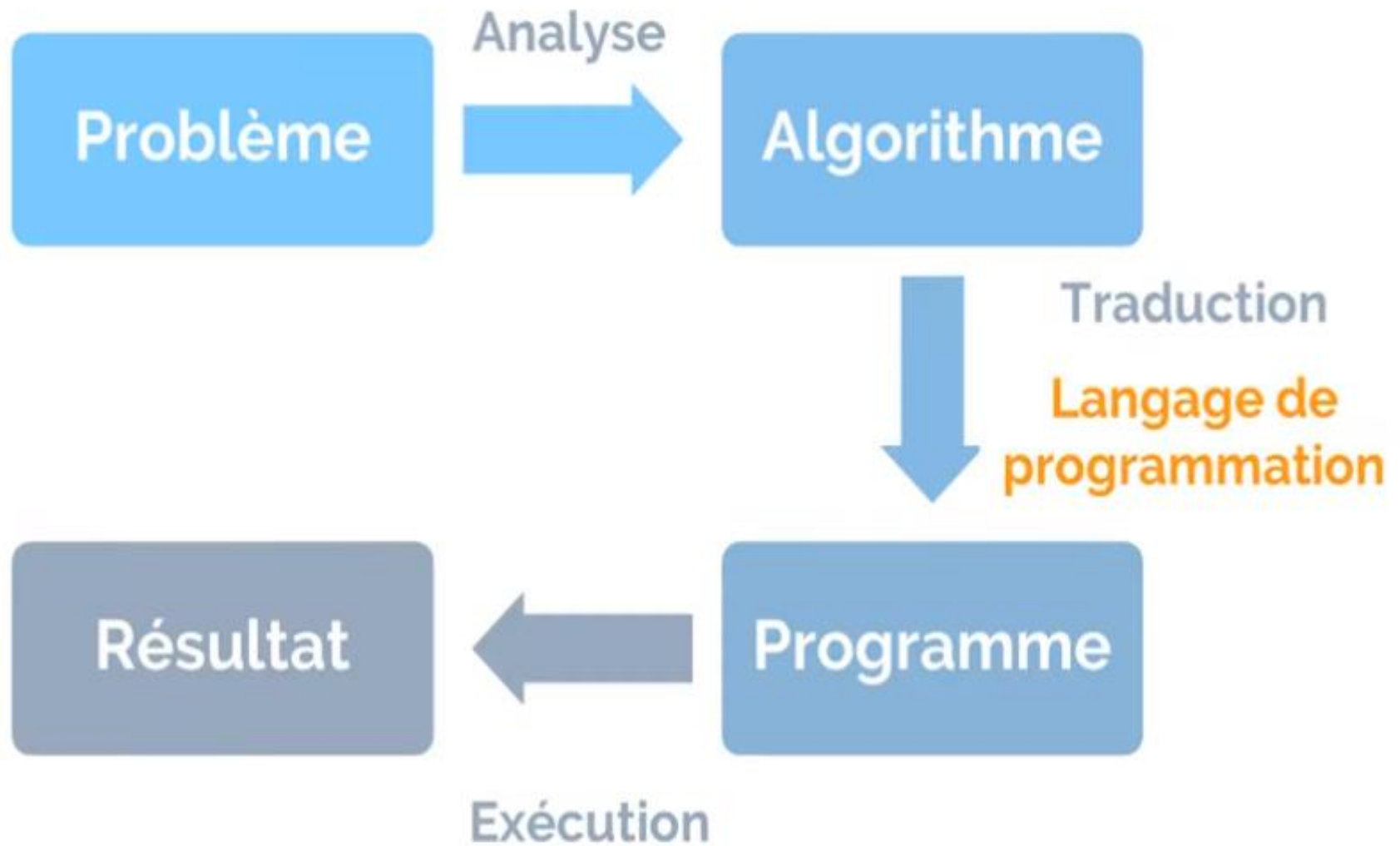


# *Programmation C*

Pr. Abdelali El Gourari



# Introduction



# Introduction: Langage de programmation



# Introduction: Pourquoi le langage de programmation C



Multi-plateformes



Gestion de la Mémoire

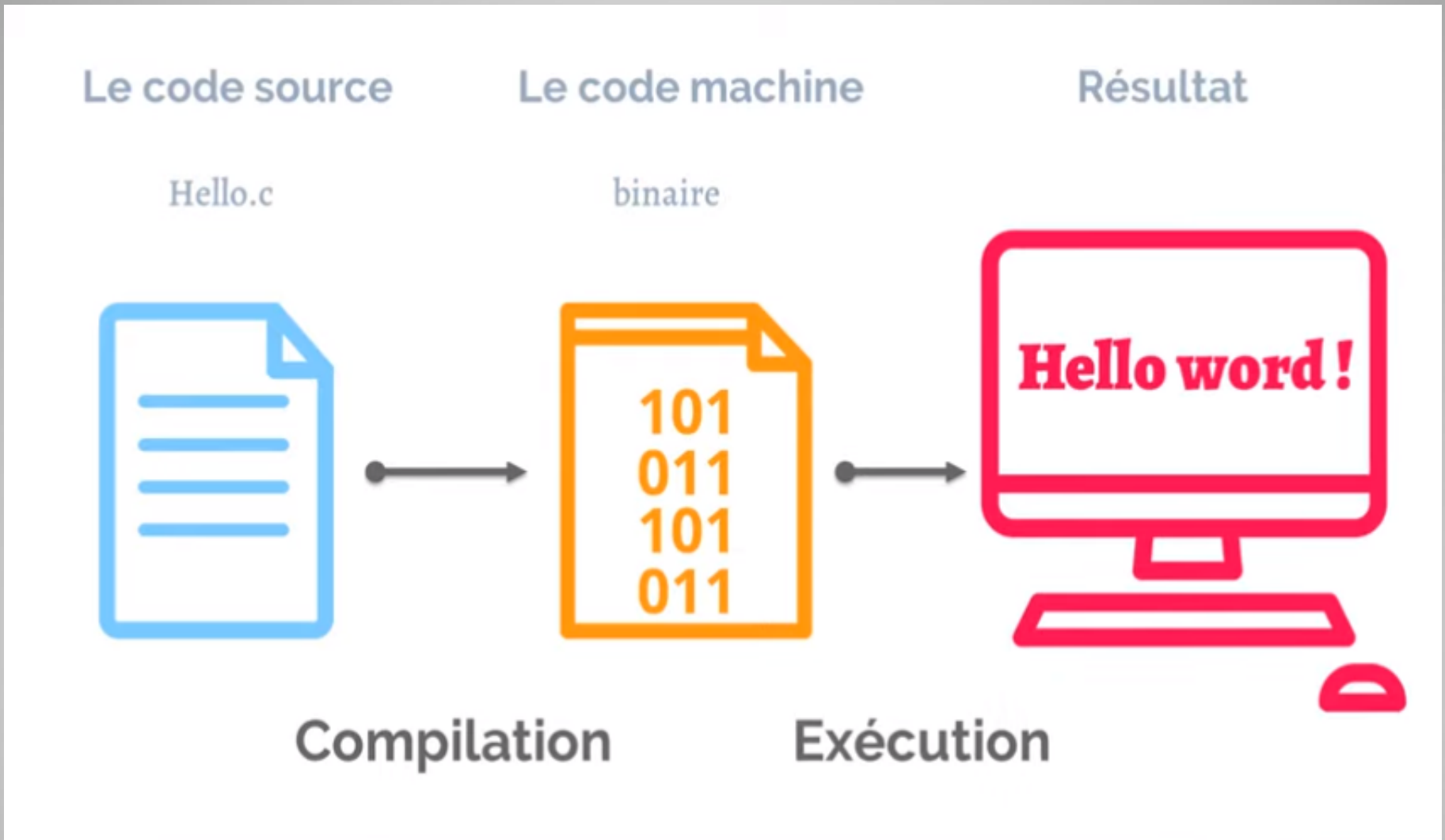


Exécution rapide



Langage compilé

# Introduction: Pourquoi le langage de programmation C

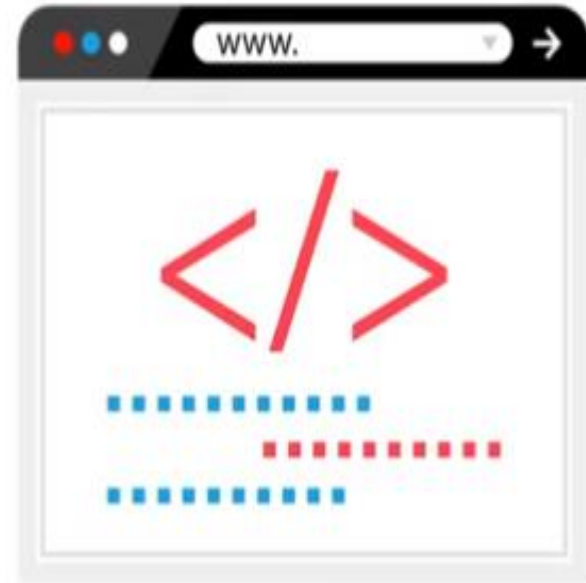


# Introduction: Le premier programme en C



Code::Blocks  
Visual C++  
Xcode

...



[www.tutorialspoint.com](http://www.tutorialspoint.com)  
[www.onlinegdb.com](http://www.onlinegdb.com)  
[www.repl.it](http://www.repl.it)

...

# Déclaration, Printf, Scanf, structure d'un programme en C et les opérateurs



Instructions  
de base



Structures  
conditionnelles

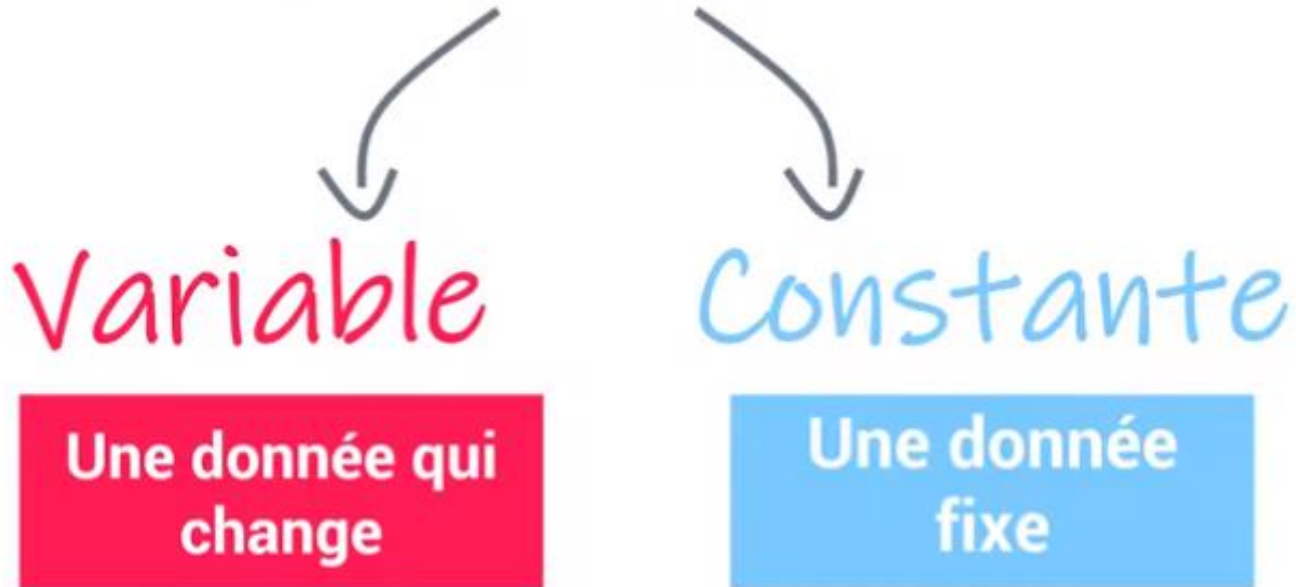


Structures  
répétitives



Tableaux

## Type de données





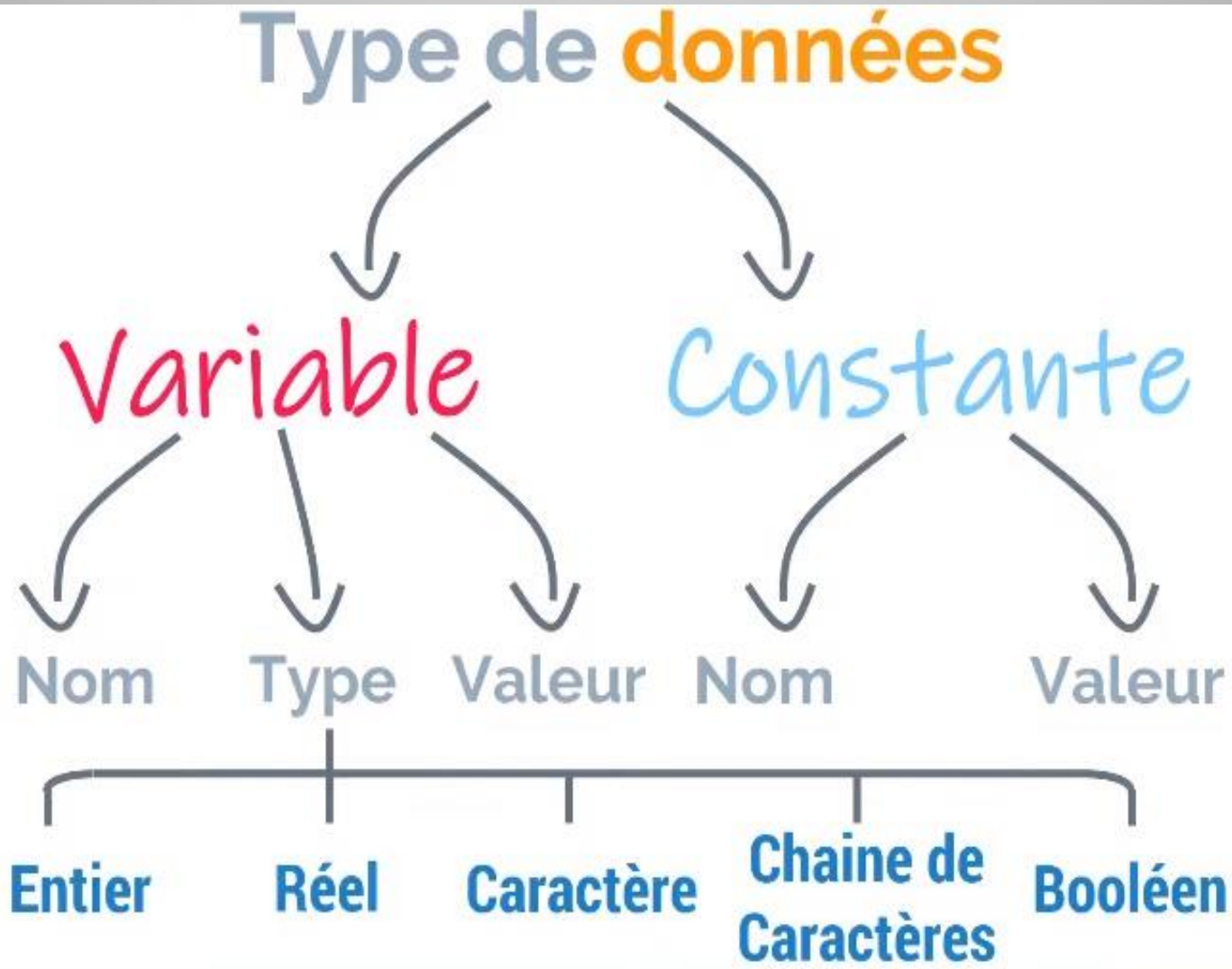
# Types des données: Variable et Constantes

