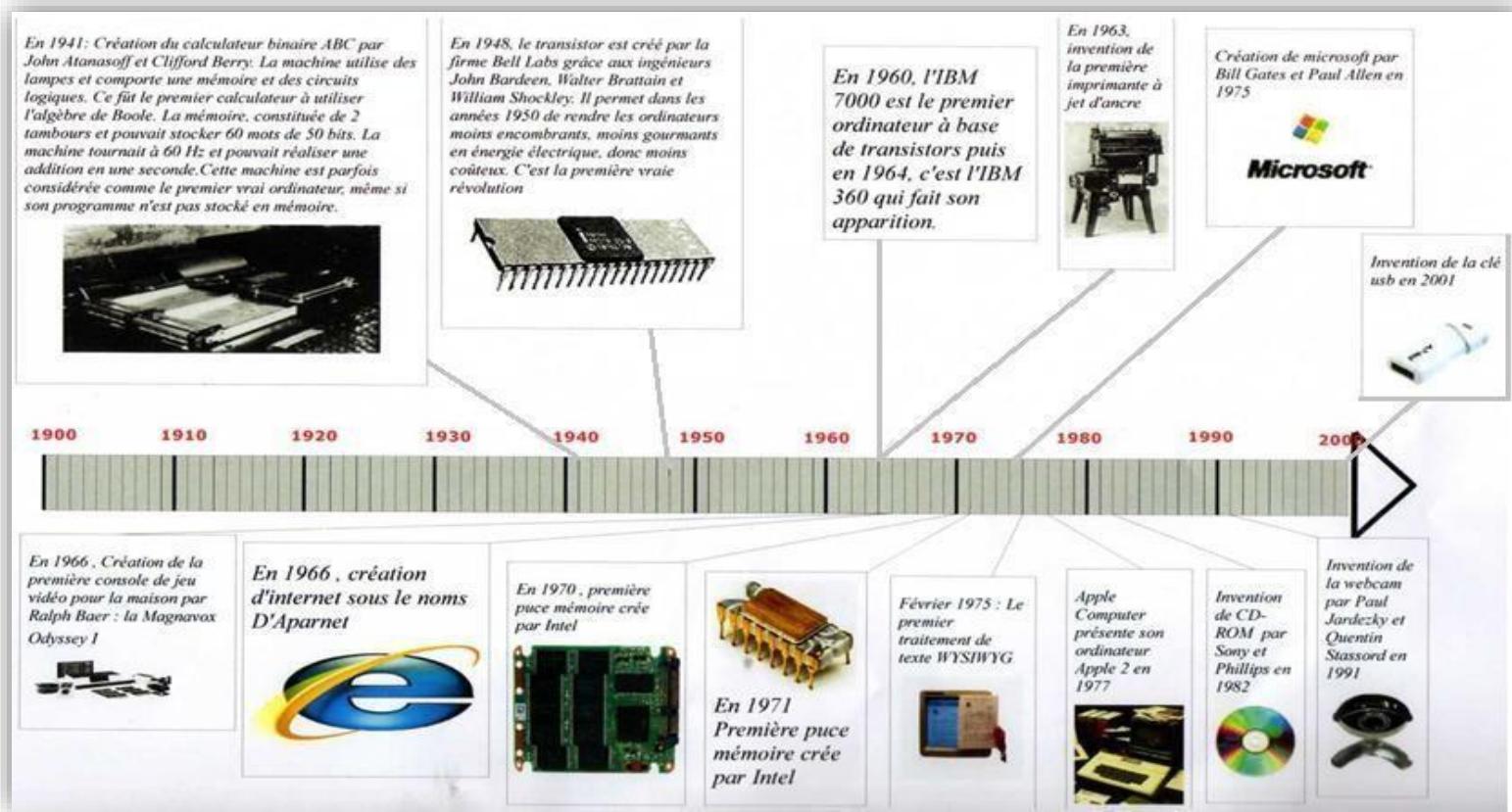
- RÉSULTATS DÉFINITIFS DU CONCOURS D'ACCÈS AU CYCLE MASTER : SCIENCE DE L'ENVIRONNEMENT\_ LISTE D'ATTENTE** (jeudi 29 septembre 2016)  
Liste d'attente du concours d'accès en 1ère année du Master : Science de l'Environnement (SE)  
Nombre de places restantes : 7 places - Les insc...
- OFFRE EN FORMATION CONTINUE AU TITRE DE L'ANNÉE UNIVERSITAIRE 2016/2017** (jeudi 29 septembre 2016)  
Au titre de l'année universitaire 2016-2017, la Faculté des Sciences et Techniques de Tanger offre dix-huit filières en formations continues couronnées pa...
- LISTES DES GROUPES ET AFFECTATIONS PROVISOIRES DES MODULES DU SEMESTRE S1 DU CYCLE LICENCE : ACTUALISÉES LE 27 / 09 / 2016** (mardi 13 septembre 2016)  
Listes des étudiants inscrits en cycle LST avec affectations des modules des semestres d'automne de l'année universitaire 2016-2017 : - Semestre S...
- JOURNÉE SENSIBILISATION DES NOUVEAUX ÉTUDIANTS** (vendredi 23 septembre 2016)  
L'association Génération Future Tanger (GFT) en collaboration avec la FSTT vous proposent 2 Activités : Une présentation sous le titre "Phase..."

On the right side of the page, there are two columns of links:

- CATALOGUES DES FORMATIONS CONTINUES**
  - DCESS
  - FORMATION A LA CARTE
  - LICENCE CNAM
  - DCA
- SOUTENANCES**
  - > Thèses soutenues
  - > Habilitations Universitaires soutenues
- TROUVEZ NOUS SUR FACEBOOK**
  - Faculté des Sciences
  - J'aime déjà 9.7 K mentions

# Chapitre 1 : Généralités

**Exemple 2** : L'histoire de l'informatique est représentée par des images, des schémas et du texte.



# Chapitre 1 : Généralités

**Exemple 3 :** Les informations sur la route sont présentées sous forme des symboles (Vitesse, limite, Rond -point, etc.)



# Chapitre 1 : Généralités

**Le traitement** est l'ensemble des opérations et des techniques appliquées sur des données pour passer d'un état initial vers un état final.



**Traitement automatique :** c'est un traitement qui se fait à l'aide des machines

# Chapitre 1 : Généralités

**Exemple 1 :** Modification de couleur avec un logiciel de Traitement d'image



Image en couleur

Image en noir et blanc



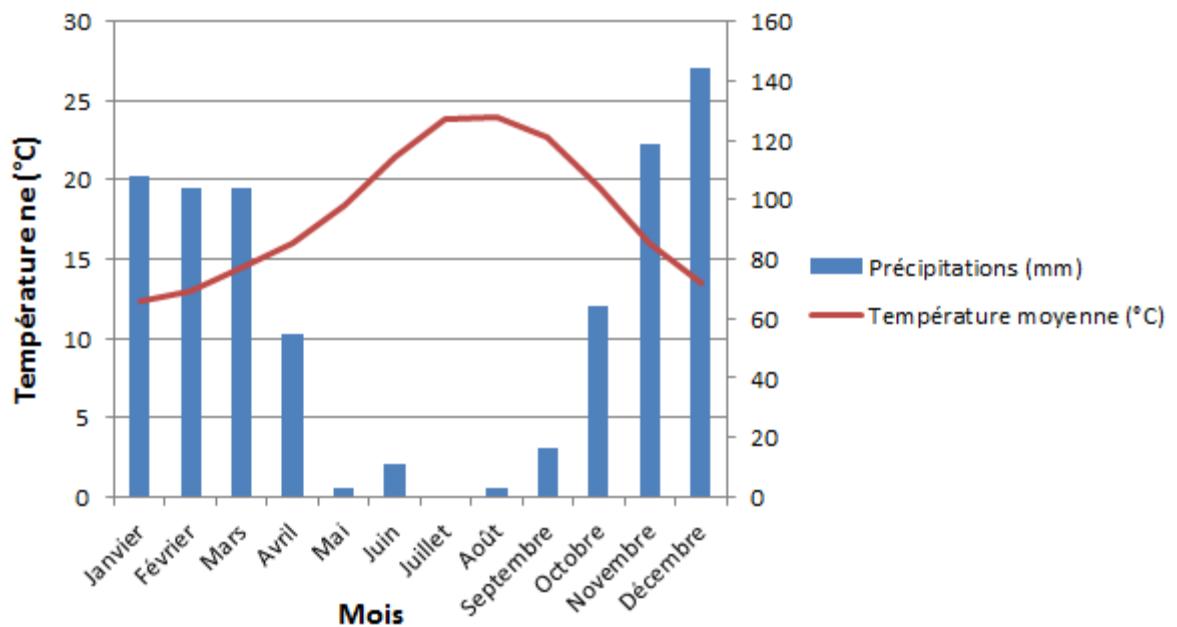
# Chapitre 1 : Généralités

**Exemple 2 :** Traitement de données avec un tableur (Excel)

**Table climatique générale de la ville de Tanger**

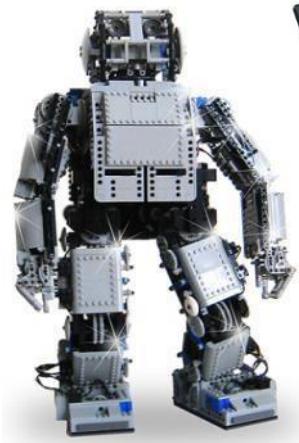
Mois	Précipitations (mm)	Température minimale (°C)	Température maximale (°C)	Température moyenne (°C)
Janvier	108	8,7		
Février	104	8,9		
Mars	104	10,7		
Avril	55	12,2		
Mai	3	13,9		
Juin	11	16,9		
Juillet	0	18,8		
Août	3	19		
Septembre	16	17,7		
Octobre	64	15,4		
Novembre	119	11,8		
Décembre	144	9,7		

