

# PROGRAMMATION EN LANGAGE C

**Pr. MOHAMMED BOUTALLINE**

Ecole Nationale des Sciences Appliquées – Béni Mellal

Téléphone : +2125 23 48 02 18

Email : [boutalline@gmail.com](mailto:boutalline@gmail.com)

Site web : [boutalline.wordpress.com](http://boutalline.wordpress.com)

# OBJECTIFS DU MODULE

Ce cours a pour objectif de procurer une connaissance moderne de la programmation afin qu'un étudiant puisse solutionner des problèmes reliés à sa spécialité.

De façon plus spécifique, ce cours devra permettre à l'étudiant de :

- Savoir la structure d'un programme en langage C (premier programme).
- Acquérir les notions de programmation de base;
- Acquérir une connaissance du langage C ;
- utiliser et implémenter des programmes en C.

# PLAN

Le module de la programmation en C est constitué des chapitres suivantes :

- Introduction
- Déclarations, Types, Expressions et affectations
- Structures Alternatives : if, else, switch
- Structures répétitives : for, while do-while
- Autres contrôles des programmes
- Tableaux 1D, 2D
- Pointeurs
- Fonctions
- Structures de données
- Fichiers

## CHAPITRE N° 1

# DÉCLARATIONS, TYPES, EXPRESSIONS ET AFFECTATIONS

# Aspect général de "C"

- Le langage C a été développé à l'origine dans les années 1970 par **Dennis Ritchie** de **Bell Telephone Laboratories**, Inc.
- C est un langage de programmation structuré de haut niveau à usage général. Les instructions de C sont constituées de termes très proches des expressions algébriques, constituées de certains mots-clés anglais tels que **if**, **else**, **for**, **do** et **while**
- C contient certaines fonctionnalités supplémentaires qui lui permettent d'être utilisé à un niveau inférieur, agissant comme un pont entre le langage machine et les langages de haut niveau.
- Cela permet à C d'être utilisé pour la programmation système ainsi que pour la programmation d'applications

# Le jeu de caractères de C

- Le langage C se compose de certains jeux de caractères, de chiffres et de symboles spéciaux. Le jeu de caractères de C se compose de tous les alphabets de la langue anglaise. C se compose de
  - ✓ Alphabets :  $a \rightarrow z, A \rightarrow Z$
  - ✓ Numérique :  $0, 1 \rightarrow 9$
  - ✓ Symboles spéciaux  $\{, \}, [, ], ?, +, -, *, /, \%, !, ;$  et d'autres
- Les mots formés à partir du jeu de caractères sont des éléments constitutifs de C. Les différents types de mots suivants sont utilisés en C
  - 1) Identifiants
  - 2) Mots clés
  - 3) Constantes
  - 4) Opérateurs
  - 5) Symboles de ponctuation
- Un programme 'C' se compose de deux types d'éléments, définis par l'utilisateur et définis par le système. Les **identifiants** ne sont rien d'autre qu'un nom donné à ces éléments.

# Identifiants

- Un identifiants est un mot utilisé par un programmeur pour nommer une variable, une fonction ou une étiquette.
- Les identifiants se composent de lettres et de chiffres s'il y a lieu, sauf que le premier caractère doit être une lettre.
- Les lettres majuscules et minuscules peuvent être utilisées

✓ **Validés** : Root, \_getchar, \_sin, x1, x2, x3, x\_1, If  
Name

✓ **Non Validés** : 324, short, price\$, My

Nommer une variable :

- Doit être un identifiant valide.
- Ne doit pas être un mot-clé
- Les noms sont sensibles à la casse.
- La bibliothèque utilise généralement des noms commençant par \_.
- Styles de dénomination: style majuscule et style de soulignement :

✓ **lowerLimit**      **lower\_limit**

✓ **incomeTax**

**income\_tax**

# Mots clés

- Les mots clés ne sont rien d'autre que des identifiants définis par le système.
- Les mots clés sont des mots réservés au langage.
- Ils ont une signification spécifique dans le langage et ne peuvent pas être utilisés par le programmeur comme noms de variables ou de constantes
- **C** est sensible à la casse, cela signifie que ceux-ci doivent être utilisés car il s'agit de 32 mots-clés dans la programmation **C**