Relevé de notes			
<b>Nom étudiant</b>		<i>Ali</i>	
<b>N°</b>	<i>300</i>	<b>Sexe</b>	<i>H</i>
<b>Nbr étudiants</b>		<i>400</i>	

<b>Matière</b>	<b>Note</b>	<b>Valide</b>
<i>Algorithmique #1</i>	<i>18</i>	<i>Oui</i>
<i>Algorithmique #2</i>	<i>15.5</i>	<i>Oui</i>
<i>Programmation C</i>	<i>08</i>	<i>Non</i>
<i>Programmation C++</i>	<i>17</i>	<i>Oui</i>
	<b>Moyenne</b>	<i>15.37</i>
	<b>Mention</b>	<i>Bien</i>


Variables
<b>Nom étudiant</b>
<b>N°</b>
<b>Sexe</b>
<b>Matière</b>
<b>Note</b>
<b>Valide</b>
<b>Moyenne</b>
<b>Mention</b>

# Types des données: variables et constantes




## Types des données - Entier

Type	Valeur min	Valeur max	Nombre d'octets
int	-32768	32767	2
long	-2147483648	2147483647	4
unsigned int	0	65535	2
unsigned long	0	4294967295	4



## Types des données - Réel

Type	Valeur min	Valeur max	Nombre d'octets
float	$3.4 \times 10^{-38}$	$3.4 \times 10^{+38}$	4
double	$1.7 \times 10^{-308}$	$1.7 \times 10^{+308}$	8
long double	$3.4 \times 10^{-4932}$	$1.1 \times 10^{+4932}$	10



## Types des données - Caractère

Type	Valeur min	Valeur max	Nombre d'octets
char	-128	127	1
unsigned char	0	255	1

## Types des données - Booléen

Type	Valeurs	Nombre de bits
bool	0 ou 1	1

## ***Définition:***

Les données sont des informations nécessaires au déroulement d'un algorithme. On distingue deux catégories :

- **Constante** : une donnée fixe qui **ne varie pas** durant l'exécution d'un algorithme.
- **Variable** : une donnée dont le contenu peut être **modifié** par une action durant l'exécution d'un algorithme.

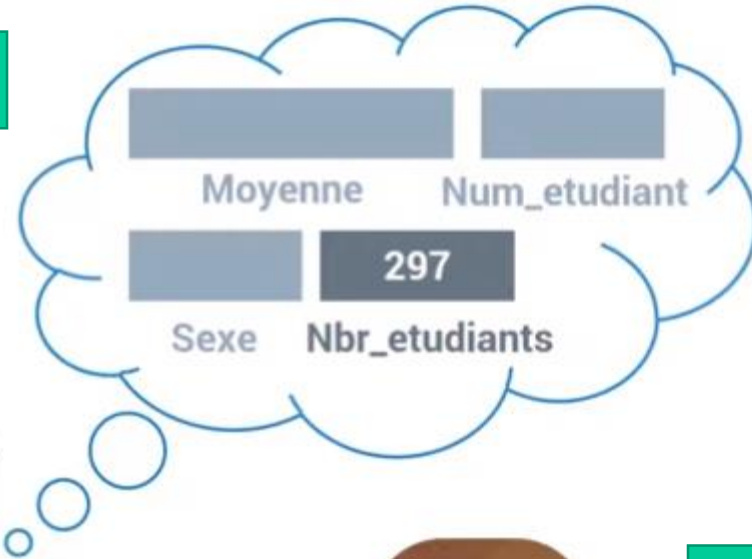
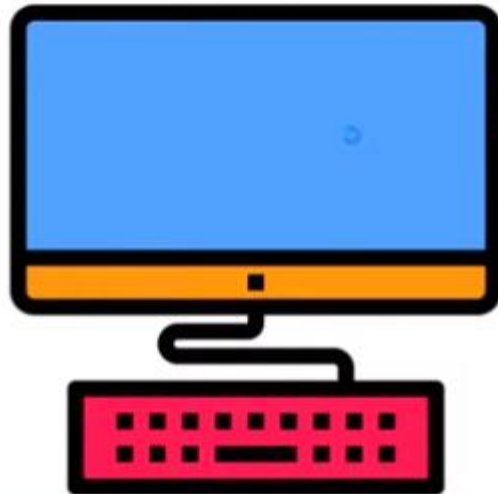
# Exercice: Types des données

Donner le type des données suivantes :

Donnée	Type
"d"	
-300	
"8" 	
2506.5	
$6 \times 10^{+802}$	
87	
true	

# Déclaration

RAM



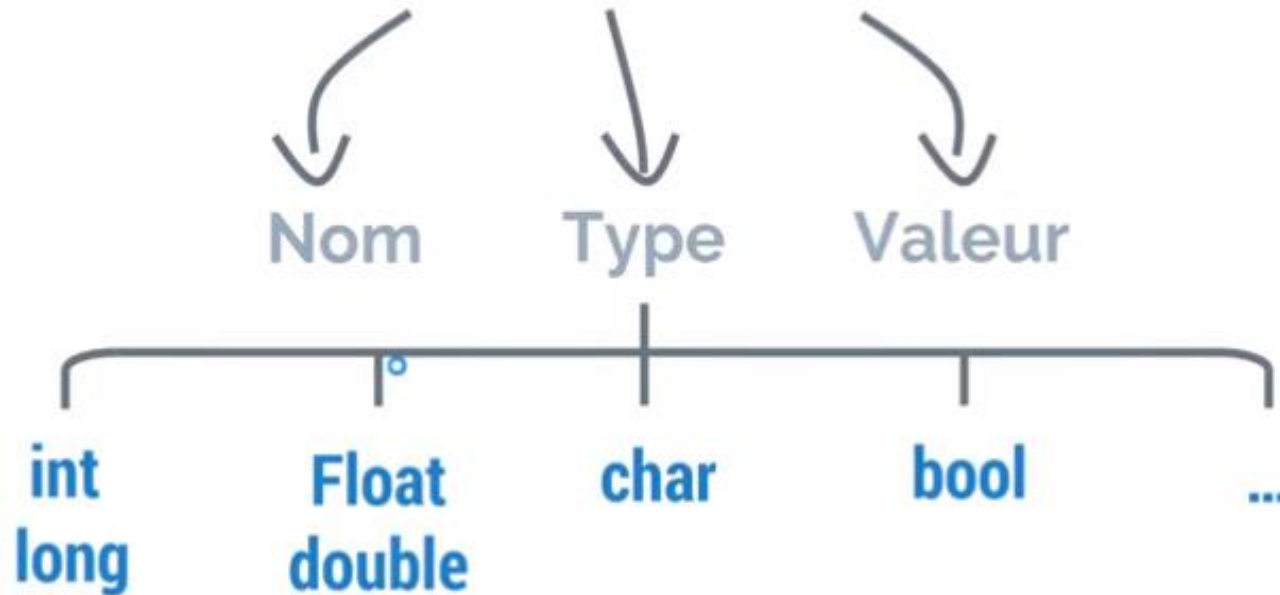
Abdelali



# Déclaration: Les constantes

## Syntaxe de déclaration d'une constante en C

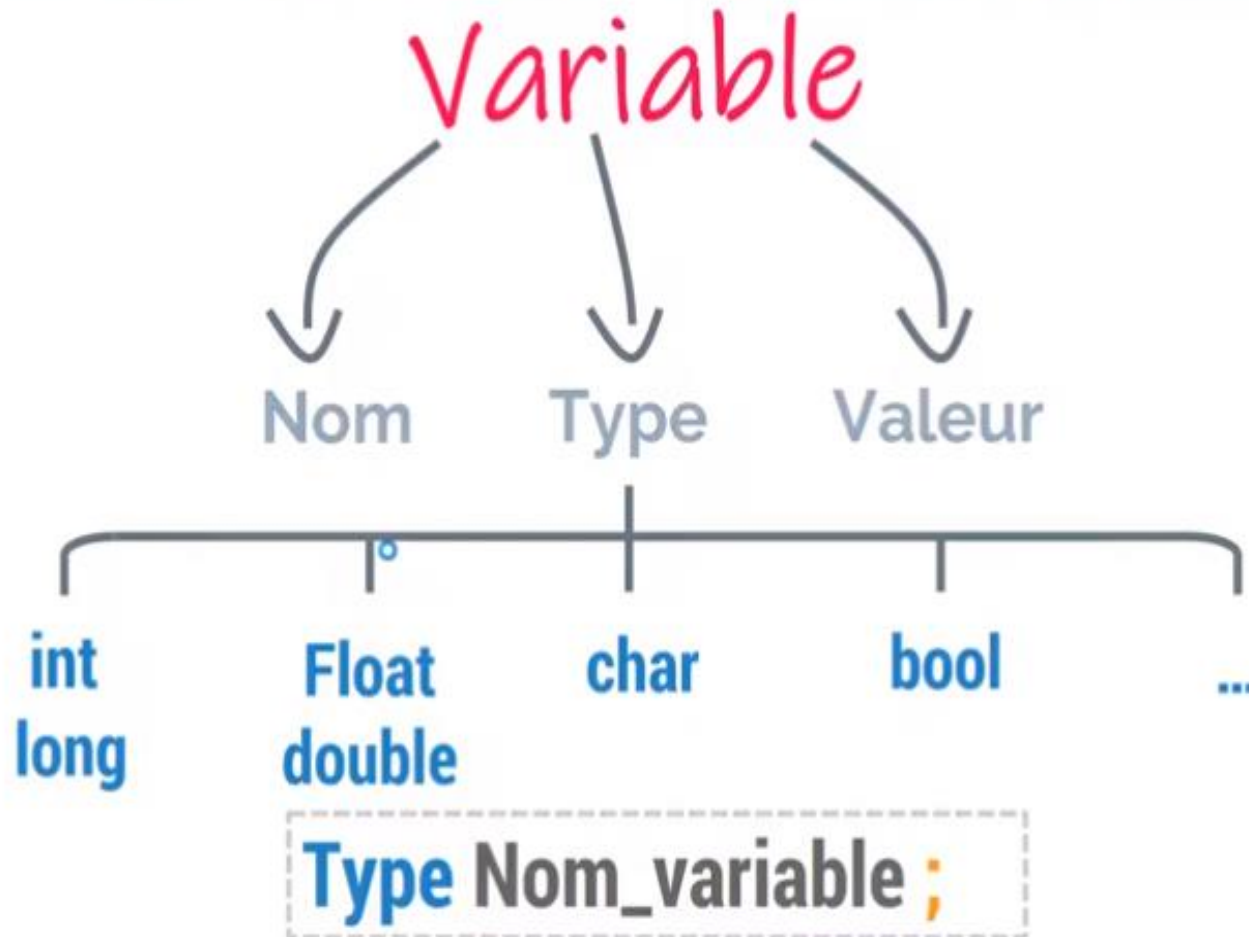
Constante



```
Const Type Nom_constant = valeur ;
```



## Syntaxe de déclaration d'une variable en C



## ***Définition:***

La déclaration permet d'informer l'ordinateur l'existence d'une donnée.

C'est-à-dire demander à l'ordinateur la permission de réserver un espace de la mémoire où l'on peut **stocker** et **récupérer** l'information.

## Exemples de constantes :

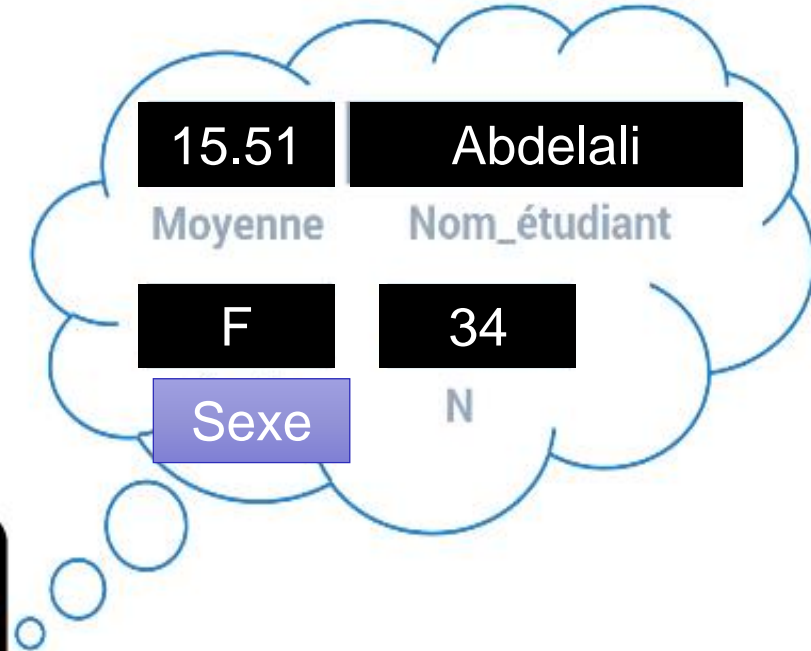
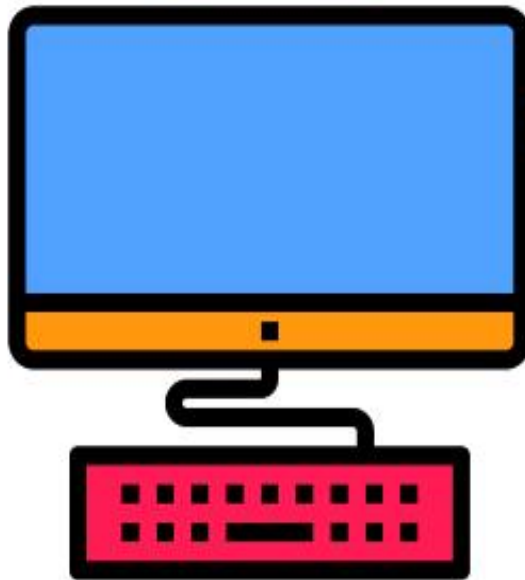
```
Const float Pi = 3.14 ;  
Const int nbr_mois = 12 ;
```

## Exemples de variables :

```
int num_Etudiant ;  
float note ;  
char sexe ;  
bool admis ;
```

# Affectation

User



```
Nom_Variable = Valeur ;
```

```
Moyenne = 15.51 ;
```

```
Sexe = "F" ;
```

```
N = 34 ;
```

## ***Définition:***

L'affectation est une opération qui consiste à attribuer à une variable :

- soit une valeur **particulière**,
- soit une valeur **contenue** dans une autre variable
- soit une valeur **calculée** à l'aide d'opérateurs arithmétiques.