**Diagramme climatique général de la ville de Tanger**



# Chapitre 1 : Généralités

**Un système informatique** est un ensemble composé de deux parties : matérielle (*hardware*) et logicielle (*software*)



Robot



Tablette



Appareil Photo Numérique



Ordinateur Portable



Réfrigérateur Intelligent



Smart TV

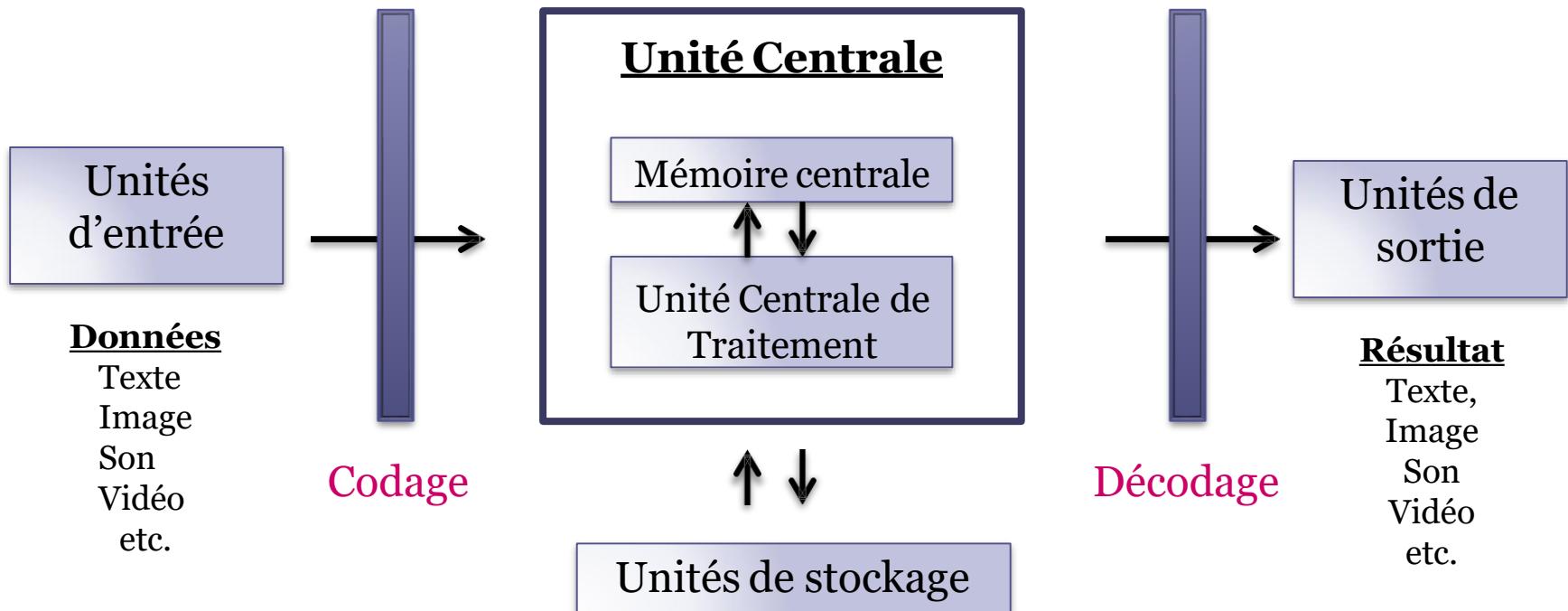


Smartphone

# Chapitre 1 : Généralités

## Schéma fonctionnel d'un système informatique

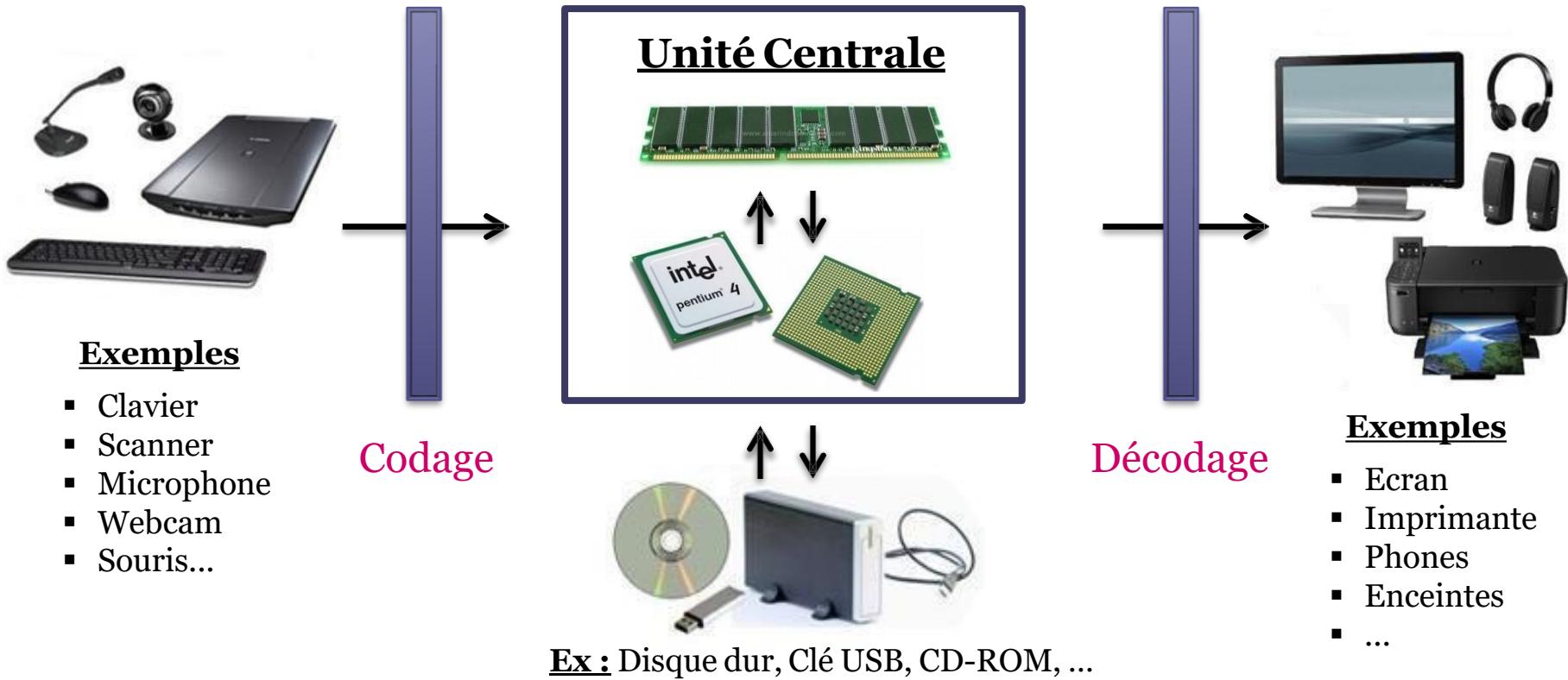
### Traitement des informations



# Chapitre 1 : Généralités

## Schéma fonctionnel d'un ordinateur

### Traitement des informations



# Chapitre 1 : Généralités

## Codage de l'information

C'est la transformation des informations vers des symboles {a, b, c, d, ..., z }, {1, 2, 3, ..... , 9}, {I, II, III, IV, V, ...} pour pouvoir la traiter, stocker, transmettre, etc.

En informatique, l'information est codée sous forme des **chiffres binaires {0,1}** appelée aussi **la base binaire**.

Chaque élément s'appelle **un bit (binary digit)**.

# Chapitre 1 : Généralités

## Codage de l'information

Avec **1 bit**, on peut représenter  **$2=2^1$**  informations binaires : **0 et 1.**

Avec **2 bits**, on peut représenter  **$4=2^2$**  informations binaires : **00, 01, 10, 11.**

...

Avec **8 bits**, on peut représenter  **$256=2^8$**  informations binaires.



Avec **n** bits, on peut représenter  **$2^n$**  informations binaires.

# Chapitre 1 : Généralités

## Codage de l'information

L'information traitée par ordinateur est constituée par un groupe de **8 bits** appelé **un octet**.

**1 Octet = 8 bits**

**1 KiloOctets (Ko)= $10^3$  Octets**

**1 MegaOctet (Mo)= $10^6$  Ko**

**1 GigaOctet (Go)= $10^9$  Mo**

**1 TeraOctet (To)= $10^{12}$  Go**

**1 PetaOctet (Po)= $10^15$  To**

**1 ExaOctet (Eo)= $10^18$  Po**

**1 ZettaOctet (Zo)= $10^{21}$  Eo**

**1 YottaOctet (Yo)= $10^{24}$  Zo**

# Chapitre 1 : Généralités

## Systèmes de numération

- **Système Binaire**
- **Système Octal**
- **Système Décimal**
- **Système Hexadécimal**

# Chapitre 1 : Généralités

## Systèmes de numération

### **Système Binaire**

- Un système de **base 2**
- Utilisation des chiffres : **B={0, 1}**

**Exemples :** **10110, 1101101, 1001, 11**

# Chapitre 1 : Généralités

## Systèmes de numération

### Système Octal

- Un système de **base 8**
- Utilisation des chiffres : **B={0, 1, 2, 3, 4, 5, 6, 7}**

Exemples : **745, 673, 71, 112**

# Chapitre 1 : Généralités

## Systèmes de numération

### **Système Décimal**

- Un système de **base 10**
- Utilisation des chiffres : **B={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}**

**Exemples:** 12435, 5674, 12, 189

# Chapitre 1 : Généralités

## Systèmes de numération

### Système Hexadécimal

- Un système de **base 16**
  - Utilisation des chiffres :
- $$\mathbf{B}=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

Exemples : **5AF, 456B, AE19, 127F**

# Chapitre 1 : Généralités

## Exercice 1

Soit **X** un nombre entier ayant la valeur **57** dans la base décimale, trouver l'équivalent de **X** dans les bases suivantes :

- *Binaire*
- *Octale*
- *Hexadécimale*

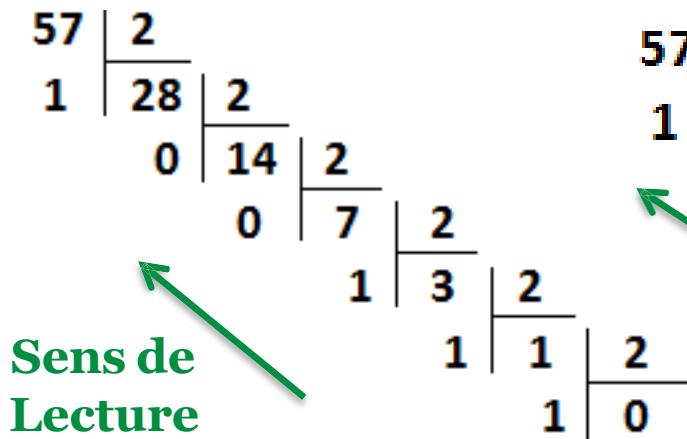
# Chapitre 1 : Généralités

## Exercice 1 - Solution

$$(57)_{10} = (?)_2$$

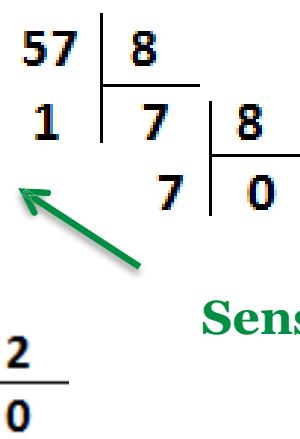
$$(57)_{10} = (?)_8$$

$$(57)_{10} = (?)_{16}$$



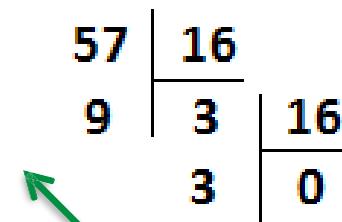
Sens de Lecture

$$(57)_{10} = (111001)_2$$



Sens de Lecture

$$(57)_{10} = (71)_8$$



$$(57)_{10} = (39)_{16}$$

# Chapitre 1 : Généralités

## Exercice 2

Trouver l'équivalent des chiffres suivants dans la base décimale

- $(10110)_2$
- $(745)_8$
- $(5AF)_{16}$

# Chapitre 1 : Généralités

## Exercice 2 - Solution

$$\begin{aligned}(10110)_2 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= (22)_{10}\end{aligned}$$

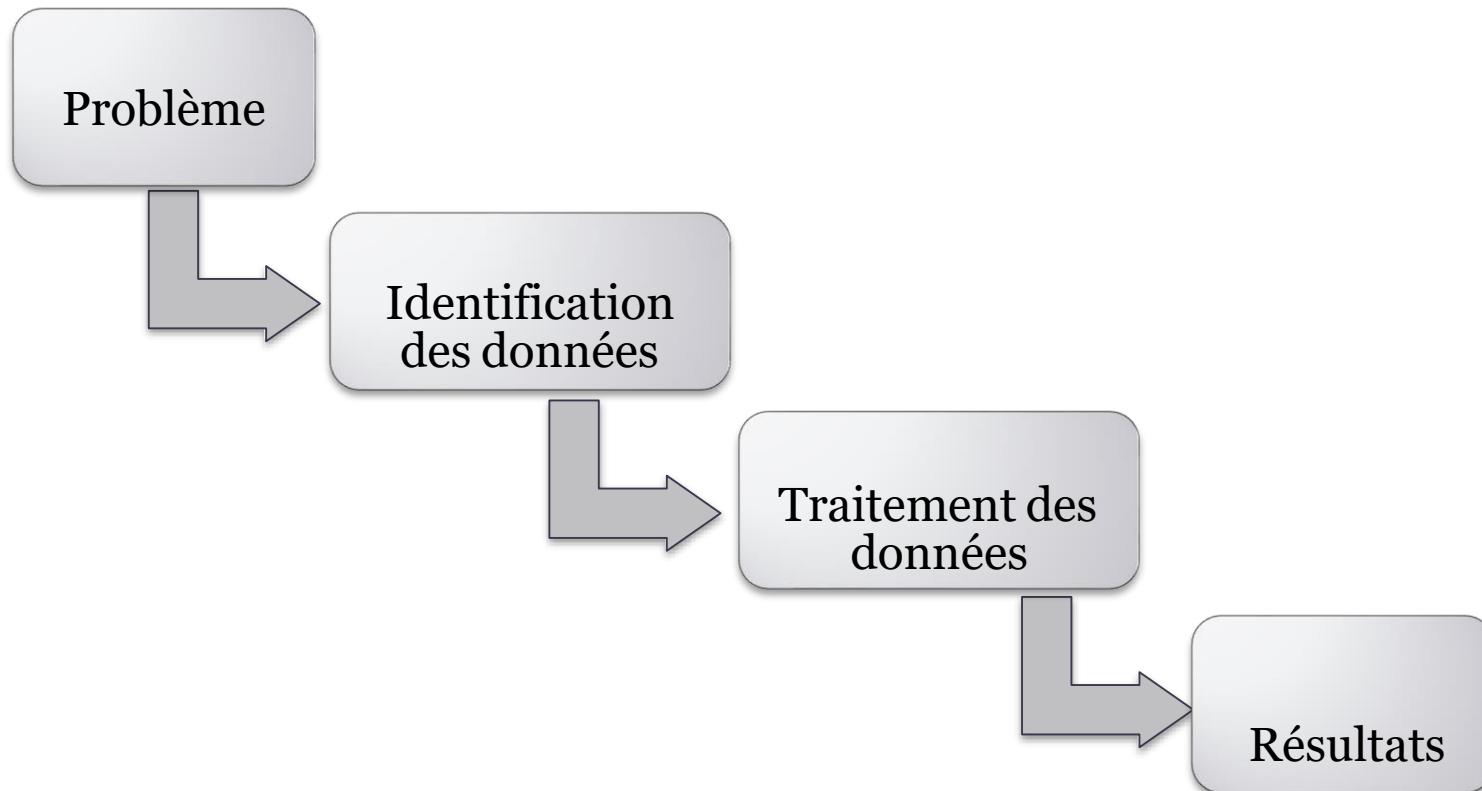
$$\begin{aligned}(745)_8 &= 7 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 \\ &= (485)_{10}\end{aligned}$$

$$\begin{aligned}(5AF)_{16} &= 5 \times 16^2 + A \times 16^1 + F \times 16^0 \\ &= (1455)_{10}\end{aligned}$$

# Chapitre 2 : Les bases de l'algorithmique

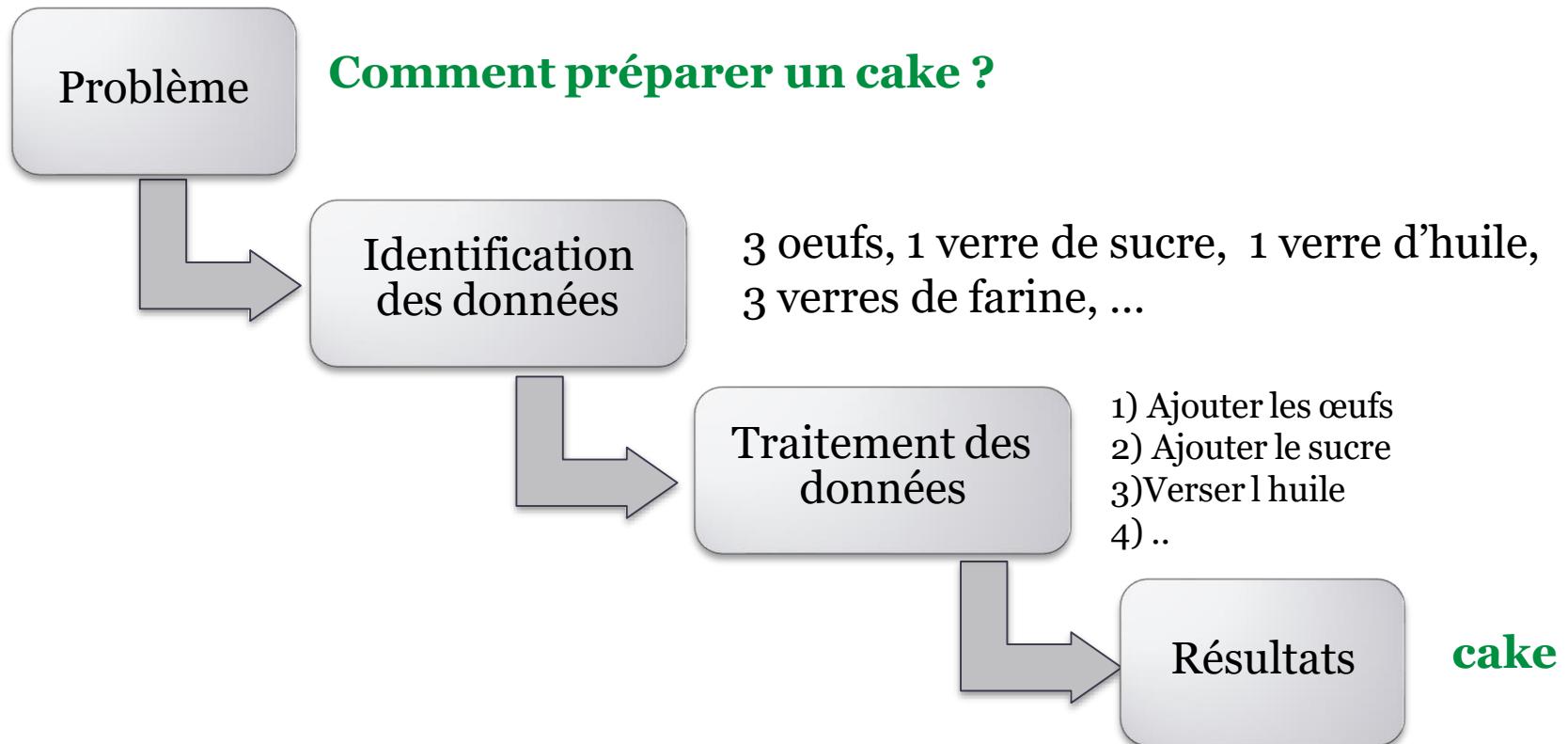
# Chapitre 2 : Les bases de l'algorithmique

## Schéma de résolution d'un problème



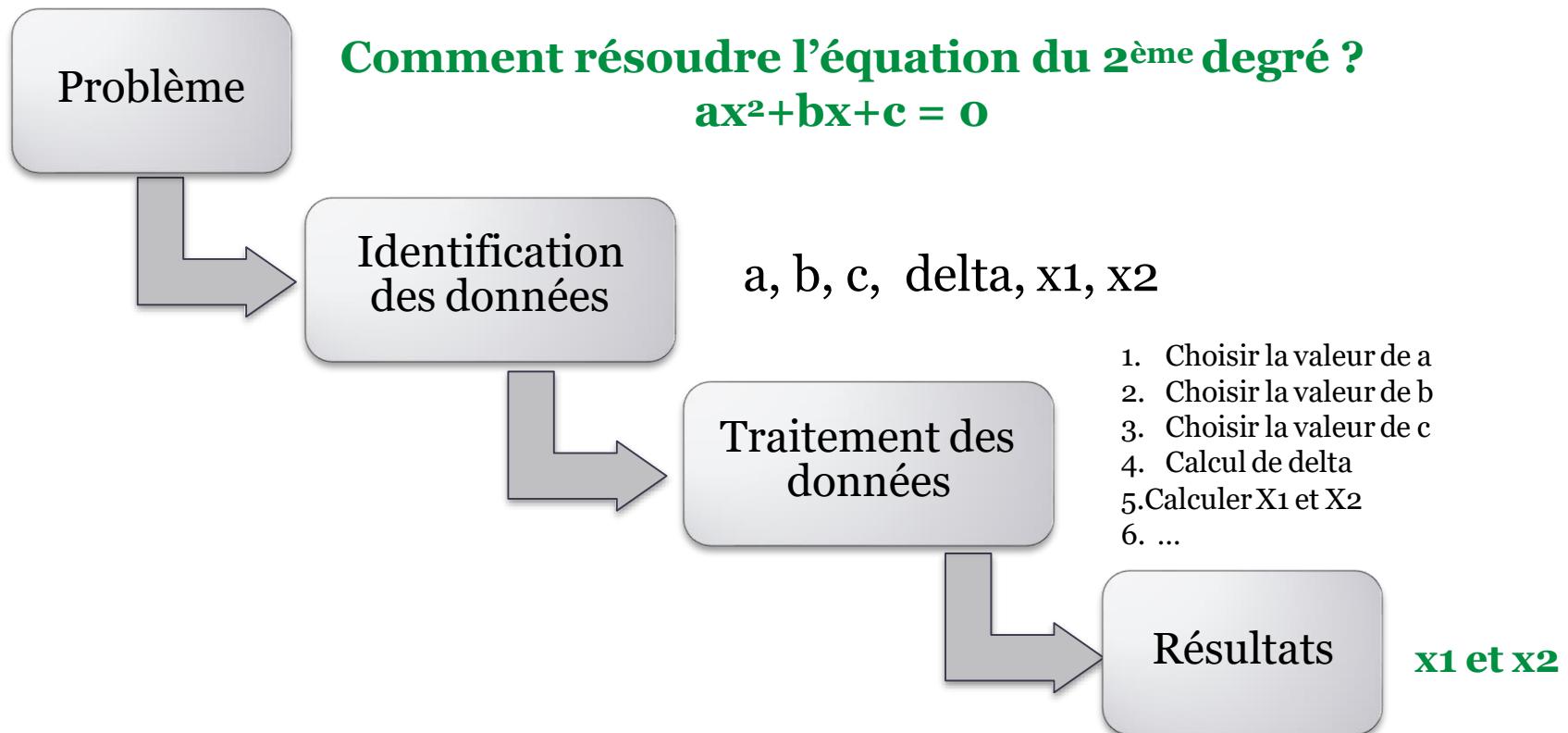
# Chapitre 2 : Les bases de l'algorithmique

## Schéma de résolution d'un problème – Exemple 1



# Chapitre 2 : Les bases de l'algorithmique

## Schéma de résolution d'un problème – Exemple 2



# Chapitre 2 : Les bases de l'algorithmique

## **Notion d'algorithme**

Un algorithme est **une suite d'actions** ou **instructions** appliquées sur **des données** dans **un ordre** bien déterminé pour **résoudre un problème** ou **atteindre un but**.

## **Exercice d'application**

Ecrire un algorithme qui permet de calculer la surface d'un disque.

# Chapitre 2 : Les bases de l'algorithmique

## Structure générale d'algorithme

**Algorithme** Nom\_de\_l'algorithme ;

    Déclarations des données

**Début**

        Instruction 1;

        Instruction 2;

        Instruction 3;

        ....

        Instruction N ;

**Fin**

# Chapitre 2 : Les bases de l'algorithmique

## Structure générale d'algorithme – Exemple 1

**Algorithme** surface\_disque ;

**Variables** R, S : réel ;

**Constante** Pi : réel = 3,141559 ;

**Début**

S  $\leftarrow$  0;

**Ecrire** ('Choisir une valeur pour le rayon :');

**Lire** (R);

S  $\leftarrow$  Pi\*R^2;

**Ecrire** ('La surface du disque est : ') ;

**Ecrire** (S) ;

**Fin**

# Chapitre 2 : Les bases de l'algorithmique

## Notion d'une donnée

- C'est **un emplacement mémoire** dans lequel on peut mettre une valeur.
- Une donnée est caractérisée par **un identificateur, une valeur, un type et une nature.**

0
clavier
a
-1,6
123

*Mémoire de  
l'ordinateur*

## Chapitre 2 : Les bases de l'algorithmique

### **Identificateur d'une donnée**

L'**identificateur** d'une donnée peut être formé par des **lettres** et des **chiffres** et des **lignes de soulignement** ( \_ ) dont le 1<sup>er</sup> caractère est obligatoirement une lettre.

**Ex:** **S, R, Pi, N\_1, etc.**

### **Valeur d'une donnée**

C'est une valeur prise par la donnée au cours de l'exécution de l'algorithme.

# Chapitre 2 : Les bases de l'algorithmique

## **Types d'une donnée**

Il représente le **type de l'ensemble des valeurs** que peut prendre une donnée. Elle peut être de type **Numérique**, **Alphanumérique** ou **Logique**.