Elle est représentée par **=**

# Exercice: Déclaration et Affectation

Soient trois variables A, B et C tels que :

A est de type entier

B est de type caractère

C est type logique

2- Cochez ce qui est juste :

A = 1 ;

A = "D" ;

B = 3 ;

B = A ;

B = "3" ;

C = 10 ;

C = 2 < 5 ;

C = 1 < -22 ;

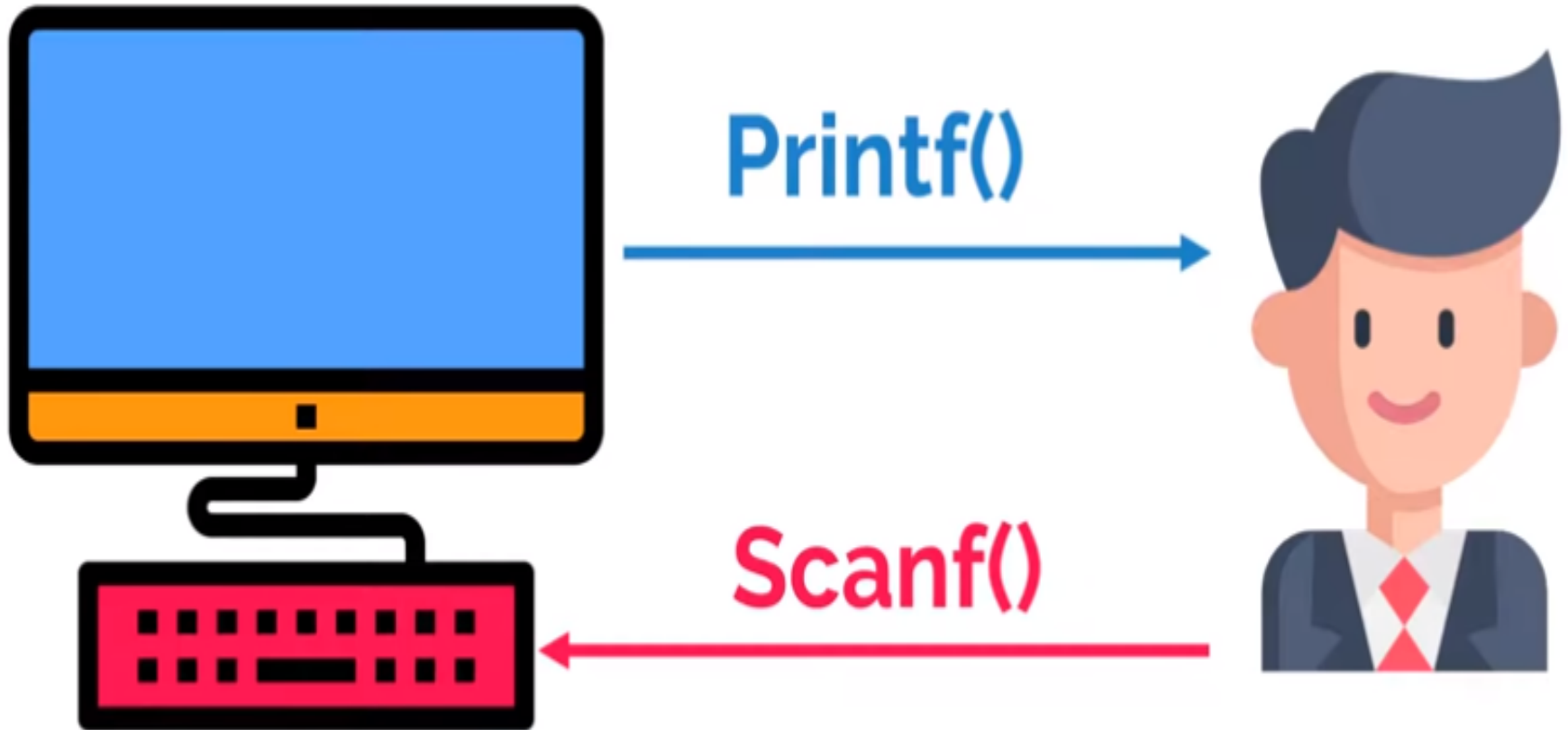
# Exercice: Affectation

Complétez le tableau suivant :

Instructions	Variables			
	A	B	C	D
$B = 2 ;$				
$C = B + 10 ;$				
$A = 4 ;$				
$D = A ;$				
$B = B * D ;$				
$C = B + 5 ;$				
$A = 10 + 4 + C ;$				
$C = A + B + D ;$				

4

# Printf() et Scanf()





## *Définition*

**Printf()** permet d'afficher la valeur d'une expression sur l'écran. Une expression peut être :

- Une chaîne de caractères (mettre la chaîne de caractères entre deux apostrophes),
- Un nombre, une variable numérique, un résultat d'une opération entre plusieurs variables.

## Syntaxe

```
Printf ( " Texte ... " , var, cons, expr , ... ) ;
```

- Le texte à afficher,
- Les spécificateurs de format
- Caractère d'échappement.

Variables, constantes ou expressions dont les valeurs sont à afficher

# L'instruction d'écriture → Printf()

## Exemple

```
Printf ( " Maroc " );
```

Signifie affiché sur l'écran le message suivant : **Maroc**

```
Printf ( " A = %d " , A );
```

Signifie affiché sur l'écran le **contenu** de la variable **A**.

```
Printf ( " les coordonnées sont : %f , %f " , X , Y );
```

Signifie affiché sur l'écran le message suivant: **les coordonnées sont :** Plus les contenus des variable X et Y.

# L'instruction d'écriture → Printf()

Les spécificateurs de format	Type
" %d "	int
" %ld "	long
" %f "	float ou double
" %c "	char

Les caractères d'échappement	Type
" \n "	Retour à la ligne
" \t "	Tabulation horizontale
" \' "	Affiche une apostrophe
" \" "	Affiche un guillemet
" \\ "	Affiche un antislash
" %% "	Affiche un %

# L'instruction de lecture → scanf()

## Syntaxe

```
scanf ( " format_1 , ... " , & var_1 , ... );
```

Spécificateur  
de format

Adresse

Nom de  
variable

## Exemples :

```
scanf ( "%d" , &x );
```

Pour demander la valeur de x (x est une variable entière)

```
scanf ( "%c" , &y );
```

Pour demander la valeur de y (y est une variable de type caractère)

```
scanf ( "%d%d%d" , &jour, &mois, &annee );
```

Pour demander le jour, le mois et l'année.



# L'instruction de lecture → scanf()

## **Remarque**

Lors de l'exécution de l'instruction de la fonction scanf() la machine attend que l'utilisateur lui fournisse une valeur afin de pouvoir continuer à exécuter l'algorithme.

## **Exercice**

Nous voulons écrire un programme qui calcule l'aire d'un cercle.

- 1 - Donner les instruction de déclaration.
- 2 - Donner les instructions qui demandent à l'utilisateur de taper les valeur des données.
- 3 - Donner les instructions de traitement
- 4 - Donner les instructions qui permettent d'afficher le résultat

## Exercice → printf() et scanf()

Nous voulons écrire un programme qui calcule l'aire d'un cercle.

- 1 - Donner les instruction de déclaration.
- 2 - Donner les instructions qui demandent à l'utilisateur de taper les valeur des données.
- 3 - Donner les instructions de traitement
- 4 - Donner les instructions qui permettent d'afficher le résultat

```
float Rayon , Surface ;  
const float Pi = 3.14 ;  
printf ( " Veuillez entrer la valeur du rayon de cercle : " ) ;  
scanf ( " %f " , &Rayon ) ;  
Surface = Rayon * Rayon * Pi ;  
printf ( " L'aire de cercle est : %f " , Surface ) ;
```

# La structure d'un programme C

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



Appel des bibliothèques

```
int main () {
```



Fonction principale

```
    printf ( " Bonjour " );
```

```
    return 0 ;
```



Partie traitement

```
}
```





# Exercice: La structure d'un programme C

Ecrire un algorithme qui demande à l'utilisateur de taper la quantité de produits vendus, le prix de vente et qui affiche le chiffre d'affaires de l'entreprise.

produits\_vendus

prix\_vente

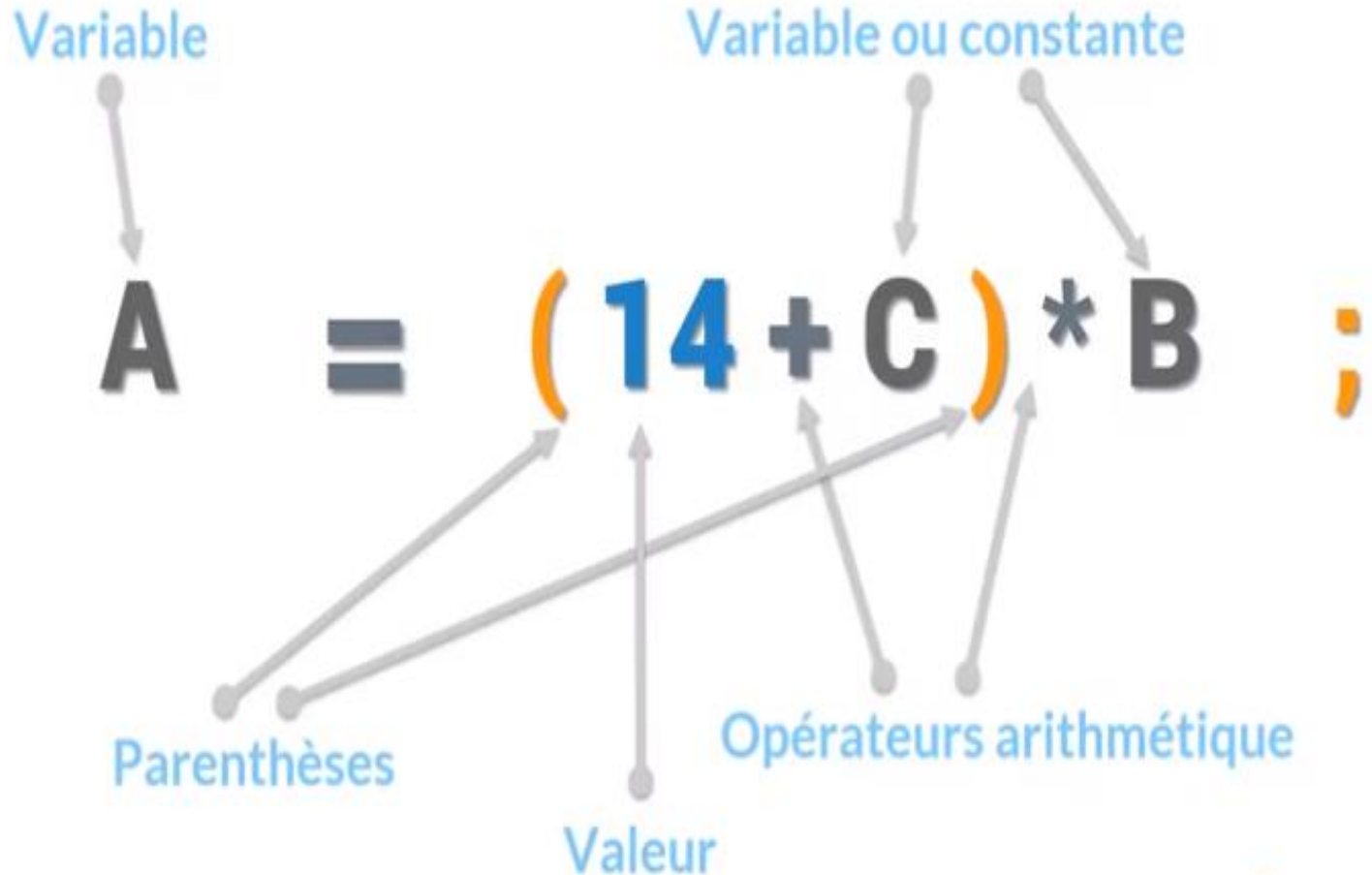


Programme



CA

# Expression arithmétique



- Une **expression arithmétique** est formée par des combinaisons d'objets numériques (entier et réel) et des opérateurs arithmétiques.
- Une expression arithmétique donne un **résultat numérique** dont le type est entier ou réel.

# Expression arithmétique

Les opérateurs arithmétiques usuels sont :

Opérateur	Signification
<b>+</b> , <b>-</b>	Addition , Soustraction
<b>*</b> , <b>/</b>	Multiplication , Division
<b>%</b>	Reste de la division entière
<b>++</b>	Incrémentement
<b>--</b>	Décrémentement

# Expression arithmétique

**A = 10 / 3;**

10 | 3  
—  
3.33

1

A

**A = 10 % 3;**

10 | 3  
1 | 3

**A ++;**      **A = A + 1;**

**A --;** → **A = A - 1;**



# Puissance en langage C

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

```
#include <math.h>
```

```
int main () {
```

```
    int X, Y, Z;
```

```
    X = 4;
```

```
    Y = 3;
```

```
    Z = pow (X, Y);
```

```
    printf (" %d ", Z);
```

```
    return 0;
```

```
}
```

Appel de la librairie **math.h**

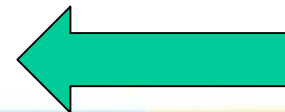
Utiliser la fonction **pow**



# Division entière en langage C

```
#include <stdio.h >
int main () {
    float X, Y;
    int Z;
    Z = X / Y;
    printf (" %d ", Z );
    return 0;
}
```

Il suffit d'affecter le résultat de la division à une **variable entière**





# Expression arithmétique élargie

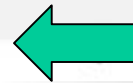
Opérateur	Opération normale	Opération élargie
<code>+=</code>	<code>X = X + Y ;</code>	<code>X += Y ;</code>
<code>-=</code>	<code>X = X - Y ;</code>	<code>X -= Y ;</code>
<code>*=</code>	<code>X = X * Y ;</code>	<code>X *= Y ;</code>
<code>/=</code>	<code>X = X / Y ;</code>	<code>X /= Y ;</code>
<code>%=</code>	<code>X = X % Y ;</code>	<code>X %= Y ;</code>



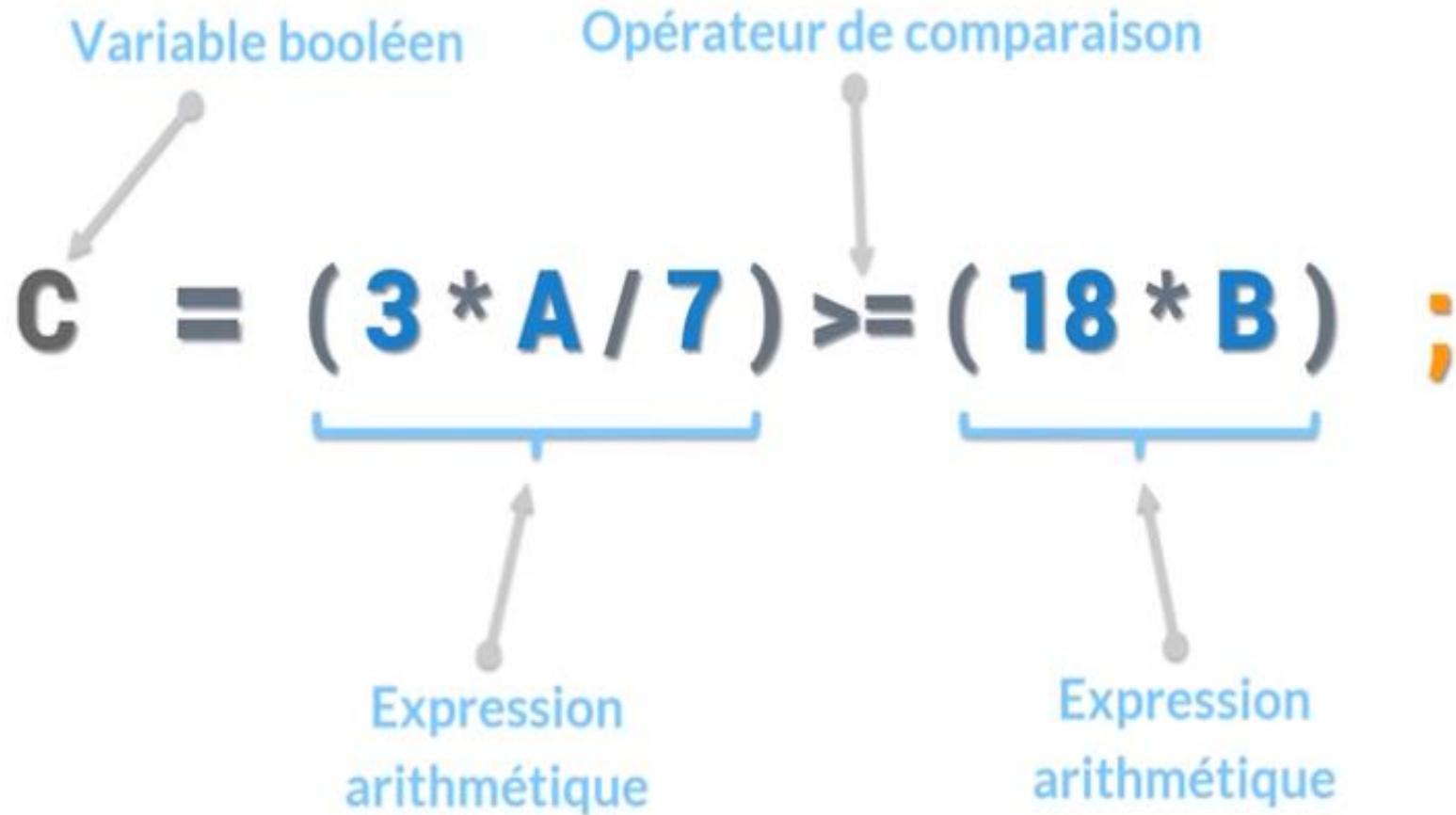
# Exercice: Expression arithmétique

Donner les valeurs des variables (de type float) après chaque instruction:

Instruction	A	B	C	D
<code>A = 4 + 2 ;</code>	6			
<code>B = ++ A ;</code>				
<code>C = A * B ++ ;</code>				
<code>D = pow ( C , 2 ) ;</code>				
<code>A /= 2 ;</code>				
<code>D = ++D - C ;</code>				
<code>C -= B-- ;</code>				
<code>B = (A-- + --C) / B ;</code>				



# Expression de comparaison



# Expression de comparaison

Une expression de comparaison donne un résultat booléen (vrai ou faux). Les opérateurs de comparaison usuels sont :  $>$  ,  $==$  ,  $<$  ,  $>=$  ,  $<=$  ,  $!=$ .

Exemple :

$$C = (3 * A / 7) <= (18 * B)$$

Expression  
arithmétique

Opérateur de  
comparaison

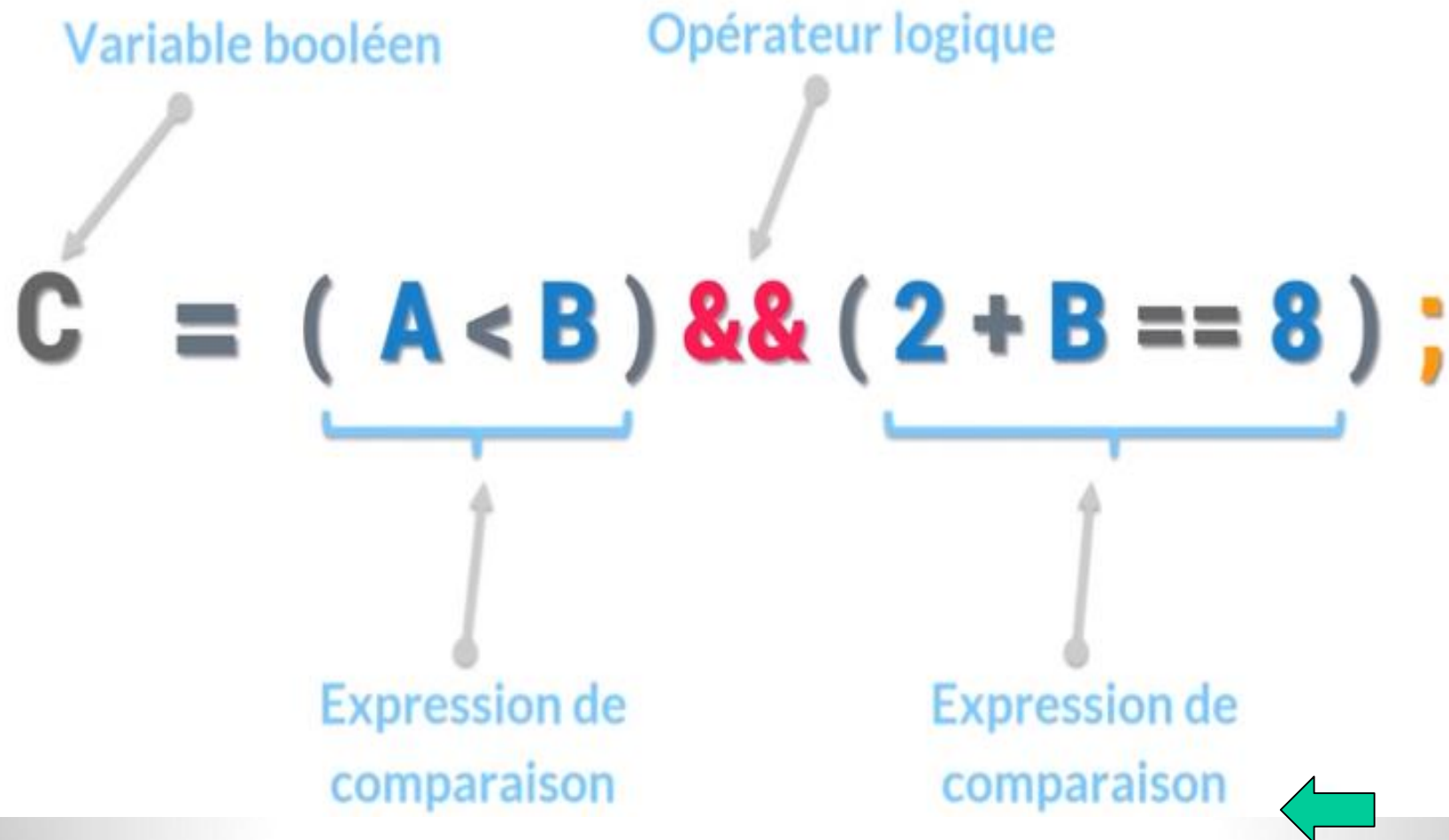
Expression  
arithmétique

# Expression de comparaison

Donner la valeur de la variable A (booléen) après chaque instruction:

Instruction	Résultat
$A = 3 < 5;$	
$A = 2 \neq 4;$	
$A = 1 > 8;$	
$A = (50 \% 3) == 2;$	
$A = (5 * 15) > \text{pow}(5, 5);$	
$A = (10 - 22) \leq (5 - 9 * 2);$	

# Expression Logique



# Expression Logique

Une expression logique est la composée d'expressions de comparaisons par les opérateurs logiques. Une expression logique donne un résultat booléen (vrai ou faux). Langage C dispose de trois opérateurs logiques : **et** (noté **&&**), **ou** (noté **||**) et **non** (noté **!**).

Exemple :

```
C = ( A < B ) && ( 2 + B == 8 );
```

Expression de  
comparaison

Opérateur  
logique


Expression de  
comparaison



# Expression Logique

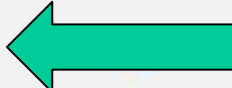
Le tableau suivant illustre les différentes valeurs de vérité que l'on obtient en combinant les valeurs de de deux variables booléennes A et B à l'aide des opérateurs logiques.

A	B	A && B	A    B	!A
False	False	False	Faux	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False



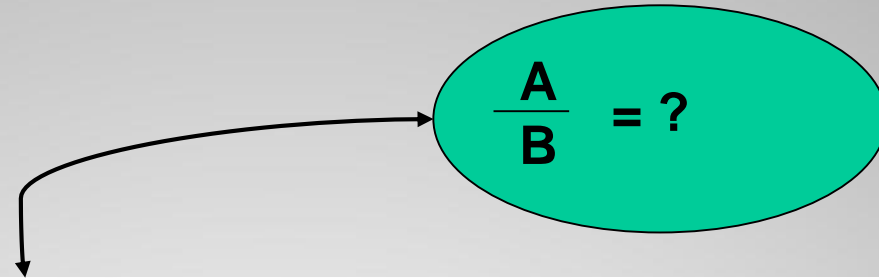
# Expression Logique

Quelle sera la valeur de chaque variable logique (C, D, E, F et G) dans chacun des cas suivants :

Instruction	Résultat
A = 5;	5
B = 6;	6
C = (A < B) && (3 > B);	
D = (A = 5)    (B > 10);	
E = ! C;	
F = C    (E && D);	
G = (! E && F)    (C && D);	



# Instructions conditionnelles



## 1 - Structure conditionnelle Simple (un choix)

Syntaxe :

```
if (Condition)
    Instruction ;
```

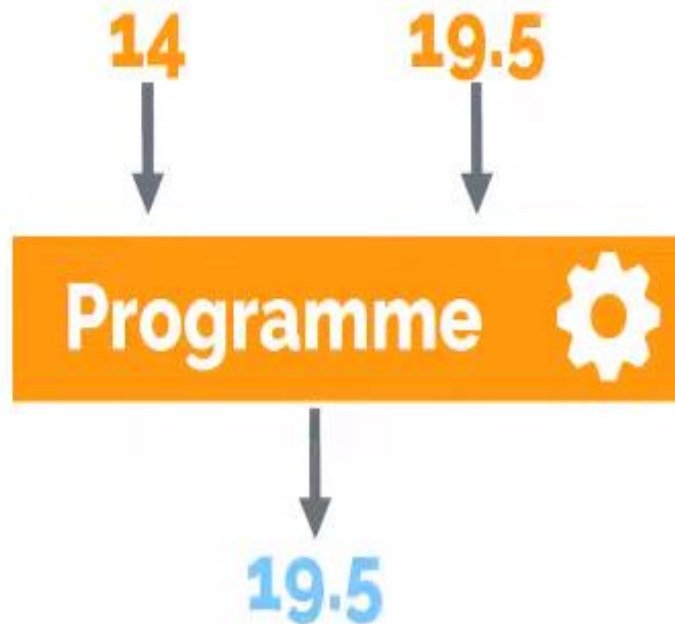
```
if (Condition)
{
    Instruction1 ;
    Instruction2 ;
    ...
}
```

Si la condition vaut Vrai alors le bloc d'instructions sera exécuté, si non il sera ignoré.