	Type de données	Exemples
<b>Numérique</b>	<b>Entier</b>	X=1
	<b>Reel</b>	X=2,2
<b>Alphanumérique</b>	<b>Caractere</b>	X='K'
	<b>Chaine</b>	X='Bonjour'
<b>Logique</b>	<b>Boolean</b>	X=Vrai ou Faux

# Chapitre 2 : Les bases de l'algorithmique

## Nature d'une donnée

Une donnée peut être de nature :

**Constante** : lorsque il garde la même valeur pour un algorithme

**Variable** : lorsque sa valeur est susceptible de varier dans un algorithme.

## Exemple

**Pi** est une constante

**R** et **S** sont des variables

# Chapitre 2 : Les bases de l'algorithmique

## Comment déclarer une donnée ? (1/2)

**Variable** Identificateur\_donnée : Type\_donnée ;

**Variable** Identificateur\_donnée : Type\_donnée = Valeur initiale ;

\*\*\*

**Constante** Identificateur\_Constante : Type de la constante = valeur de la constante;

## Exemple

**Variable** N1 : entier ;

**Variable** N2 : réel= 2,2 ;

**Constante** Pi : réel = 3,141559 ;

# Chapitre 2 : Les bases de l'algorithmique

## Comment déclarer une donnée ? (2/2)

**Variables** Iden1, Iden2, Iden3 = Valeur initiale, ... : Type de données ; //  
*les données doivent avoir même type.*

**Variables** Iden1 : Type de données ;  
Iden2 : Type de données ;  
etc. // pour le cas des données ayant des types différents

## Exemple

**Variables** N1, N2, N3=10 : entier ;  
**Variables** X1 : reel ;  
X2 : caractere ;

# Chapitre 2 : Les bases de l'algorithmique

## ***Instruction de lecture des variables : Ecrire***

C'est l'**action** qui permet à l'algorithme d'afficher des **messages** ou les **valeurs des données** à l'utilisateur.

### ***Syntaxe générale***

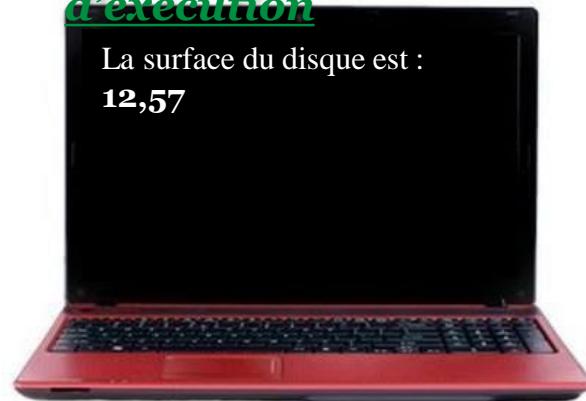
```
Ecrire (identificateur_donnee) ;  
Ecrire (iden_donnee1, iden_donnee2, ...) ;  
Ecrire ('message') ;
```

### ***Résultat***

### ***d'exécution***

La surface du disque est :

**12,57**



### ***Exemple***

```
Ecrire ('La surface du disque est : ') ;  
Ecrire (S) ;
```

# Chapitre 2 : Les bases de l'algorithmique

## Instruction de lecture des variables : Lire

C'est l'**action** qui permet à l'utilisateur de **fournir à l'algorithme** les valeurs des variables.

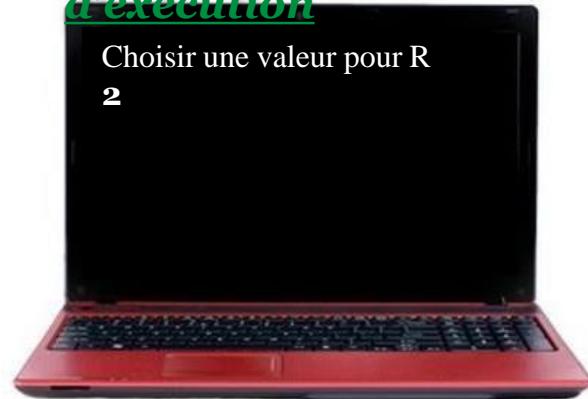
### Syntaxe générale

```
Lire (iden_var) ;  
***
```

```
Lire (iden_var1, iden_var2, ...,) ;
```

### Résultat d'exécution

Choisir une valeur pour R  
**2**



### Exemple

```
Ecrire ('Choisir une valeur pour R');  
Lire (R);
```

# Chapitre 2 : Les bases de l'algorithmique

## ***Instruction d'affectation : ( $\leftarrow$ )***

C'est **l'action** qui permet d'attribuer à une variable **une valeur simple** ou bien **résultante d'une expression arithmétique**.

### **Syntaxe générale**

**Identificateur\_variable  $\leftarrow$  valeur simple ;**

**Identificateur\_variable  $\leftarrow$  expression arithmétique ;**

### **Exemple**

`S  $\leftarrow$  0 ;`

`S  $\leftarrow$  Pi * R2 ;`

### ***Remarque***

La valeur affectée doit être compatible avec le type de la variable destinataire.

## Chapitre 2 : Les bases de l'algorithmique

### Exercices : Série N1

# Chapitre 3 : Les structures alternatives

# Chapitre 3 : Les structures alternatives

## **Les opérateurs & Les expressions**

- Un opérateur est un **signe** qui relie deux valeurs pour produire un **résultat**.
- Il peut être de type **numérique, alphanumérique, de comparaison** ou **logique**.

# Chapitre 3 : Les structures alternatives

## Les opérateurs numériques ou arithmétiques

Type des données	Opérateur	Action
Entier et Réel	+	Addition
	-	Soustraction
	*	Multiplication
	/	Division
	^	Puissance
	<b>mod</b>	Reste de la division

# Chapitre 3 : Les structures alternatives

## **Les opérateurs numériques ou arithmétiques :** exemple

Expression	Instruction	Variable	Valeur
$5+2-1$	$A \leftarrow 5+2-1$	A	
$5*3$	$B \leftarrow 5*3$	B	
$(-1*6)+8^2$	$C \leftarrow (-1*6) + 8^2$	C	
$B+5-C$	$A \leftarrow B+5-C$	A	
$A/4$	$C \leftarrow A/4$	C	
$A \bmod B$	$B \leftarrow A \bmod B$	B	

# Chapitre 3 : Les structures alternatives

## **Les opérateurs numériques ou arithmétiques :** exemple

Expression	Instruction	Variable	Valeur
$5+2-1$	$A \leftarrow 5+2-1$	A	6
$5*3$	$B \leftarrow 5*3$	B	15
$(-1*6)+8^2$	$C \leftarrow (-1*6) + 8^2$	C	58
$B+5-C$	$A \leftarrow B+5-C$	A	-38
$A/4$	$C \leftarrow A/4$	C	-9,5
$A \bmod B$	$B \leftarrow A \bmod B$	B	8

# Chapitre 3 : Les structures alternatives

## Les opérateurs alphanumériques

Ce sont les opérateurs **&** ou **+** appliqués sur des données de type alphanumérique (**caractère** ou **chaine**)

### Exemple

Expression	Instruction	Variable	Valeur
‘Bien’ & ‘venue’	S1 $\leftarrow$ ‘Bien’ & ‘venue’	S1	
‘ au’ + ‘ monde ’	S2 $\leftarrow$ ‘ au’ + ‘ monde ’	S2	
S2 + ‘informatique’	S3 $\leftarrow$ S2 + ‘ informatique’	S3	
S1 & S3	S1 $\leftarrow$ S1 & S3	S1	

# Chapitre 3 : Les structures alternatives

## Les opérateurs alphanumériques

Ce sont les opérateurs **&** ou **+** appliqués sur des données de type alphanumérique (**caractère** ou **chaine**)

### Exemple

Expression	Instruction	Variable	Valeur
‘Bien’ & ‘venue’	S1 $\leftarrow$ ‘Bien’ & ‘venue’	S1	Bienvenue
‘ au ’ + ‘ monde ’	S2 $\leftarrow$ ‘ au ’ + ‘ monde ’	S2	au monde
S2 + ‘informatique’	S3 $\leftarrow$ S2 + ‘ informatique ’	S3	au monde informatique
S1 & S3	S1 $\leftarrow$ S1 & S3	S1	Bienvenue au monde informatique

# Chapitre 3 : Les structures alternatives

## Les opérateurs de comparaison

<	inférieur à
$\leq$	inférieur ou égal à
>	supérieur à

$\geq$	<b>supérieur ou égal à</b>
=	égal à
$\neq$	différent

## Les opérateurs logiques

Il s'agit des opérateurs **ET**, **OU**, **XOR** et **NON**, appliqués sur des données de type **booléen** ou sur des **expressions logiques simples**.

# Chapitre 3 : Les structures alternatives

## Expression logique simple

- C'est une **comparaison** de deux valeurs de même type en utilisant **un opérateur de comparaison**.
- La valeur d'une expression logique est de type **booléen** (vrai ou faux)

## Exemple

A < 0

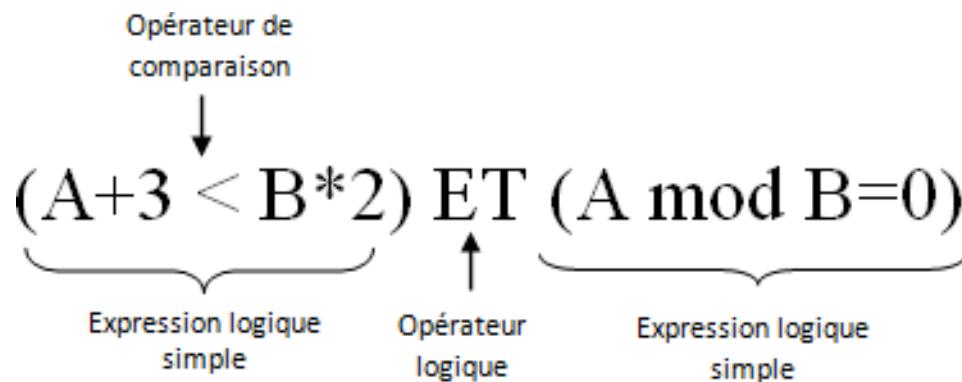
A = B

# Chapitre 3 : Les structures alternatives

## Expression logique complexe

- C'est une **combinaison** entre deux expressions logiques simples en utilisant **un opérateur logique**.
- La valeur d'une expression logique complexe est de type **booléen** (vrai ou faux)

### Exemple



# Chapitre 3 : Les structures alternatives

## Exercice d'application

A, B, C et D sont des variables de type logique et X variable de type numérique.

Soient les instructions suivantes :

```
...
lire(X) ;
A ← X < 2 ;
B ← X > 12 ;
C ← A ET B ;
D ← A OU B ;
D ← NON (A OU B) ;
```

...

Quels seront les valeurs des variables A, B, C et D si X=13 ?

# Chapitre 3 : Les structures alternatives

## *Rappel*

Valeur de A	Valeur de B	Valeur de (A ET B)	Valeur de (A OU B)	Valeur de (A XOR B)	Valeur de Non A
Vrai	Vrai	Vrai	Vrai	Faux	Faux
Vrai	Faux	Faux	Vrai	Vrai	Faux
Faux	Vrai	Faux	Vrai	Vrai	Vrai
Faux	Faux	Faux	Faux	Faux	Vrai

## *Solution*

X	A	B	C	D
13	Faux	Vrai	Faux	Faux

# Chapitre 3 : Les structures alternatives

## **La structure alternative**

C'est une structure qui permet d'exécuter telles instructions ou telles autres en fonction de réponses à des conditions.

Il existe 4 types de structures alternatives :

- **Structure alternative simple**
- **Structure alternative complète**
- **Structure alternative imbriquée**
- **Structure à choix multiple**

# Chapitre 3 : Les structures alternatives

## **La structure alternative simple :** syntaxe générale

Si (**expression logique**) alors

    Instruction1;

    Instruction2;

    ...

    Instruction n;

Finsi



Instructions à exécuter  
si l'**expression logique**  
**est Vraie**

# Chapitre 3 : Les structures alternatives

## **La structure alternative simple :** exemple

**Algorithme** Moyenne;

**Variables** CC1, CC2, Moy=0 : réel;

**Début**

**Ecrire** ('Entrer les deux notes');

**Lire** (CC1,CC2);

Moy  $\leftarrow$  (CC1+CC2)/2;

**Ecrire** ('La moyenne est : ', Moy) ;

**Si (Moy>=10) alors**

**Ecrire ('Module validé');**

**Finsi**

**Fin**

# Chapitre 3 : Les structures alternatives

## **La structure alternative complète :** syntaxe générale

**Si (expression logique) alors**

Instruction1;  
Instruction2;  
...  
Instruction n;



Instructions à exécuter  
si l'expression logique  
est **Vraie**

**Sinon**

Instruction1;  
Instruction2;  
...  
Instruction n;



Instructions à exécuter  
si l'expression logique  
est **Fausse**

**Finsi**

# Chapitre 3 : Les structures alternatives

## **La structure alternative complète :** exemple

**Algorithme** Moyenne;

**Variables** CC1, CC2, Moy: réel;

**Début**

**Ecrire** ('Saisir les deux notes');

**Lire** (CC1,CC2);

Moy  $\leftarrow$  (CC1+CC2)/2;

**Ecrire** ('La moyenne est : ', Moy) ;

**Si** (*Moy>=10*) **alors**

*Ecrire ('Module validé');*

**Sinon**

*Ecrire ('Module à rattraper ou non validé');*

**Finsi**

**Fin**

# Chapitre 3 : Les structures alternatives

## **La structure alternative imbriquée :** syntaxe générale

**Si** (*expression logique 1*) **alors**

Instructions à exécuter si l'**expression logique 1** est vraie

**Sinon**

**Si** (*expression logique 2*) **alors**

Instructions à exécuter si l'**expression logique 2** est vraie

**Sinon**

Instructions à exécuter si l'**expression logique 2** est fausse

**Finsi**

**Finsi**

# Chapitre 3 : Les structures alternatives

## **La structure alternative imbriquée :** exemple

**Algorithme** Moyenne;

**Variables** CC1, CC2, Moy: réel;

**Début**

**Ecrire** ('saisir les deux notes'); **Lire** (CC1,CC2);

Moy  $\leftarrow$  (CC1+CC2)/2;

**Ecrire** ('La moyenne est : ', Moy) ;

**Si** (Moy $\geq$ 10) **alors**

**Ecrire** ('Module validé');

**Sinon**

**Si** (Moy  $\geq$ 5) **alors**

**Ecrire** ('Module à rattraper');

**Sinon**

**Ecrire** ('Module non validé');

**Finsi**

**Finsi**

**Fin**

# Chapitre 3 : Les structures alternatives

## **La structure à choix multiple :** syntaxe générale

**Selon *Variable* faire**

*Valeur\_1* : Instructions à exécuter si *Variable* = *Valeur\_1*;

*Valeur\_2* : Instructions à exécuter si *Variable* = *Valeur\_2*;

...

*Valeur\_N* : Instructions à exécuter si *Variable* = *Valeur\_N*;

**Autrement** : Instructions à exécuter si *Variable* n'appartient pas à  
la liste des valeurs {*Valeur\_1*, *Valeur\_2*, ... ,  
*Valeur\_N*}

**Finselon**

# Chapitre 3 : Les structures alternatives

## **La structure à choix multiple :** exemple

**Selon Ch faire**

‘A’ : *Ecrire(‘Addition ’);*

‘S’ : *Ecrire (‘Soustraction’);*

‘M’ : *Ecrire (‘Multiplication’);*

‘D’ : *Ecrire (‘Division’);*

**Autrement** : *Ecrire (‘choix introuvable’);*

**Finselon**

# Chapitre 3 : Les structures alternatives

Exercices : Série N2

# Chapitre 4 : Les structures de répétition

# Chapitre 4 : Les structures de répétition

## Les boucles

Une boucle est un ensemble d'instructions qui se **répètent** selon une **condition** bien déterminée.

Trois boucles sont utilisées :

- La boucle **POUR ... FAIRE**
- La boucle **REPETER ... JUSQU'A**
- La boucle **TANTQUE ... FINTANTQUE**

# Chapitre 4 : Les structures de répétition

## **La boucle : Pour ... Faire**

### Syntaxe générale

**Pour** compteur **allant de** valeur initiale **à** valeur finale **Faire**

Instruction 1 ;  
Instruction 2 ;  
...  
Instruction N ;



Instructions à  
exécuter **N** fois

**FinPour**

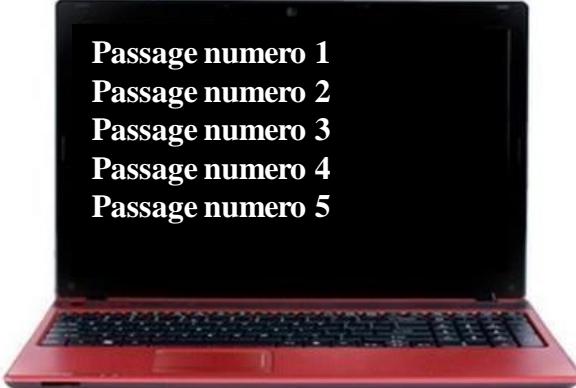
# Chapitre 4 : Les structures de répétition

## **La boucle : Pour ... Faire**

### **Exemple**

```
Algorithme Exemple_1 ;  
Variable i : Entier ;  
Début  
Pour i allant de 1 à 5 Faire  
    Ecrire ('Passage numéro : ', i);  
Finpour  
Fin
```

### **Résultat**



```
Passage numero 1  
Passage numero 2  
Passage numero 3  
Passage numero 4  
Passage numero 5
```

# Chapitre 4 : Les structures de répétition

## **La boucle : Répéter ... Jusqu'à**

### Syntaxe générale

#### Répéter

Instruction 1 ;  
Instruction 2 ;  
...  
Instruction N ;



Instructions à exécuter  
tant que l'expression  
logique est **Fausse**

#### Jusqu'à Expression Logique

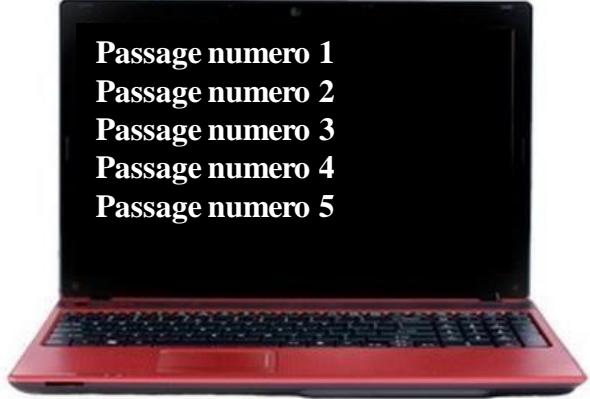
# Chapitre 4 : Les structures de répétition

## **La boucle : Répéter ... Jusqu'à**

### **Exemple**

```
Algorithme Exemple_2 ;  
Variable i : Entier ;  
Début  
    i ← 1 ;  
    Répéter  
        Ecrire ('Passage numéro : ', i);  
        i ← i+1;  
    Jusqu'à i > 5  
Fin
```

### **Résultat**



```
Passage numero 1  
Passage numero 2  
Passage numero 3  
Passage numero 4  
Passage numero 5
```

# Chapitre 4 : Les structures de répétition

## **La boucle : Répéter ... Jusqu'à**

### **Exercice d'application**

Ecrire un algorithme qui demande à l'utilisateur d'introduire un ensemble des valeurs numériques puis calculer et afficher leur carré.

L'algorithme s'arrête une fois l'utilisateur introduit la valeur 0.

# Chapitre 4 : Les structures de répétition

## **La boucle : Répéter ... Jusqu'à**

### **Solution**

**Algorithme** Exercice\_1;

**Variables** a, C : **réel** ;

**Début**

**Répéter**

Ecrire (' Entrer une valeur ');

Lire (a) ;

C $\leftarrow$  a\*a;

Ecrire ('le carré de la valeur saisie est ', C ) ;

**Jusqu'à a = 0**

**Fin**

# Chapitre 4 : Les structures de répétition

## **La boucle : Tantque ... FinTantque**

### Syntaxe générale

#### **Tantque ( Expression Logique )**

Instruction 1 ;  
Instruction 2 ;  
...  
Instruction N ;



Instructions à exécuter  
tant que l'expression  
logique est **Vraie**

#### **FinTantque**

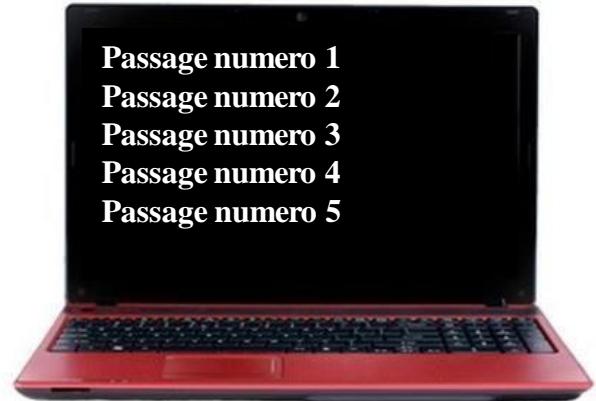
# Chapitre 4 : Les structures de répétition

## **La boucle : Tantque ... FinTanque**

### **Exemple**

```
Algorithme Exemple_3 ;  
Variable i : Entier ;  
Début  
i← 1 ;  
Tantque (i<=5)  
    Ecrire ('Passage numéro : ', i);  
    i← i+1;  
FinTantque  
Fin
```

### **Résultat**



# Chapitre 4 : Les structures de répétition

## ***La boucle : Tantque ... FinTantque***

### **Exercice d'application**

Ecrire un algorithme qui demande à l'utilisateur d'introduire un ensemble des chiffres supérieurs ou égales à 100.

L'algorithme s'arrête une fois l'utilisateur introduit une valeur inférieure strictement à 100.

# Chapitre 4 : Les structures de répétition

## **La boucle : Tantque ... FinTantque**

### **Solution**

**Algorithme** Exercice\_2;

**Variable** N : Entier ;

**Debut**

**Ecrire** (‘Donnez un nombre supérieur ou égal à 100’);

**Lire** (N) ;

**Tantque** (N>=100)

**Ecrire** (‘Donnez un nombre supérieur ou égal à 100’);

**Lire** (N) ;

**FinTantque**

**Ecrire** (‘ok merci’);

**Fin**

## Chapitre 4 : Les structures de répétition

Exercices : Série N3

# Chapitre 5 : Les tableaux

# Chapitre 5 : Les tableaux

## Définition

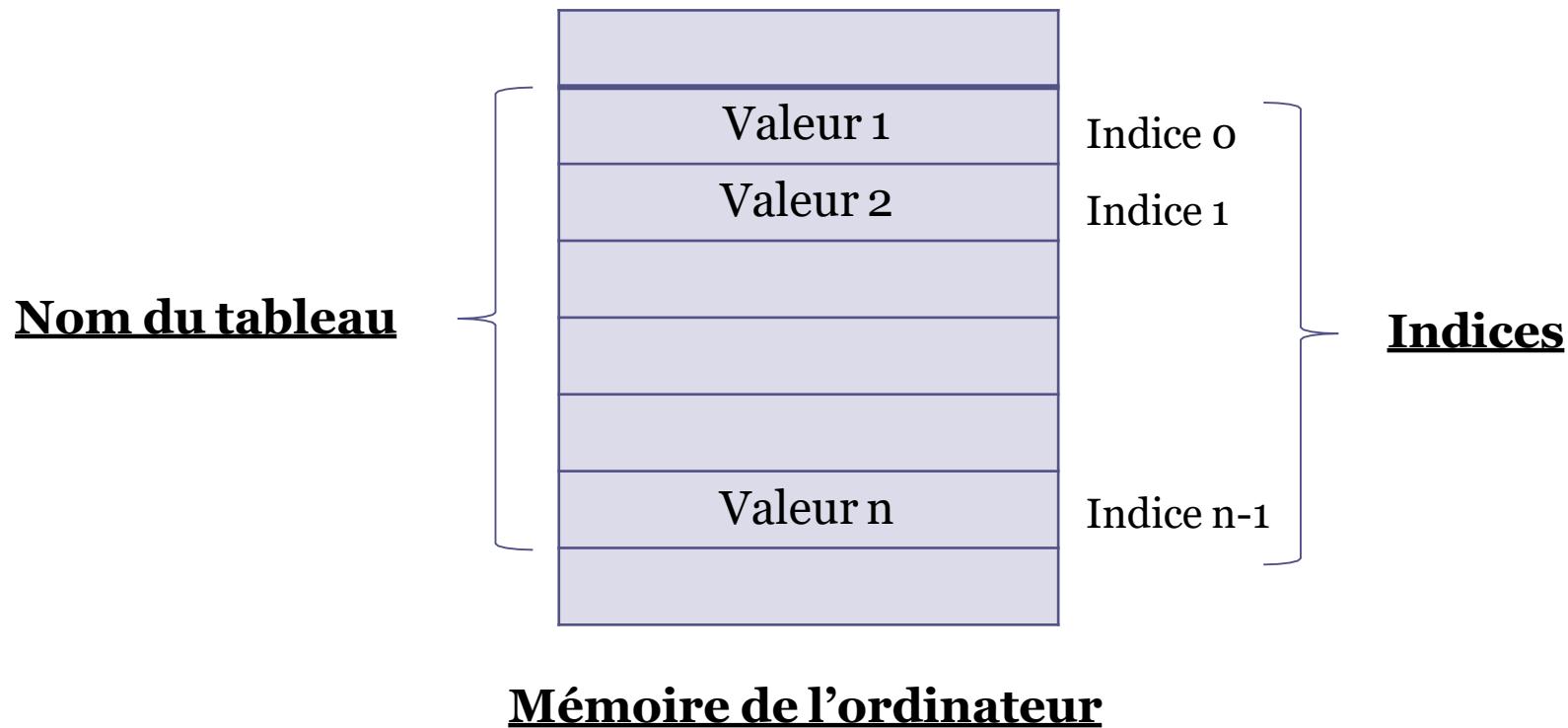
Un **tableau** est une **variable composée** d'un ensemble de données de **même type**, stockées de **manière contiguë** en mémoire (les unes à la suite des autres).