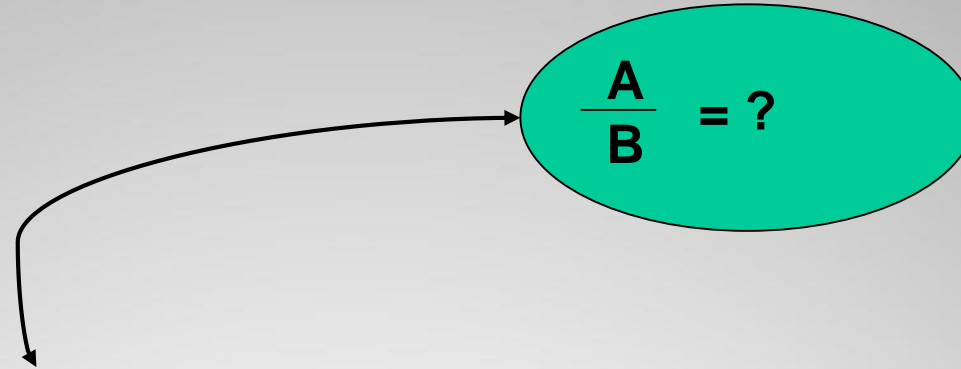


# Instructions conditionnelles

Ecrire un programme qui permet de calculer le maximum de deux nombres réels saisis par l'utilisateur.



# Instructions conditionnelles



## 2 - Structure alternative (deux choix)

Syntaxe :

```
if (Condition)
    Instruction1 ;
else
    Instruction2 ;
```

```
if (Condition) {
    Instruction1 ;
    ...
}
else {
    Instruction2 ;
    ...
}
```

## 3 - Structure imbriquée (multiple choix)

Syntaxe :

```
if (Condition1) {  
    Instructions ;  
}  
else if (Condition2) {  
    Instructions ;  
}  
else if (Condition3) {  
    ...  
}  
else {  
    Instructions ;  
}
```

# Instructions conditionnelles

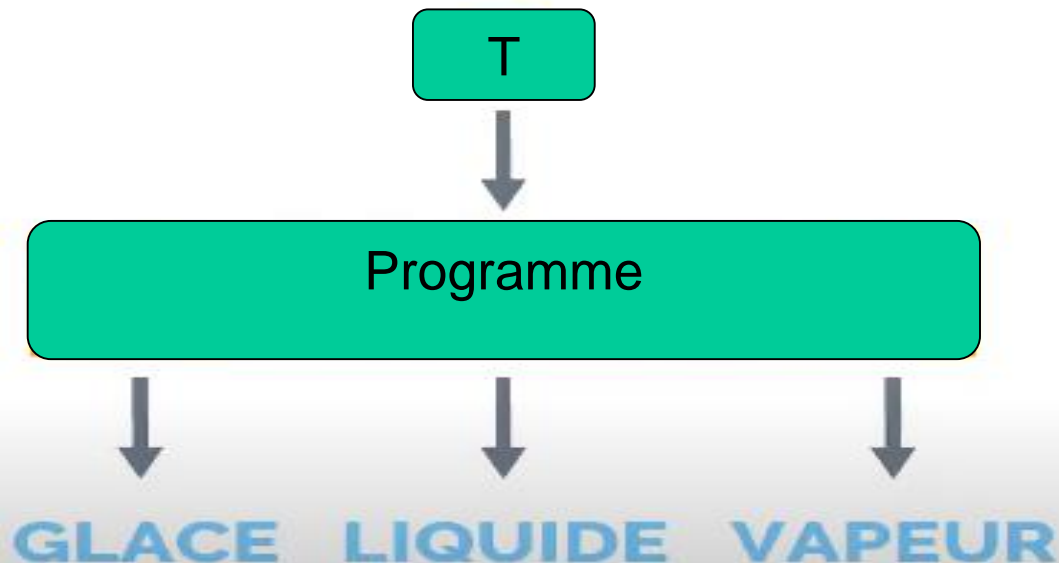
```
#include <stdio.h>
```

```
int main(){  
    int A,B;  
    float C;  
    printf("Veuillez entrer la valeur de A:");  
    scanf("%d",&A);  
    printf("Veuillez entrer la valeur de B:");  
    scanf("%d",&B);  
    if (B!=0){  
        C=A/B;  
        printf ("Le resultat est: %.2f",C);  
    }  
    else{  
        printf ("Erreur");  
    }  
  
    return 0;  
}
```

# Exercice: Instructions conditionnelles

Ecrire un programme qui permet de lire la valeur de la température de l'eau et d'afficher son état :

- **GLACE** Si la température est inférieure à 0.
- **LIQUIDE** Si la température est strictement supérieure à 0 et strictement inférieure à 100.
- **VAPEUR** Si la température est supérieure à 100.



# Instructions conditionnelles

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

```
int main(){  
    float T;  
    printf("Veuillez entrer la valeur de T:");  
    scanf("%d",&T);  
    if(T<=0){  
        printf("GLASE");  
    }  
    else if(T>0 && T<100){  
        printf("LIQUIDE");  
    }  
    else if (T>=100){  
        printf("LIQUIDE");  
    }  
  
    return 0;  
}
```



## 4 - Structure à choix multiple : switch

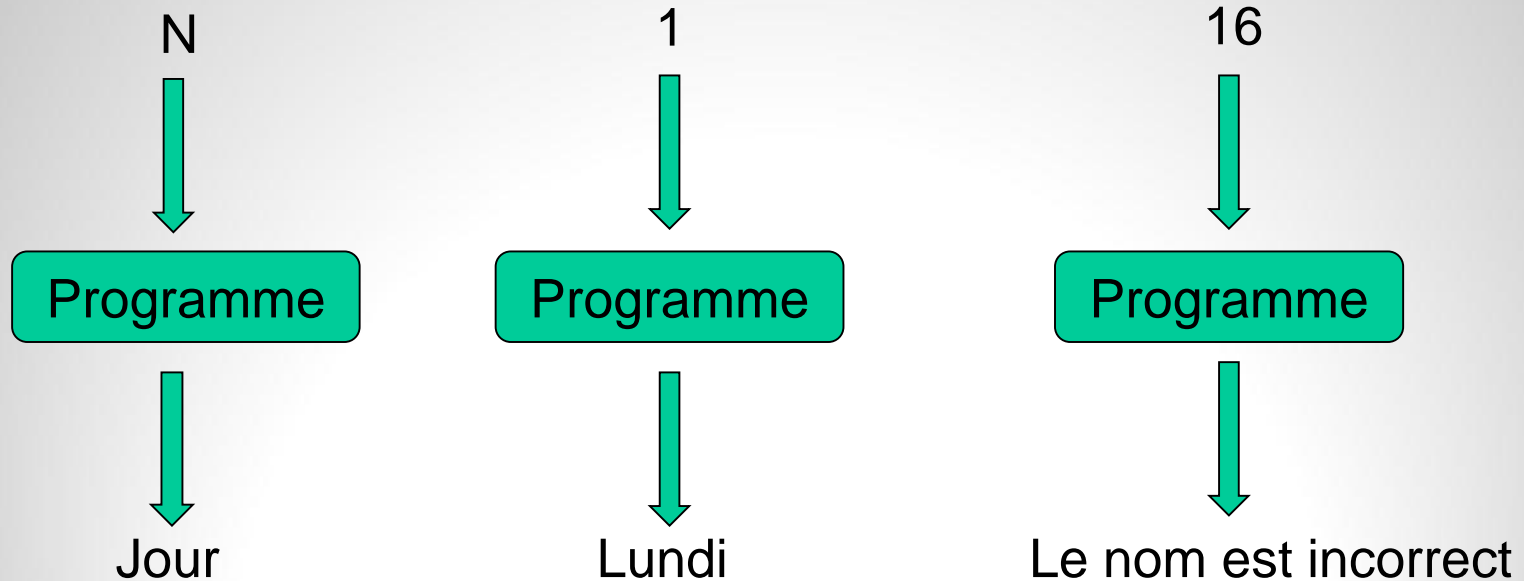
Lorsque l'imbrication des alternatives devient importante, l'utilisation de la structure à choix multiple devient nécessaire.

```
switch (Expression) {  
    case valeur1: Instruction1;  
                break;  
    case valeur2: Instruction2;  
                break;  
    ...      ...      :      ...  
    case valeurN: InstructionN;  
                break;  
    default: AutreInstruction;  
            Break;  
}
```



## Exercice: Instructions conditionnelles

Ecrire un programme qui permet de demander à utilisateur de saisir un entier entre 1 et 7 au clavier, et afficher le nom du jour correspondant.



# Instructions conditionnelles

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

```
int main(){  
    int N;  
    printf("Veuillez entrer un nombre entre 1 et 7:");  
    scanf("%d",&N);  
    switch(N){  
    case 1: printf("Lundi");  
        break;  
    case 2: printf("Mardi");  
        break;  
    case 3: printf("Mercredi");  
        break;  
        ...  
        ...  
        ...  
        ...
```

```
    case 4: printf("Jeudi");  
        break;  
    case 5:  
    printf("Vendredi");  
        break;  
    case 6:  
    printf("Samedi");  
        break;  
    case 7:  
    printf("Dimanche");  
        break;  
    default : printf("Erreur");  
        break;  
    }  
    return 0;  
}
```

# Structures répétitives

## *Table de multiplication de 7*

```
#include <stdio.h >
int main () {
    int M;
    M = 7 * 0;
    printf ( " 7 x 0 = %d \n " , M );
    M = 7 * 1;
    printf ( " 7 x 1 = %d \n " , M );
    M = 7 * 2;
    printf ( " 7 x 2 = %d \n " , M );
    M = 7 * 3;
    printf ( " 7 x 3 = %d \n " , M );
    M = 7 * 4;
    printf ( " 7 x 4 = %d \n " , M );
```

```
    M = 7 * 5;
    printf ( " 7 x 5 = %d \n " , M );
    M = 7 * 6;
    printf ( " 7 x 6 = %d \n " , M );
    M = 7 * 7;
    printf ( " 7 x 7 = %d \n " , M );
    M = 7 * 8;
    printf ( " 7 x 8 = %d \n " , M );
    M = 7 * 9;
    printf ( " 7 x 9 = %d \n " , M );
    M = 7 * 10;
    printf ( " 7 x 10 = %d \n " , M );
    return 0; }
```

# Structures répétitives

La **structure répétitive (Boucle)** permet d'exécuter plusieurs fois une séquence d'instructions.

Dans une boucle, le nombre de répétitions peut être connu, **fixé à l'avance**, comme il peut **dépendre d'une condition** permettant l'arrêt et la sortie de cette boucle.

Boucle **for** ←

Le nombre de répétitions peut être connu

Boucle **while**

Le nombre de répétitions dépend d'une condition

Boucle **do ... while ...**

# Structures répétitives: *La boucle for*

## 1 – La boucle for

Cette boucle permet d'exécuter une séquence d'instructions un nombre de fois connu fixé à l'avance.

Syntaxe :

```
for ( Initialisation ; Condition ; Incrémentation ) {  
    Instructions à répéter ;  
}
```

**Initialisation** : est une instruction (ou un bloc d'instructions) exécutée avant le premier tour de la boucle.

**Condition** : Tant que cette condition est vraie, la boucle for continue.

**Incrémentation** : cette instruction est exécutée à la fin de chaque tour de boucle pour mettre à jour la variable.

# Structures répétitives: *La boucle for*

Exemple : la boucle suivante affiche le message « Hello World! » 5 fois.

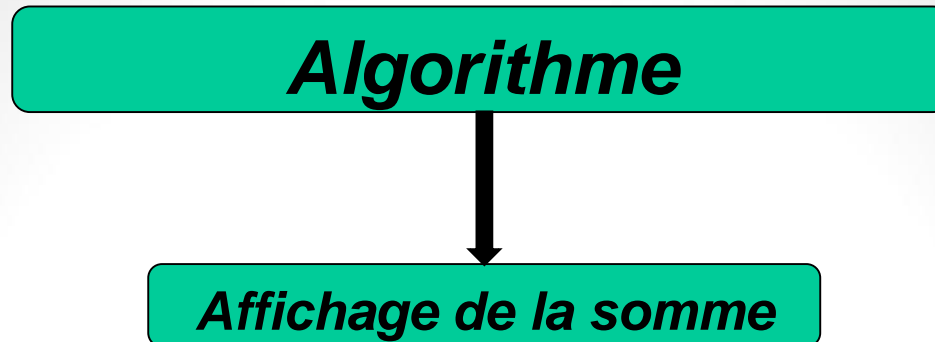
```
for ( i = 0 ; i < 5 ; i ++ ) {  
    printf ( " Hello World ! " ) ;  
}
```



## Structures répétitives: *La boucle for*

Ecrire un algorithme qui permet de calculer la somme des 20 premiers entiers positifs.

*La somme des 20 premiers entiers positifs est: 210*





# Structures répétitives: *La boucle for*

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

```
int main(){
```

```
    int i,S;
```

```
    S=0;
```

```
    for (i=1;i<=20;i++) {
```

```
        S=S+i;
```

```
    }
```

```
        printf ("la somme des 20 premiers entiers positifs est: %d",S);
```

```
    return 0;
```

```
}
```

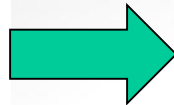
# Structures répétitives: La boucle *while*

$0 * X = ?$

$1 * X = ?$

$2 * X = ?$

$3 * X = ?$



## Table de multiplication

Veillez saisir un nombre : -3

Veillez saisir un nombre : 13

Veillez saisir un nombre : 3

$0 \times 3 = 0$

$6 \times 3 = 18$

$1 \times 3 = 3$

$7 \times 3 = 21$

$2 \times 3 = 6$

$8 \times 3 = 24$

$3 \times 3 = 9$

$9 \times 3 = 27$

$4 \times 3 = 12$

$10 \times 3 = 30$

$5 \times 3 = 15$

## 2 – La boucle while

Cette boucle permet de répéter un bloc d'instructions tant qu'une condition est vraie.

Syntaxe :

```
while ( Condition ) {  
    Instructions à répéter ;  
}
```

Remarque : La vérification de la condition s'effectue avant l'exécution des instructions. Celles-ci peuvent donc ne jamais être exécutées.



# Structures répétitives: *La boucle while*

## Exercice: Table de multiplication

```
#include <stdio.h>
int main(){
    int i,N;
    printf("Veuillez entrer la valeur de N:");
    scanf("%d",&N);
    while (N<0 || N>=10){
        printf("Veuillez entrer la valeur de N:");
        scanf("%d",&N);
    }
    /*    for(i=0;i<=10;i++){
        printf("%d x %d = %d \n",i,N,i*N);
    }*/
    i=0;
    while(i<=10){
        printf ( "%d x %d = %d \n",i,N,i*N);
        i++;
    }
    return 0;
}
```

# Structures répétitives: La boucle *while*

Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus grand ! », et inversement, « Plus petit ! » si le nombre est inférieur à 10.



```
Entrez un nombre entre 10 et 20 : 6
Plus petit !
Entrez un nombre entre 10 et 20 : -2
Plus petit !
Entrez un nombre entre 10 et 20 : 29
Plus grand !
Entrez un nombre entre 10 et 20 : 15
Bravo ! vous avez tapé un nombre
compris entre 10 et 20
```

# Structures répétitives: *La boucle while*

```
#include <stdio.h>
int main(){
    int N;
    printf("Veuillez entrer un nombre compris entre 10 et 20 : ");
    scanf("%d",&N);
    while (N<10 || N>20){
        if (N<10){
            printf(" Plus petit !\n : ");
        }
        else
            printf(" Plus grand !\n : ");
        printf("Veuillez entrer une valeur entre 10 et 20 : ");
        scanf("%d",&N);
    }
    printf("Bravo ! vous avez tape un nombre compris entre 10 et 20: ");
    return 0;
}
```

## 3 – La boucle do ... while ...

Cette boucle permet de répéter un bloc d'instructions tant qu'une condition est vraie.

Syntaxe :

```
do {  
    Instructions à répéter ;  
} while ( Condition ) ;
```

Remarque : La vérification de la condition s'effectue après l'exécution des instructions. Celles-ci sont donc exécutées au moins une fois.





# Structures répétitives: *La boucle Do... while()*

## **Exercice: Table de multiplication**

```
#include <stdio.h>

int main(){
    int i,N;
do {
    printf("Veuillez entrer la valeur de N:");
    scanf("%d",&N);
} while (N<0 || N>=10);
    for(i=0;i<=10;i++){
        printf("%d x %d = %d \n",i,N,i*N);
    }
return 0;
}
```

# Structures répétitives: *La boucle Do... while()*

Ecrire un programme qui demande a l'utilisateur de saisir un nombre entier strictement supérieur à 1, et qui calcule la somme des entiers jusqu'à ce nombre.

Par exemple, si l'on entre 5, le programme doit calculer :

$$1 + 2 + 3 + 4 + 5$$



Entrez un nombre : -13

Entrez un nombre : 1

Entrez un nombre : 0

Entrez un nombre : 7

La somme est : 28

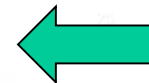
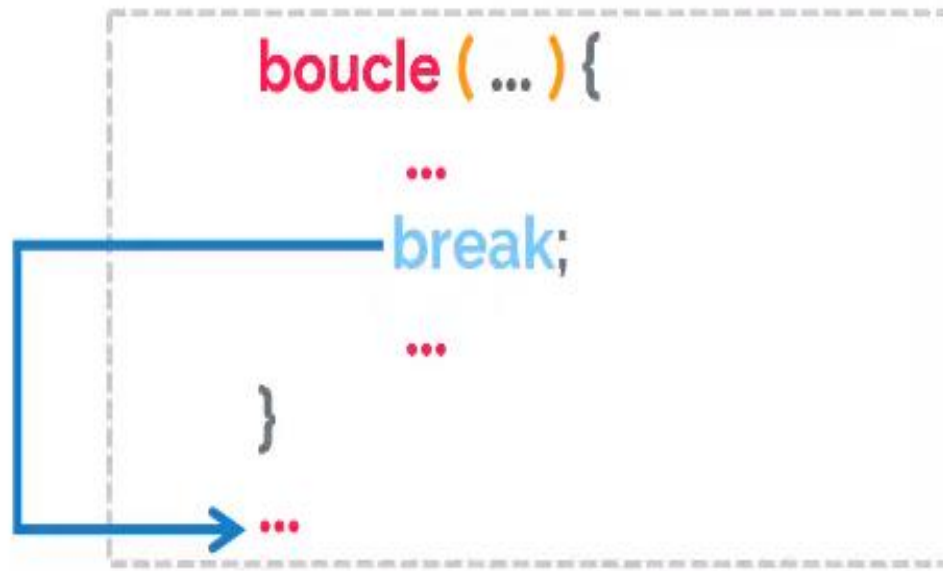
# Structures répétitives: *La boucle Do... while()*

```
#include <stdio.h >
int main () {
    int N , S , i ;
    do {
        printf ( " Veuillez entrer un nombre : " );
        scanf ( "%d" , &N );
    } while ( N < 1 );
    S = 0 ;
    for ( i = 1 ; i <= N ; i ++ ) {
        S = S + i ;
    }
    printf ( " La somme est : %d" , S );
    return 0 ;
}
```

## 4 – L'instruction break

L'instruction **break** permet d'arrêter le déroulement d'une boucle, et de passer à l'instruction qui suit cette boucle.

Syntaxe :



# Les instructions de contrôle de boucles: **break**

Ecrire un programme qui calcule la somme d'un maximum de 8 nombres entrés par l'utilisateur, si un nombre négatif est entré, la boucle se termine.



```
Entrez N1 :      6
Entrez N2 :     19
Entrez N3 :     15
Entrez N4 :      8
Entrez N5 :     -2
La somme est :   48
```

# Structures répétitives: *La boucle for*

```
#include <stdio.h>

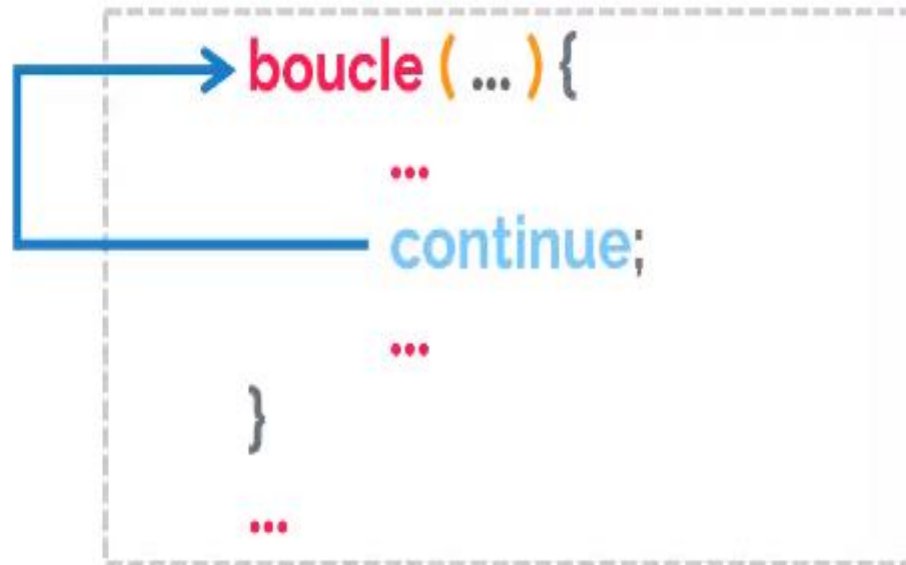
int main(){
int i,S,N;

S=0;
for(i=0;i<8;i++){
    printf("Veuillez entrer la valeur de N: ");
    scanf("%d",&N);
    if (N<0)
        break;
    S=S+N;
}
printf("La somme est: %d",S);
return 0;
}
```

## 5 – L'instruction continue

L'instruction **continue** permet d'ignorer l'itération actuelle de la boucle et de passer à l'itération suivante.

Syntaxe :





# Les instructions de contrôle de boucles: **continue**

Ecrire un programme qui calcule la somme d'un maximum de 8 nombres entrés par l'utilisateur, si un nombre négatif est entré, la boucle ignore ce nombre.



Entrez N1 :	2
Entrez N2 :	-5
Entrez N3 :	30
Entrez N4 :	-66
Entrez N5 :	-2
Entrez N6 :	15
Entrez N7 :	6
Entrez N8 :	-34
La somme est :	53

# Les instructions de contrôle de boucles: **continue**

```
#include <stdio.h>

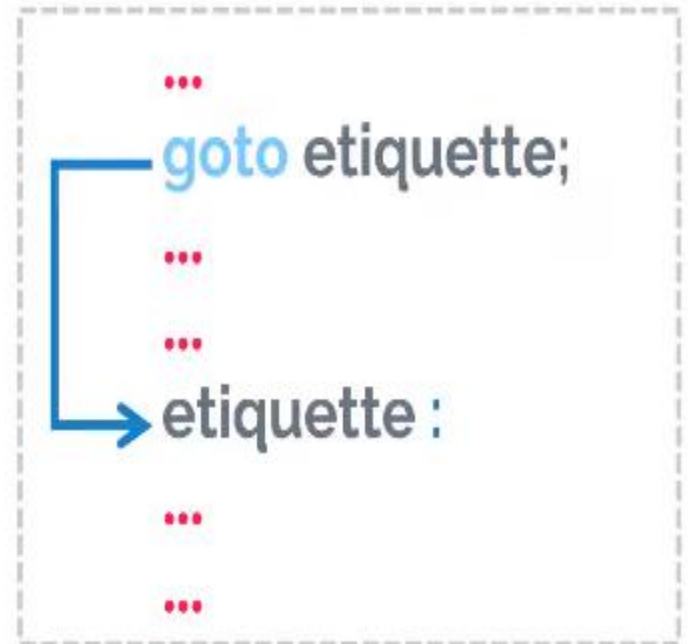
int main(){
int i,S,N;

S=0;
for(i=0;i<8;i++){
    printf("Veuillez entrer la valeur de N: ");
    scanf("%d",&N);
    if (N<0)
        continue;
    S=S+N;
}
printf("La somme est: %d",S);
return 0;
}
```

## 6 - L'instruction goto

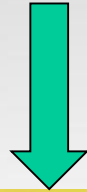
L'instruction **goto** permet de se repositionner sur une autre section de code à exécuter, introduite par une étiquette, au lieu de poursuivre une exécution séquentielle.

Syntaxe :



# Les instructions de contrôle de boucles: **goto**

Sans utiliser de boucles, écrivez un programme qui demande un nombre entre 1 et 5, jusqu'à ce que la réponse soit appropriée.



```
Entrez un nombre entre 1 et 5 : 9
Entrez un nombre entre 1 et 5 : -2
Entrez un nombre entre 1 et 5 : 29
Entrez un nombre entre 1 et 5 : 2
Bravo ! vous avez tapé un nombre
compris entre 1 et 5
```

# Les instructions de contrôle de boucles: **goto**

```
#include <stdio.h>

int main(){
int N;
    lieu:
    printf("Veuillez entrer un nombre compris entre 1 et 5: ");
    scanf("%d",&N);
    if (N<1 || N>5)
        goto lieu;
    printf("Bravo ! Votre reponce est appoprier !!!");
    return 0;
}
```

# Exercices

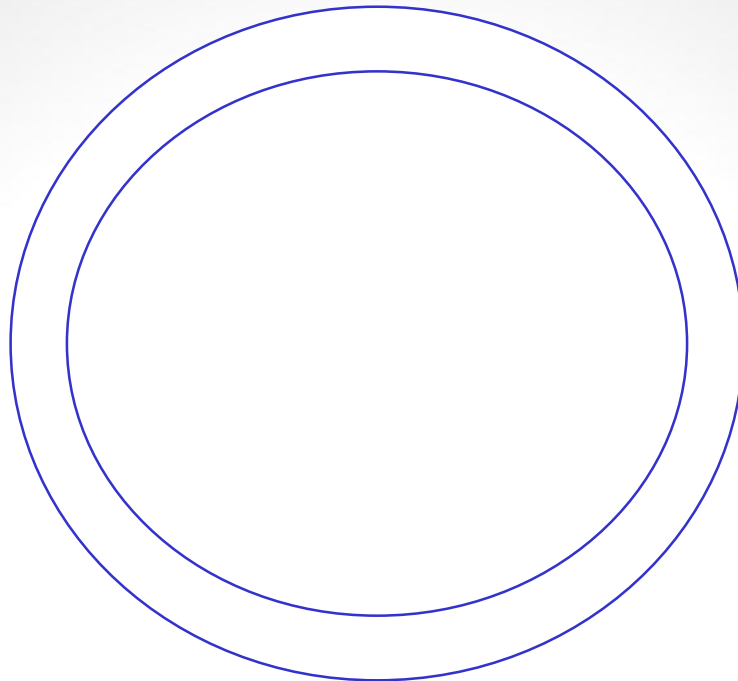
**Exercice 1:** Ecrire un programme C qui permet de lire deux nombres réels, et d'afficher ensuite leur produit, avec une précision de trois chiffres après la virgule.

**Exercice 2:** Ecrire un programme C qui lit en entrée trois entiers et affiche leur moyenne avec une précision de deux chiffres après la virgule.

# Exercices

**Exercice 3:** Ecrire un programme C qui lit deux réels  $R_1$  et  $R_2$  qui représentent les rayons de deux cercles concentriques, et renvoie ensuite l'aire de la surface comprise entre les deux cercles (surface grise).

**Remarque:**  $R_1$  peut être supérieur à  $R_2$ , comme il peut lui être inférieur.





# Exercices

**Exercice 4:** Ecrire un programme C qui permet de comparer deux entiers  $a$  et  $b$ , et d'afficher selon le cas l'un des messages suivants:  $a=b$ ,  $a>b$  ou  $a<b$ .

**Exercice 5:** Ecrire un programme C qui permet de dire si un entier est pair ou impair.

**Exercice 6:** Ecrire un programme C qui lit deux entiers représentant le grand rayon  $a$  et le petit rayon  $b$  d'une ellipse. Puis détermine si oui ou non l'aire de cette ellipse est supérieure ou égale à 100.