On distingue deux types des tableaux :

- Tableau à une seule dimension
- Tableau à deux dimensions

# Chapitre 5 : Les tableaux

## Les tableaux avec une seule dimension



# Chapitre 5 : Les tableaux

## **Comment déclarer un tableau à une seule dimension**

**Variables Nom\_Tableau (Nombre des éléments) : Type des éléments ;**

### **Exemple**

**Variable Notes (10) : réel ;**

# Chapitre 5 : Les tableaux

## Comment remplir un tableau à une seule dimension

**Algorithme** exemple\_1 ;

**Variable** X (4) : entier ;

**Début**

X (0)  $\leftarrow$  12 ;

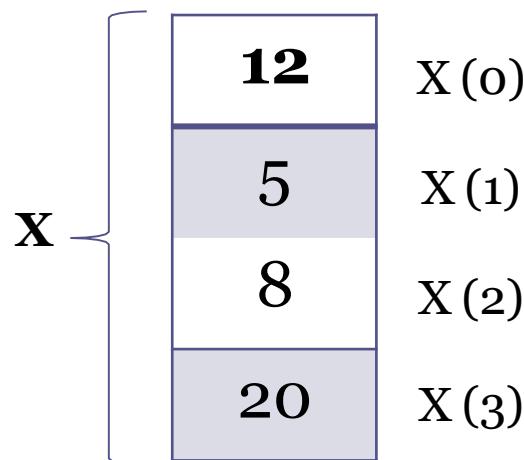
X (1)  $\leftarrow$  5 ;

X (2)  $\leftarrow$  8 ;

X (3)  $\leftarrow$  20 ;

**Fin**

### Au niveau de la Mémoire



Ce pseudo code permet de remplir le tableau **X** par les valeurs : **12, 5, 8 et 20**

# Chapitre 5 : Les tableaux

## Comment remplir un tableau à une seule dimension

**Algorithm**me exemple\_2 ;

**Variable** X (4) : entier ;

**Début**

X (0)  $\leftarrow$  1 ;

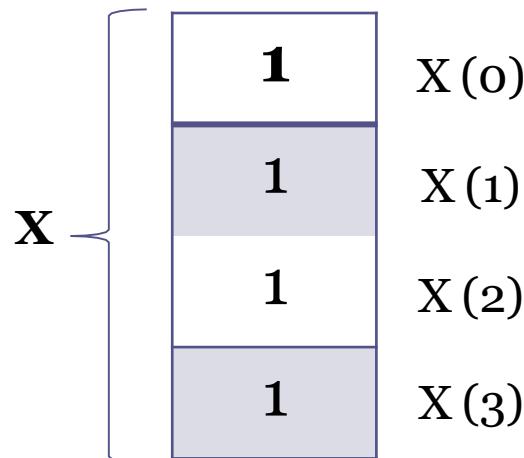
X (1)  $\leftarrow$  1 ;

X (2)  $\leftarrow$  1 ;

X (3)  $\leftarrow$  1 ;

**Fin**

### Au niveau de la Mémoire



Ce pseudo code permet de remplir toutes les cases du tableau **X** par une même valeur qui est **1**.

# Chapitre 5 : Les tableaux

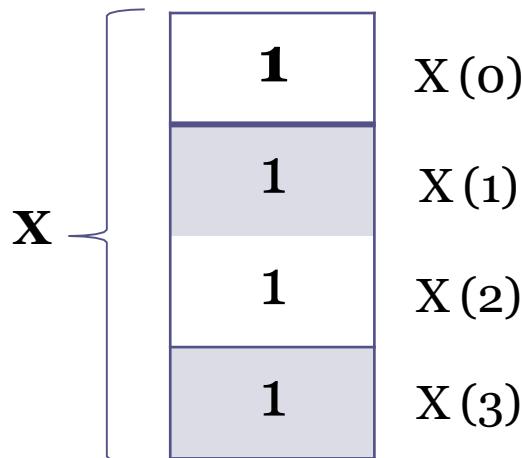
## Comment remplir un tableau à une seule dimension

```

Algorithme exemple_3 ;
Variable i, X (4) : entier ;
Début
  Pour i allant de 0 à 3 faire
    X (i)  $\leftarrow$  1 ;
  Fipour
  Fin

```

### Au niveau de la Mémoire



Ce pseudo code permet de remplir toutes les cases du tableau **X** par une même valeur qui est **1** en utilisant la boucle **Pour**.

# Chapitre 5 : Les tableaux

## Comment remplir un tableau à une seule dimension

**Algorithm**e exemple\_4 ;

**Variable** X (4) : entier ;

**Début**

**Ecrire** (' Entrer les éléments du tableau' ) ;

**Lire** (X (0));

**Lire** (X (1));

**Lire** (X (2));

**Lire** (X(3));

**Fin**

Ce pseudo code permet de lire les valeurs du tableau **X** c'est-à-dire l'utilisateur choisit les valeurs au moment de l'exécution.

# Chapitre 5 : Les tableaux

## Comment remplir un tableau à une seule dimension

```
Algorithm exemple_5 ;  
Variable X (4), i : entier ;  
Début  
Ecrire ('Entrer les éléments du tableau') ;  
Pour i allant de 0 à 3 faire  
    Ecrire (' Entrer l'élément' , i) ;  
    Lire (X (i));  
Finpour  
Fin
```

Ce pseudo code permet de lire les valeurs du tableau **X** c'est-à-dire l'utilisateur choisit les valeurs au moment de l'exécution.

# Chapitre 5 : Les tableaux

## Comment afficher un tableau à une seule dimension

```
Algorithme exemple_6 ;  
Variable X (4), i : entier ;  
Début  
Pour i allant de 0 à 3 faire  
    X (i)  $\leftarrow$  12 ;  
Finpour  
Ecrire ('les éléments du tableau sont :') ;  
Pour i allant de 0 à 3 faire  
    Ecrire (X(i));  
Finpour  
Fin
```

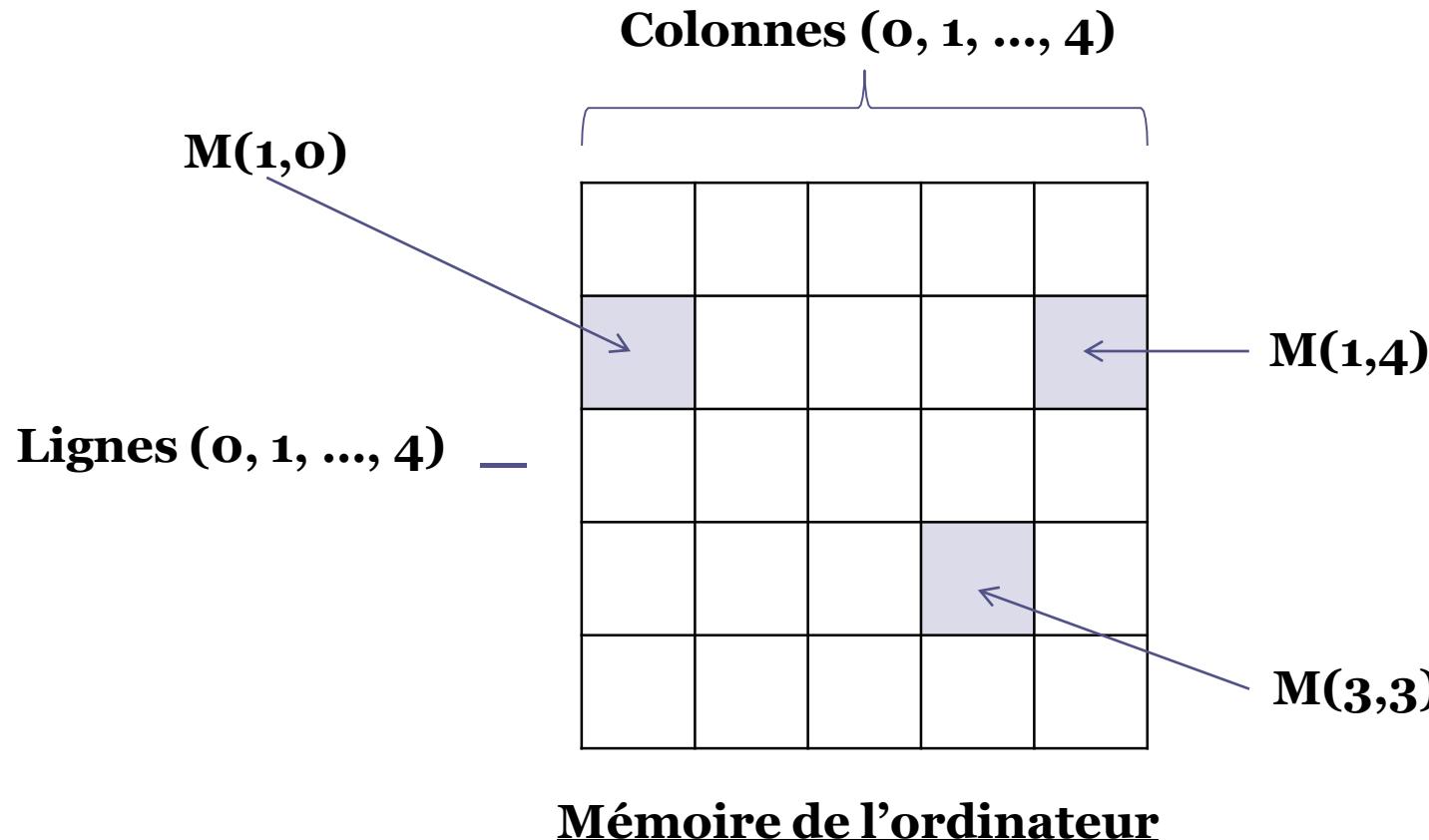
### Résultat



Ce pseudo code permet d'afficher les valeurs du tableau **X** qui sont : **12, 12, 12 et 12**

# Chapitre 5 : Les tableaux

## Les tableaux avec deux dimensions



# Chapitre 5 : Les tableaux

## **Comment déclarer un tableau à deux dimension**

**Variables Nom\_Matrice (Nombre des lignes, Nombre des colonnes) : Type des éléments ;**

### **Exemple**

**Variable Matrice (3, 4) : réel ;**

# Chapitre 5 : Les tableaux

## Comment remplir un tableau à deux dimension

```

Algorithme exemple_1;
Variable M (3, 3), i, j : entier ;
Début
Ecrire ('Remplir la matrice :');
Pour i allant de 0 à 2 faire
    Pour j allant de 0 à 2 faire
        Ecrire (' M (', i, '/', j, ') :');
        Lire (M (i, j));
    Finpour
Finpour
Fin

```

### Résultat

#### d'exécution

Remplir la matrice :

M (0/ 0): 12  
 M (0/ 1): 23  
 M (0 /2) : 33  
 M (1/0): 41  
 M (1/1): 15  
 M (1 / 2) : 6  
 M (2/0) : 27  
 M (2 /1) : 118  
 M (2 / 2) : 9



Ce pseudo code permet de lire les valeurs du tableau **M** c'est-à-dire l'utilisateur choisit les valeurs au moment de l'exécution.

# Chapitre 5 : Les tableaux

## Comment afficher un tableau à deux dimension

...