Remarque:  **$S(\text{ellipse}) = \text{Pi} * a * b$**

# Les Tableaux

Étudiants	Note
<i>Ali</i>	18
<i>Ahmed</i>	15.5
<i>mouna</i>	08
<i>mustapha</i>	17



Donner la note de l'étudiant num 1 : 10.5

Donner la note de l'étudiant num 2 : 8

Donner la note de l'étudiant num 3 : 13

Donner la note de l'étudiant num 4 : 14.5

```
#include <stdio.h >
int main () {
    float N1, N2, N3, N4 ;
    printf ( " Donner la note de l'étudiant num 1 : " );
    scanf ( "%f" , &N1);
    printf ( " Donner la note de l'étudiant num 2 : " );
    scanf ( "%f" , &N2);
    printf ( " Donner la note de l'étudiant num 3 : " );
    scanf ( "%f" , &N3);
    printf ( " Donner la note de l'étudiant num 4 : " );
    scanf ( "%f" , &N4);
    return 0;
}
```

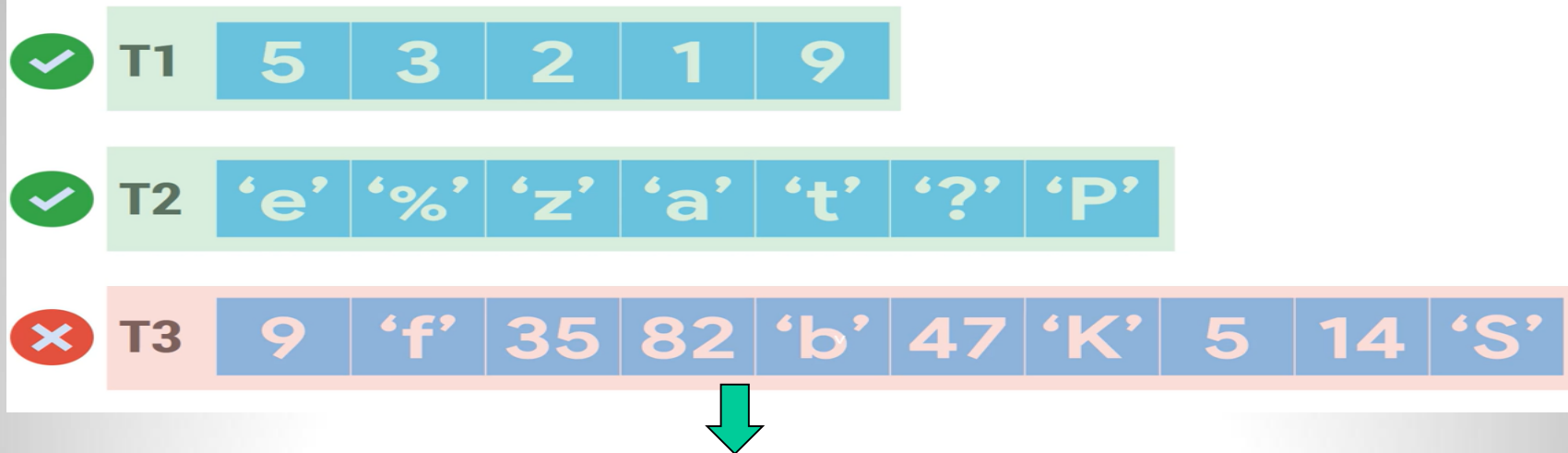
# Les Tableaux

```
#include <stdio.h >
int main () {
    float N1 , N2 , N3 , ... , N318 ;
    printf ( " Donner la note de l'étudiant num 1 : " );
    scanf ( "%f" , &N1);
    printf ( " Donner la note de l'étudiant num 2 : " );
    scanf ( "%f" , &N2);
    printf ( " Donner la note de l'étudiant num 3 : " );
    scanf ( "%f" , &N3);
    ... ..
    printf ( " Donner la note de l'étudiant num 318 : " );
    scanf ( "%f" , &N318);
    return 0 ;
}
```

# Les Tableaux

## *C'est quoi un tableau c*

Un tableau est une suite d'éléments de même type. Il utilise plusieurs cases mémoire à l'aide d'un seul nom. Comme toutes les cases portent le même nom, elles se différencient par un indice.



Déclarer un  
Tableau

Initialisation  
d'un tableau

Accéder à un  
Élément

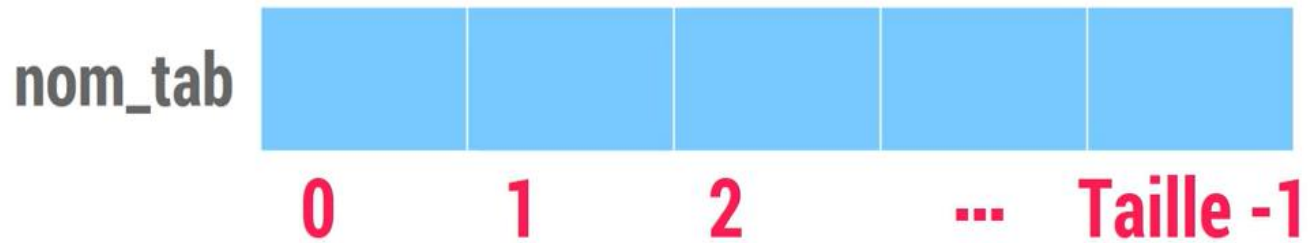
Afficher les  
Éléments

Remplir un  
Tableau

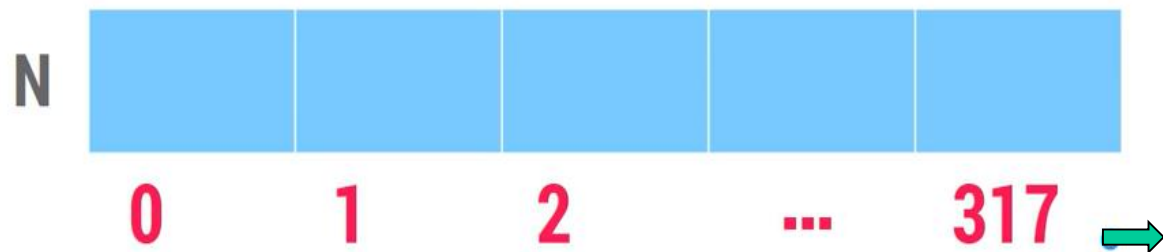
# Les Tableaux

## Déclaration d'un tableau

Syntaxe : `Type nom_tab [ Taille ] ;`



Déclaration de la variable N de programme note : `float N [ 318 ] ;`



# Les Tableaux

## *Déclaration et initialisation d'un tableau*

Syntaxe 1 : `Type nom_tab [ N ] = { val1 , val2 , ... , valn } ;`

Syntaxe 2 : `Type nom_tab [ ] = { val1 , val2 , ... , valn } ;`

Exemples : `int T1 [ 5 ] = { 3 , 18 , 47 , 8 , 362 } ;`

`int T2 [ 5 ] = {} ;`

`int T3 [ 5 ] = { 3 , 18 } ;`

`int T4 [ ] = { 3 , 18 , 47 , 8 , 362 } ;`

`int T5 [ 5 ] = { 3 , 18 , 47 , 8 , 362 , 9 } ;`



# Les Tableaux

Exemples :

```
int T1 [ 5 ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T2 [ 5 ] = { } ;
```

```
int T3 [ 5 ] = { 3 , 18 } ;
```

```
int T4 [ ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T5 [ 5 ] = { 3 , 18 , 47 , 8 , 362 , 9 } ;
```

T1

3	18	47	8	362
0	1	2	3	4



# Les Tableaux

Exemples :

```
int T1 [ 5 ] = { 3 , 18 , 47 , 8 , 362 } ;
```

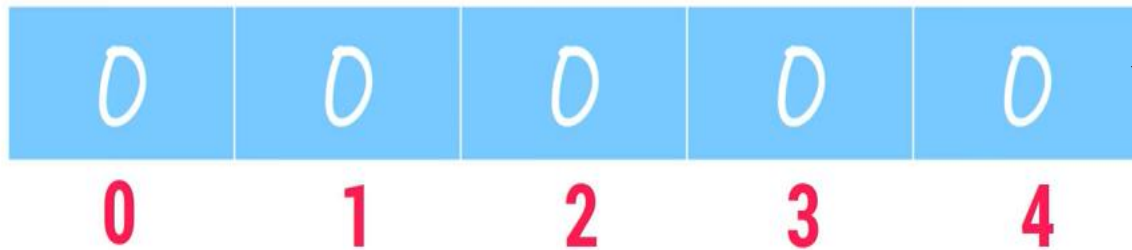
```
int T2 [ 5 ] = { } ;
```

```
int T3 [ 5 ] = { 3 , 18 } ;
```

```
int T4 [ ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T5 [ 5 ] = { 3 , 18 , 47 , 8 , 362 , 9 } ;
```

T2



# Les Tableaux

Exemples :

```
int T1 [ 5 ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T2 [ 5 ] = { } ;
```

```
int T3 [ 5 ] = { 3 , 18 } ;
```

```
int T4 [ ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T5 [ 5 ] = { 3 , 18 , 47 , 8 , 362 , 9 } ;
```

T3

3	18	0	0	0
0	1	2	3	4

# Les Tableaux

Exemples :

```
int T1 [ 5 ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T2 [ 5 ] = {} ;
```

```
int T3 [ 5 ] = { 3 , 18 } ;
```

```
int T4 [ ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T5 [ 5 ] = { 3 , 18 , 47 , 8 , 362 , 9 } ;
```

T4

3	18	47	8	362
0	1	2	3	4

# Les Tableaux

Exemples :

```
int T1 [ 5 ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T2 [ 5 ] = { } ;
```

```
int T3 [ 5 ] = { 3 , 18 } ;
```

```
int T4 [ ] = { 3 , 18 , 47 , 8 , 362 } ;
```

```
int T5 [ 5 ] = { 3 , 18 , 47 , 8 , 362 , 9 } ;
```



T5



# Les Tableaux

## *Accéder aux éléments d'un tableau*

Syntaxe d'affectation: `nom_tab [ indice ] = Valeur ;`

Syntaxe lecture: `scanf ("%...", & nom_tab [ indice ] );`

Syntaxe écriture: `printf ("%...", nom_tab [ indice ] );`

# Les Tableaux

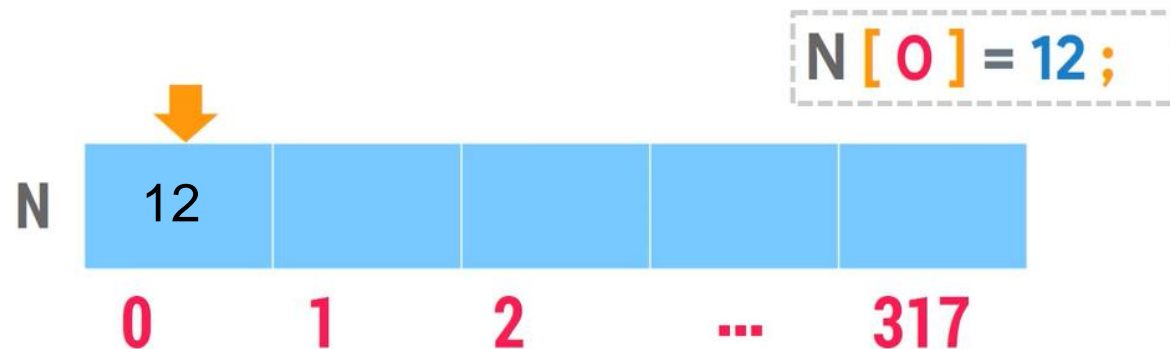
## Accéder aux éléments d'un tableau

Syntaxe d'affectation: `nom_tab [ indice ] = Valeur ;`

Syntaxe lecture: `scanf ("%..." , & nom_tab [ indice ] ) ;`

Syntaxe écriture: `printf ("%..." , nom_tab [ indice ] ) ;`

Affectation de la note 12 à l'étudiant num 1 :





# Les Tableaux

Syntaxe d'affectation: `nom_tab [ indice ] = Valeur ;`

Syntaxe lecture: `scanf ("%...", &nom_tab [ indice ] );`

Syntaxe écriture: `printf ("%...", nom_tab [ indice ] );`

Utilisation de la lecteur pour saisir la note de l'étudiant num 3 :

`scanf ("%f", &N [ 2 ] );`





# Les Tableaux

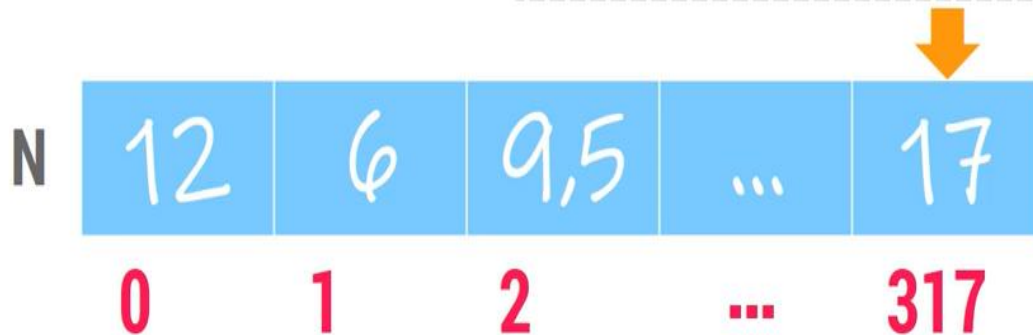
Syntaxe d'affectation: `nom_tab [ indice ] = Valeur ;`

Syntaxe lecture: `scanf ("%...", & nom_tab [ indice ] );`

Syntaxe écriture: `printf ("%...", nom_tab [ indice ] );`

Affichage de la note du dernier étudiant de la liste :

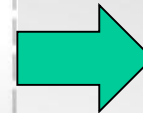
`printf ("%f", N [ 317 ] );`



# Les Tableaux

## Exemple

```
T[0] = 6;  
T[1] = T[0] - 4;  
T[2] = T[0] * T[1];  
T[3] = 20;  
T[4] = T[0] + T[1] + T[2] + T[3];
```



T[0]	
T[1]	
T[2]	
T[3]	
T[4]	

## *Remplir un tableau*

Syntaxe de lecteur : `scanf ("%..." , &nom_tab [ indice ] );`

**Exemple** : Remplissage de tous les éléments du tableau T avec l'instruction  
scanf :

```
for (i = 0; i < 5; i++) {  
    scanf ("%d" , &T [i]);  
}
```