**Ecrire** (' Les éléments de la matrice : ') ;

**Pour i allant de 0 à 2 faire**

**Pour j allant de 0 à 2 faire**

Ecrire (M (i, j)) ;

**Finpour**

**Finpour**

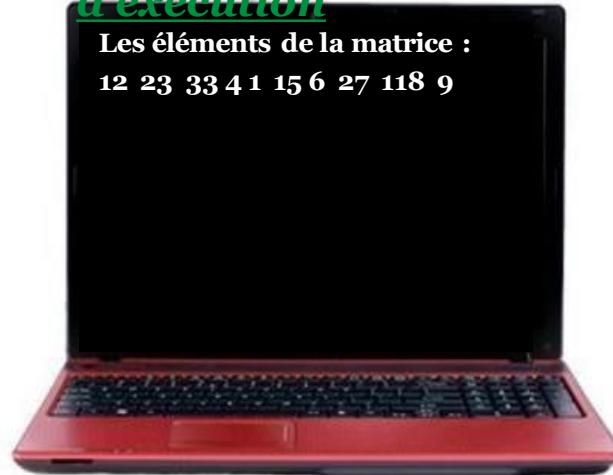
...

### Résultat

### d'exécution

Les éléments de la matrice :

12 23 33 41 15 6 27 118 9



Ce pseudo code permet d'afficher les valeurs du tableau **M** déjà saisies dans l'exemple précédent.

## Chapitre 5 : Les tableaux

Exercices : Série N4

# Chapitre 6 : Programmation I

# 1. Notion de programme

## ● Rappel:

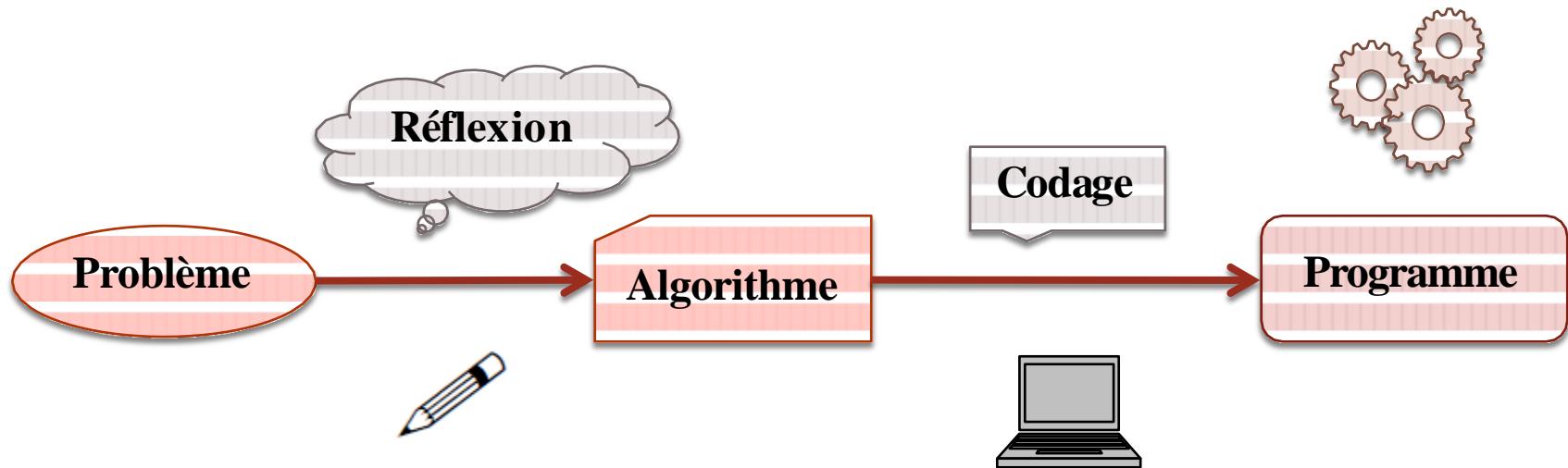
- Un ordinateur est une machine électronique programmable servant au traitement de l'information codée sous forme binaire (0 ou 1).
- Comme l'ordinateur a l'avantage d'exécuter très rapidement et sans erreurs les ordres qu'on lui donne (*les instructions*), il exécute beaucoup de traitements complexes plus vite et plus sûrement qu'un homme.
- Pour donner des ordres à l'ordinateur, il est nécessaire de pouvoir communiquer avec lui.
- Cette communication passe par un **langage de programmation**, dans lequel est écrit **le programme**.

# 1. Notion de programme

- Un **algorithme** représente l'enchaînement des actions (*instructions*) nécessaires pour faire exécuter une tâche à un ordinateur (résoudre un problème).
- Un **algorithme** s'écrit le plus souvent en pseudo-langage de programmation (appelé **langage algorithmique**)
- Un **programme** est un assemblage et un enchaînement d'instructions élémentaires écrit dans un **langage de programmation**, et exécuté par un ordinateur afin de traiter les données d'un problème et renvoyer un ou plusieurs résultats.

# 1. Notion de programme

- Un algorithme n'est donc exécutable directement par aucune machine.
- Mais il a l'avantage d'être traduit facilement dans tous les langages de programmation.



# 1. Notion de programme

- L'**algorithmique**, l'art d'écrire des algorithmes, permet de se focaliser sur la procédure de résolution du problème sans avoir à se soucier des spécificités d'un langage particulier.
- Pour résoudre un problème, il est vivement conseillé de réfléchir d'abord à l'algorithme avant de programmer proprement dit, c'est à dire d'écrire le programme en **langage de programmation**.

## 2. Notion de variables et déclarations

- Les programmes ont pour but de traiter différentes **données** afin de produire des **résultats**.
- Les résultats peuvent eux-mêmes être des **données** pour d'autres **programmes**.



## 2. Notion de variables et déclarations

- Les données d'un programme doivent être récupérées en mémoire centrale, à partir du clavier ou d'un fichier par exemple, pour pouvoir être traitées par le processeur qui exécute le programme.
- Toutes les données d'un programme sont mémorisées en mémoire centrale, dans des sortes de cases que l'on appelle **variables**.

## 2. Notion de variables et déclarations

- Une **variable** peut être représentée par une case mémoire, qui contient la valeur d'une donnée.
- Chaque **variable** possède un nom unique appelé **identificateur** par lequel on peut accéder à son **contenu**.
- **Exemple** : on peut avoir en mémoire une variable **prix** et une variable **quantité** qui contiennent les valeurs **10.2** et **5**.

10.2

prix

5

quantité

## 2. Notion de variables et déclarations

- Attention à ne pas confondre la variable et son contenu
- Une **variable** est un **contenant**, c'est à dire une sorte de boîte, alors que le **contenu** d'une variable est une valeur numérique, alphanumérique ou booléenne, ou de tout autre type.
- Deux variables peuvent avoir la même valeur, mais une variable ne peut pas avoir plusieurs valeurs en même temps (*sauf les types composés*).

## 2. Notion de variables et déclarations

- En revanche, la valeur d'une variable peut varier au cours du programme.
- L'ancienne valeur est tout simplement écrasée et remplacée par la nouvelle.
- Les variables dont la valeur ne change pas au cours de l'exécution du programme sont appelées **variables constantes** ou plus simplement **constantes**.

# 3. Les bases du langage C

- Le C est un langage compilé (par opposition aux langages interprétés).
- Cela signifie qu'un programme C est décrit par un fichier texte, appelé fichier source. Ce fichier n'étant évidemment pas exécutable par le microprocesseur, il faut le traduire en langage machine.
- Cette opération est effectuée par un programme appelé compilateur.
- La compilation se décompose en fait en 4 phases successives :
  - Le traitement par le préprocesseur
  - La compilation
  - L'assemblage
  - L'édition de liens

# 3.1 Jeu de caractères

- 26 lettres de l'alphabet (minuscules, majuscules)
- chiffres 0 à 9
- caractères spéciaux :

!	*	+	\	"	<
#	(	=		{	>
%	)	e	;	]	/
^	-	[	:	,	?
&	_	}	'	.	(espace)

- séquences d'échappement telles :
  - passage à la ligne (\n),
  - tabulation (\t),
  - backspace (\b).

## 3.2 Identificateurs et mots-clés

- **Identificateur** : nom donné aux diverses composantes d'un programme (variables, tableaux, fonctions, ...)
  - Formé de lettres et de chiffres ainsi que du caractère ‘\_’ permettant une plus grande lisibilité.
  - Le 1<sup>er</sup> caractère doit obligatoirement être une lettre ou bien le caractère ‘\_’
  - Peut contenir jusqu'à 31 caractères minuscules et majuscules.
- *Il est d'usage de réserver les identificateurs entièrement en majuscules aux variables du préprocesseur.*

# 3.2 Identificateurs et mots-clés

## ● Exemples :

### ● Identificateurs valides :

● x	y12	somme_1	_temperature
● noms	surface	fin_de_fichier	TABLE

### ● Identificateurs invalides :

● 4eme	commence par un chiffre
● x#y	caractère non autorisé (#)
● no--commande	caractère non autorisé (-)
● taux change	caractère non autorisé (espace)

## 3.2 Identificateurs et mots-clés

- **Mots réservés (mots-clés)** : Un certain nombre de mots, appelés *mots-clefs*, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs.

<b>auto</b>	<b>const</b>	<b>double</b>	<b>float</b>	<b>int</b>	<b>short</b>
<b>break</b>	<b>continue</b>	<b>else</b>	<b>for</b>	<b>long</b>	<b>signed</b>
<b>case</b>	<b>default</b>	<b>enum</b>	<b>goto</b>	<b>register</b>	<b>sizeof</b>
<b>char</b>	<b>do</b>	<b>extern</b>	<b>if</b>	<b>return</b>	<b>static</b>
<b>struct</b>	<b>unsigned</b>	<b>switch</b>	<b>void</b>	<b>typedef</b>	<b>volatile</b>
<b>union</b>	<b>while</b>				

### 3.3 Les commentaires

- Un commentaire débute par `/*` et se termine par `*/`.

- **Exemple :**

```
/* Ceci est un commentaire */
```

- On ne peut pas imbriquer des commentaires.

```
/*
    Un commentaire
    /* Un autre commentaire */
*/
```

# 4 La structure d'un programme

- Un programme C est une ou plusieurs fonctions stockées dans un ou plusieurs fichiers dont l'une doit s'appeler **main**.
- Une fonction est formée d'un entête (type et nom de la fonction suivis d'une liste d'arguments entre parenthèses) et d'instruction composée constituant le corps de la fonction.

# 4 La structure d'un programme

- Un programme C se présente de la façon suivante :

[directives du préprocesseur]

[déclarations de variables externes]

[fonctions secondaires]

```
int main() {
```

déclarations de variables internes

instructions

```
    return 0;
```

```
}
```

## 4.1 Les directives du préprocesseur (précompilation)

- Elles commencent toutes par un #.

Commande	Signification
<code>#include &lt;stdio.h&gt;</code> <code>#include &lt;math.h&gt;</code> <code>#include &lt;conio.h&gt;</code>	<i>permet d'utiliser les fonctions printf() et scanf()</i> <i>permet d'utiliser les fonctions mathématiques</i> <i>permet d'utiliser la fonction getch()</i>
<code>#define PI 3.14159</code> <code>#undef PI</code>	<i>définit la constante PI</i> <i>à partir de cet endroit, la constante PI n'est plus définie</i>
<code>#ifdef PI</code> instructions 1 ... <code>#else</code> instructions 2 ... <code>#endif</code>	<i>si la constante PI est définie, on compile les instructions 1,</i> <i>sinon, les instructions 2</i>

## 4.2 Les instructions

- Une *instruction* est une expression suivie de ‘ ; ’.
- Le point-virgule signifie en quelque sorte « *évaluer cette expression* ».
- Plusieurs instructions peuvent être rassemblées par des accolades { et } pour former une *instruction composée* ou *bloc* qui est syntaxiquement équivalent à une instruction.
- **Exemple :**

```
if (x != 0) {  
    z = y / x;  
    t = y % x;  
}
```

## 4.3 Les instructions

- Une *instruction* composée d'un *spécificateur de type* et d'une liste d'identificateurs séparés par une virgule est une déclaration.
- Exemple :

```
int a;  
int b = 1, c;  
double x = 2.38e4;  
char message[80];
```

- En C, toute variable doit faire l'objet d'une déclaration avant d'être utilisée.