# Les Tableaux

## *Afficher les éléments du tableau*

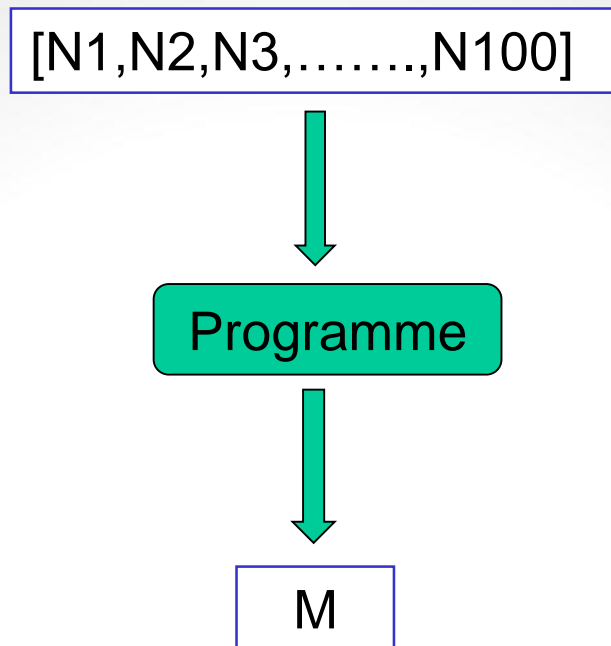
Syntaxe d'écriture : `printf ("%...", nom_tab [ indice ] );`

**Exemple** : Affichage des valeurs de tous les éléments du tableau T :

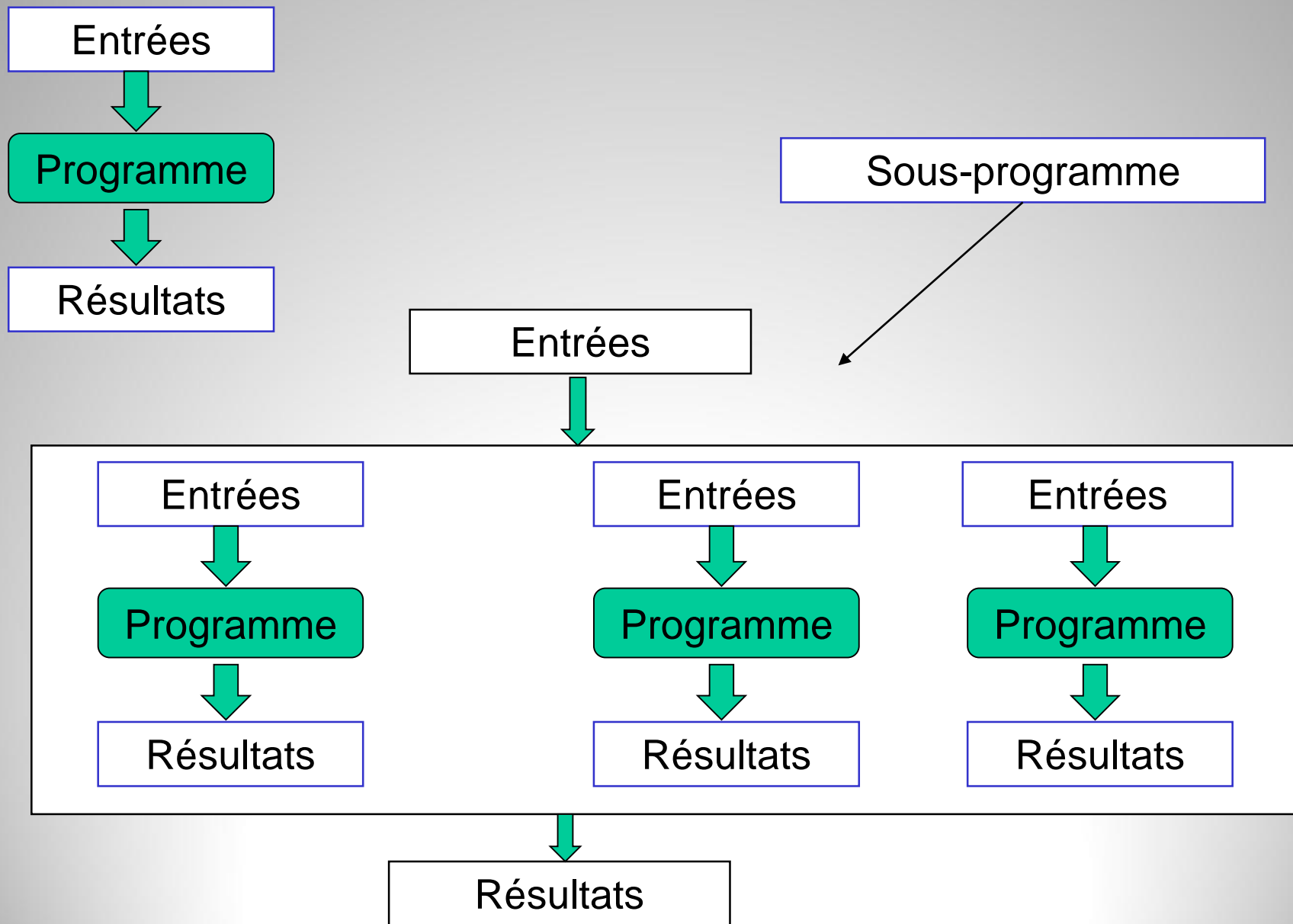
```
for ( i = 0 ; i < 5 ; i ++ ) {  
    printf ( "%d " , T [ i ] );  
}
```

## **EXERCICE**

Écrire un programme qui permet de demander à l'utilisateur de saisir les notes des étudiants (318 étudiants), puis le programme calcule et affiche la moyenne des notes.



# Les Fonctions



# Les Fonctions

Déclaration d'une fonction

Appel d'une fonction

1

Type de retour

2

Nom de fonction

3

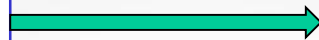
Arguments de fonction

5

Résultats Retourné

4

Traitements





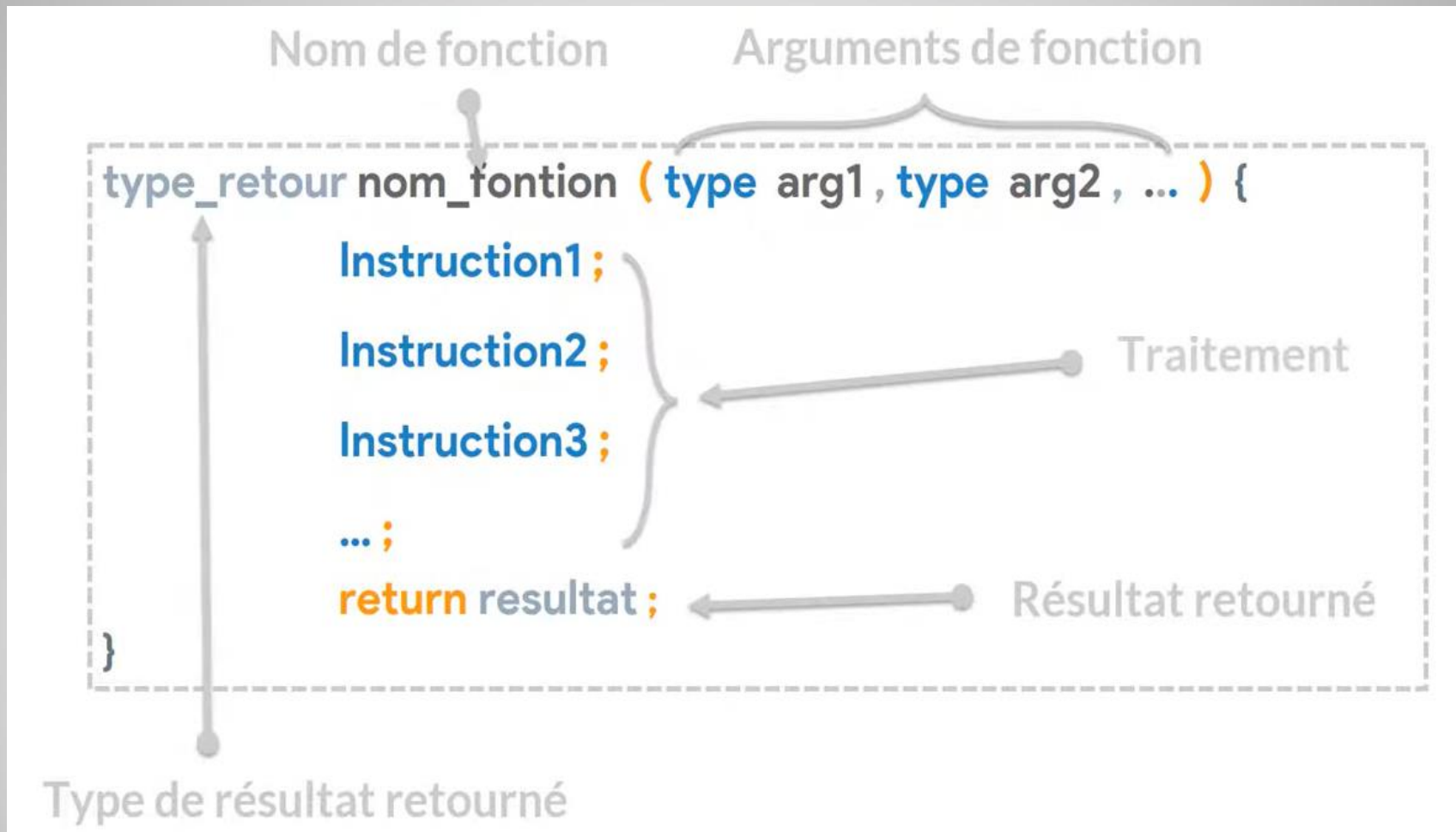
## *Définition*

Une fonction est une suite d'instructions regroupées sous un nom; elle prend en entrée des paramètres (arguments) et retourne un résultat.

Une fonction est écrite séparément du corps de programme principal (main) et sera appelée par celui-ci lorsque cela sera nécessaire.

# Les Fonctions

## Syntaxe



# Les Fonctions

## *Les types de fonctions*

### Pas de retour et pas d'arguments

```
void nom_fonction ( ) {  
    Instructions ;  
}
```

### Pas de retour et avec arguments

```
void nom_fonction ( type arg1 , ... ) {  
    Instructions ;  
}
```

### Avec retour et pas d'arguments

```
type_retour nom_fonction ( ) {  
    Instructions ;  
    return resultat ;  
}
```

### Avec retour et avec arguments

```
type_retour nom_fonction ( type arg1 , ... ) {  
    Instructions ;  
    return resultat ;  
}
```



# Les Fonctions

## *Les types de fonctions : Exemple*

### Pas de retour et avec arguments

```
void puissance ( int N ) {  
    int P ;  
    P = N * N ;  
    printf ( " La puissance de %d est : %d ", N , P ) ;  
}
```

### Avec retour et avec arguments

```
int puissance ( int N ) {  
    int P ;  
    P = N * N ;  
    return P ;  
}
```

# Les Fonctions

## *Appel de la fonctions*

### Pas de retour et pas d'arguments

```
nom_fun ();
```

### Pas de retour et avec arguments

```
nom_fun (arg1 , arg2 , ... );
```

### Avec retour et pas d'arguments

```
x = nom_fun ();
```

```
x = y / nom_fun ();
```

```
printf ( nom_func () );
```

### Avec retour et avec arguments

```
x = nom_fun (arg1 , arg2 , ... );
```

```
x = y / nom_fun (arg1 , arg2 , ... );
```

```
printf ( nom_func (arg1 , arg2 , ... ) );
```



# Les Fonctions

## Appel de la fonctions

```
#include <stdio.h>
void puissance ( int N ) {
    int P;
    P = N * N;
    printf ( " La puissance de %d est : %d ", N , P);
}
int main () {
    int N;
    printf ( "Veuillez saisir la valeur de N:" );
    scanf ( "%d" , &N );
    puissance (N);
    return 0;
}
```

Déclaration  
de la fonction



L'appel de  
la fonction





# Les Fonctions

## Appel de la fonctions

```
#include <stdio.h>
int puissance ( int N ) {
    int P ;
    P = N * N ;
    return P ;
}
int main () {
    int N , P ;
    printf ( "Veuillez saisir la valeur de N : " );
    scanf ( "%d" , &N );
    P = puissance ( N );
    printf ( " La puissance de %d est : %d " , N , P );
    return 0 ;
}
```

Déclaration  
de la fonction

L'appel de  
la fonction





# Les Fonctions

## Les variables locales

```
#include <stdio.h>
int puissance (int N) {
    int P;
    P = N * N;
    return P;
}
int main () {
    int N, P;
    printf ( "Veuillez saisir la valeur de N : " );
    scanf ( "%d" , &N );
    P = puissance (N);
    printf ( " La puissance de %d est : %d " , N , P );
    return 0;
}
```

Les variables P et N sont des variables locales. Elles sont décalés à l'intérieur de la fonction puissance et main.



# Les Fonctions

## Les variables globale

```
#include <stdio.h>
int N;
int puissance () {
    int P;
    P = N * N;
    return P;
}
int main () {
    int P;
    printf ( "Veuillez saisir la valeur de N : " );
    scanf ( "%d" , &N );
    P = puissance (N);
    printf ( " La puissance de %d est : %d " , N , P );
    return 0;
}
```

La variable N est une variable globale. Elle est alors disponible à la fonction puissance et main



## Exercice 1

Ecrire un programme qui demande à l'utilisateur de saisir les valeurs de deux variables A et B (locales). Ensuite, il permet de définir et d'appeler les fonctions suivantes :

- Une fonction qui retourne si les valeurs de A et B sont de même signe ou non. (*Une fonction sans valeur de retour et avec arguments*)
- Une fonction qui renvoie le minimum de A et B. (*Une fonction avec une valeur de retour et avec arguments*)
- Une fonction qui renvoie le maximum de A et B. (*Une fonction avec une valeur de retour et avec arguments*)



# Les Fonctions

## *Exercice 1*

**Veillez saisir la valeur de A : 31**

**Veillez saisir la valeur de B : 8**

**Les valeurs de A et B ont le même signe.**

**La valeur minimale est : 8**

**La valeur maximale est : 31**

# Les Exercices

## *Exercice 1*

Ecrire un programme C qui définit et utilise une fonction de prototype **int Somme(int,int)** qui prend en paramètres deux entiers et renvoie leur somme.