# 4.3 Les instructions

## ● 2.4 Exemple

```
#include <stdio.h>
#define PI 3.14159
#include <conio.h>
/* calcul de la surface d'un cercle */
main() {
    float rayon, surface;
    float calcul(float );
    printf("Rayon = ? ");
    scanf("%f", &rayon);
    surface = calcul(rayon);
    printf("Surface = %f\n", surface);
    getch();
}
/* définition de fonction */
float calcul(float r) {
    /* définition de la variable locale */
    float a;
    a = PI * r * r;
    return(a);
}
```

## 4.5 Les types de base

- Le langage C contient des types de base
  - les entiers (**int**),
  - les réels (**float**) simple et double précision (**double**)
  - et les caractères (**char**).
- De plus il existe un type ensemble vide : le type **void**.
- Les mots-clés **short** et **long** permettent d'influer sur la taille mémoire des entiers et des réels.

## 4.6 Les constantes

- Une constante est une valeur qui apparaît littéralement dans le code source d'un programme.
- Constantes entières 1, 2, 3, ...
- Constantes caractères 'a', 'A', ...
- Constantes chaînes de caractères "**Bonjour**"
- Pas de constantes logiques
  - Pour faire des tests, on utilise un entier. **0** est équivalent à faux et tout ce qui est  **$\neq 0$**  est vrai.

# 4.7 Les variables

- Les variables peuvent stocker des chiffres des nombres, des caractères, des chaînes de caractères,... dont la valeur peut être modifiée au cours de l'exécution du programme.
- Pour déclarer une variable, on fait précédé son nom par son type.
- Exemples de déclarations :

Déclaration	Signification
<code>int a ;</code>	a est entier
<code>int z=4 ;</code>	z est entier et vaut 4
<code>unsigned int x ;</code>	x est un entier positif (non signé)
<code>float zx, zy ;</code>	zx et zy sont de type réel
<code>float zx=15.15 ;</code>	zx est de type réel et vaut 15.15
<code>double z ;</code>	z est un réel en double précision
<code>char zz ;</code>	zz est une variable caractère
<code>char zz='a' ;</code>	zz vaut 'a'

## 4.8 Les opérateurs

- **L'affectation (=)** : sert à mettre dans la variable de gauche la valeur de ce qui est à droite.
- Sa syntaxe est la suivante : **variable = expression**
- **Exemple :**

```
main() {
    int i, j = 2;
    float x = 2.5;
    i = j + x;
    x = x + i;
    printf("%f", x);
}
```

- imprime pour **x** la valeur **6.5** (et non **7**),
- car dans l'instruction **i = j + x;**, l'expression **j + x** a été convertie en entier.

# 4.8 Les opérateurs

- **Les opérateurs arithmétiques :** Les opérateurs arithmétiques classiques sont l'opérateur unaire - (changement de signe) ainsi que les opérateurs binaires.

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	reste de la division (modulo)

## 4.8 Les opérateurs

- Exemple :

```
float x;  
x = 3 / 2;
```

- affecte à **x** la valeur **1**.
- Par contre :

```
x = 3 / 2.;
```

- affecte à **x** la valeur **1 . 5**.

# 4.8 Les opérateurs

- **Les opérateurs relationnels** : servent à comparer deux expressions

>	strictement supérieur
$\geq$	supérieur ou égal
<	strictement inférieur
$\leq$	inférieur ou égal
$=$	égal
$\neq$	différent

- **Remarque** : Ne pas confondre l'opérateur d'affectation  $=$  et l'opérateur de comparaison  $==$ .

# 4.8 Les opérateurs

- Exemple :

```
main() {  
    int a = 0;  
    int b = 1;  
    if (a == b)  
        printf("\n a et b sont égaux \n");  
    else  
        printf("\n a et b sont différents \n");  
}
```

# 4.8 Les opérateurs

- **Les opérateurs logiques booléens :**

& &	<b>et</b> logique
	<b>ou</b> logique
!	<b>négation</b> logique

- Comme pour les opérateurs de comparaison, la valeur retournée par ces opérateurs est un **int** qui vaut **1** si la condition est vraie et **0** sinon.

# 4.8 Les opérateurs

- Les opérateurs d'affectation composée :

`+ =      - =      * =      / =      % =`

- Exemple :

```
i+=5 ;      /* i=i+5 */  
i-=3 ;      /* i=i-3 */  
i*=4 ;      /* i=i*4 */  
i/=2 ;      /* i=i/2 */  
i%=3 ;      /* i=i%3 */
```

# 4.8 Les opérateurs

- Les opérateurs d'incrémentation `++` et de décrémentation `--`: s'utilisent aussi bien en postfixé (`i++`) qu'en préfixé (`++i`).
- Dans les deux cas la variable `i` sera incrémentée, toutefois dans la notation postfixé la valeur renournée sera l'ancienne valeur de `i` alors que dans la notation préfixé se sera la nouvelle.
- Exemple :

```
int a = 3, b, c;  
b = ++a;           /* a et b valent 4 */  
c = b++;          /* c vaut 4 et b vaut 5 */
```

## 4.8 Les opérateurs

- **L'opérateur virgule :** Une expression peut être constituée d'une suite d'expressions séparées par des virgules :

```
expression-1, expression-2, ..., expression-n
```

- Cette expression est alors évaluée de gauche à droite. Sa valeur sera la valeur de l'expression de droite.
- **Exemple :**

```
main() {
    int a, b;
    b = ((a = 3), (a + 2));
    printf("\n b = %d \n", b);
}
```

- imprime  $b = 5$ .

## 4.8 Les opérateurs

- L'opérateur conditionnel ternaire `? :`

```
condition ? expression-1 : expression-2
```

- Cette expression est égale à `expression-1` si condition est satisfaite, et à `expression-2` sinon.
- Par exemple, l'expression

```
x >= 0 ? x : -x
```

- correspond à la valeur absolue d'un nombre.

## 4.8 Les opérateurs

- De même l'instruction

```
m = ((a > b) ? a : b);
```

- affecte à **m** le maximum de **a** et de **b**.
- Et l'instruction :

```
d = (a > b) ? ((a>c) ? a : c) : (( b > c) ? b : c);
```

- affecte à **d** le maximum de **a** de **b** et de **c**.

## 4.8 Les opérateurs

- De même l'instruction

```
m = ((a > b) ? a : b);
```

- affecte à **m** le maximum de **a** et de **b**.
- Et l'instruction :

```
d = (a > b) ? ((a>c) ? a : c) : (( b > c) ? b : c);
```

- affecte à **d** le maximum de **a** de **b** et de **c**.

# 4.8 Les opérateurs

- L'opérateur de conversion de type : appelé cast, permet de modifier explicitement le type d'un objet.
- On écrit :

```
(type) objet
```

- Exemple :

```
main() {
    int i = 3, j = 2;
    printf("%f \n", (float)i/j);
}
```

- Affiche la valeur 1.5.

## 4.8 Les opérateurs

- L'opérateur d'adresse & : appliqué à une variable retourne l'adresse-mémoire de cette variable.

**&objet**

- Exemple:

```
main() {
    int i = 3;
    printf("i = %d et @i = %d\n", i, &i);
}
```

- Cela imprime : i = 3 et @i = 57573556

# 5. Les instructions de contrôle

- Le langage C dispose d'instructions de contrôle permettant de réaliser :

- **des choix** :

- L'instruction conditionnelle **if...else...**
    - L'instruction d'aiguillage ou de choix **switch...case...default...**

- **des boucles** les instruction répétitives :

- **while...do...**
    - **do...while...**
    - **for...**

- **des branchements** Les instructions de branchement inconditionnel

- **goto**
    - **break** ( associé aux boucles)
    - **continue** ( associé aux boucles)
    - **return**

## 5.1 Les instructions de branchement conditionnel

- Toute instruction qui permet de contrôler le fonctionnement d'un programme.
- On distingue les instructions de branchement et les boucles.
- Les instructions de branchement permettent de déterminer quelles instructions seront exécutées et dans quel ordre.

## 5.1.1 Branchement conditionnel if -- else

- L'instruction conditionnelle **if** -- **else** permet de tester une condition puis d'exécuter une action parmi deux actions possibles.
- La forme la plus générale est celle-ci :

```
if (expression-1 )
    instruction-1
else if (expression-2 )
    instruction-2
...
else if (expression-n )
    instruction-n
else
    instruction-∞
```

- avec un nombre quelconque de **else if** ( ... ).

## 5.1.1 Branchement conditionnel if -- else

- Le dernier **else** est toujours facultatif.
- La forme la plus simple est :

```
if (expression )  
    instruction
```

- Chaque instruction peut être un bloc d'instructions.

## 5.1.2 Branchement multiple switch

- L'instruction **switch** est une instruction de choix multiple. Elle permet d'évaluer une expression puis d'exécuter une action parmi plusieurs actions étiquetées.
- Si la valeur de l'expression correspond à une des étiquettes, l'action correspondante est exécutée .
- Sa forme la plus générale est :

```
switch (expression) {  
    case constante-1:    liste d'instructions 1  
                          break;  
    case constante-2:    liste d'instructions 2  
                          break;  
    ...  
    case constante-n:    liste d'instructions n  
                          break;  
    default:             liste d'instructions ∞  
                          break;  
}
```

## 5.1.2 Branchement multiple switch

- « **expression** » doit être une expression entière
- « **la constante** » doit être une constante entière( de type : **int, char, short, long, unsigned**).
- Par contre « **default** » est facultatif.
- L'expression est comparée successivement ( de haut en bas) aux constantes spécifiées après chaque “**case**”.
- S'il y a égalité, alors les instructions correspondantes à ce « **case** » sont exécutées.
- Dans le cas où aucune égalité n'est vérifiée, les instructions spécifiées après “**default**” sont exécutées.
- Pour sortir de l'instruction **switch** délimitée par { et } et continuer en séquence, on doit utiliser l'instruction **break** .

## 5.1.2 Branchement multiple switch

- Exemple:

```
#include <stdio.h>
main() {
    int a, b, r;
    char op;
    printf("Donner deux operandes : ");
    scanf("%d %d", &a, &b);
    printf("Donner l'operateur : ");
    fflush(stdin);
    scanf("%c", &op);
    switch(op) {
        case '+': r = a + b;
                     printf("%d %c %d = %d\n", a, op, b, r); break;
        case '-': r = a - b;
                     printf("%d %c %d = %d\n", a, op, b, r); break;
        case '*': r = a * b;
                     printf("%d %c %d = %d\n", a, op, b, r); break;
        case '/': r = a / b;
                     printf("%d %c %d = %d\n", a, op, b, r); break;
        default : printf("Operateur invalide !!!");
    }
}
```

## 5.1.3 Les boucles

- Les boucles permettent de répéter une série d'instructions tant qu'une certaine condition n'est pas vérifiée.
  - L'instruction **while**
  - L'instruction **do...while...**
  - L'instruction **for**

# Boucle while

- Tant qu'une condition spécifiée n'est pas vérifiée, elle permet de répéter une ou plusieurs actions.
- La syntaxe de **while** est la suivante :

```
while (expression )  
    instruction
```

- Si expression est **nulle** au départ, **instruction** ne sera jamais exécutée.
- **instruction** peut évidemment être une instruction composée.

# Boucle while

- Par exemple, le programme suivant imprime les entiers de 1 à 9.

```
i = 1;  
while (i < 10) {  
    printf("\n i = %d", i);  
    i++;  
}
```

# Boucle do---while

- Il peut arriver que l'on ne veuille effectuer le test de continuation qu'après avoir exécuté l'instruction.
- Dans ce cas, on utilise la boucle **do---while**.
- Sa syntaxe est :

```
do  
    instruction  
while (expression);
```

- Ici, instruction sera exécutée tant que **expression** est non nulle.
- Cela signifie donc que **instruction** est toujours exécutée au moins une fois.

# Boucle do---while

- Par exemple, pour saisir au clavier un entier entre 1 et 10 :

```
int a;
do{
    printf("Entrez un entier entre 1 et 10 : ");
    scanf("%d", &a);
}while ((a <= 0) || (a > 10));
```

# Boucle for

- L'instruction **for** est une instruction de boucle faisant intervenir l'initialisation, le test de limite et l'incrémentation en une seule action.
- La syntaxe de **for** est :

```
for ( expr 1 ; expr 2 ; expr 3 )
    instruction
```

- Une version équivalente plus intuitive est :

```
expr 1;
while (expr 2 ) {
    instruction
    expr 3;
}
```

# Boucle for

- Par exemple, pour imprimer tous les entiers de 0 à 9, on écrit :

```
for (i = 0; i < 10; i++)
    printf("\n i = %d", i);
```

- Alla fin de cette boucle, **i** vaudra **10**.