---

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

---



# Variables

- Une variable n'est rien d'autre qu'un nom donné à une zone de stockage que nos programmes peuvent manipuler. Chaque variable en C a un type spécifique, qui détermine la taille et la disposition de la mémoire de la variable; la plage de valeurs qui peuvent être stockées dans cette mémoire.
- Le nom d'une variable peut être composé de lettres, de chiffres et du caractère de soulignement. Il doit commencer par une lettre ou un trait de soulignement. Les lettres majuscules et minuscules sont distinctes car C est sensible à la casse. Il existe les types de variables de base suivants -

<i>Type</i>	<i>Description</i>
<b>char</b>	Généralement un <b>seul octet</b> . C'est un type entier.
<b>int</b>	La taille entière la plus naturelle de la machine.
<b>float</b>	Une valeur à virgule flottante simple précision.
<b>double</b>	Une valeur à virgule flottante double précision.
<b>void</b>	Représente l'absence de type.

Une séquence de caractères entre guillemets sous la forme "...". Par exemple, "13" est une chaîne littérale et non le numéro 13. 'a' et "a" sont différents.

- Les entiers sont stockés dans différentes tailles. Ils peuvent être signés ou non signés.
- Exemple : Supposons qu'un entier soit représenté par un octet (8 bits). Le bit le plus à gauche est le bit de signe. Si le bit de signe est 0, le nombre est traité comme positif.

La suite 01001011 = 75 (décimal).

Le plus grand nombre positif est 01111111 =  $2^7 - 1 = 127$ .

- Les nombres négatifs sont stockés comme complément à deux ou comme complément à un.
  - 75 = 10110100 (*un complément*).
  - 75 = 10110101 (*complément à deux*).

## Types intégraux

- **char** : Stockée en 8 bits:
  - ✓ Non signé de 0 à 255.
  - ✓ Signé de -128 à 127.
- **short int**: Stockée en 16 bits. :
  - ✓ Non signé 0 à 65535.
  - ✓ Signé -32768 à 32767.
- **int** : Identique à un **int court** ou **long**.
- **long int** : Stocké en 32 bits. :
  - ✓ Non signé 0 à 4294967295.
  - ✓ Signé -2147483648 à 2147483647

## Nombres à virgule flottante

- Les nombres à virgule flottante sont des nombres rationnels. Numéros toujours signés.
- **float** Précision approximative de 6 chiffres décimaux.
- Généralement stocké sur 4 octets avec 23 bits de mantisse signée et 8 bits d'exposant et un bit de signe.
- **double précision** approximative de 14 chiffres décimaux.
- Généralement stocké sur 8 octets avec 52 bits de mantisse et 11 bits d'exposant et un bit de signe.
- Il faut vérifier dans le fichier **limits.h** ce qui est implémenté sur une machine particulière.

# Variables

## Déclaration d'une variable

- Chaque variable utilisée doit être déclarée.
- La syntaxe de l'instruction de déclaration est :

```
data-type var1, var2, ...;
```

- La déclaration annonce le type de données d'une variable et alloue un emplacement mémoire approprié. Aucune valeur initiale (comme 0 pour les entiers) ne doit être supposée.
- Il est possible d'attribuer une valeur initiale à une variable dans la déclaration elle-même

```
data-type var = expression;
```

- Exemples :

```
int sum = 0;
```

```
char newLine = '\n';
```

```
float epsilon = 1.0e-6;
```

# Constants

- Une constante est une valeur ou un identifiant dont la valeur ne peut pas être modifiée dans un programme. Par exemple: **1**, **2.5**
- Comme mentionné, un identifiant peut également être défini comme une constante. par exemple. **const** double **PI** = **3.14**
- Ici, **PI** est une constante. Fondamentalement, cela signifie que **PI** et **3.14** sont les mêmes pour ce programme.

## Constantes entières

- Une constante entière est une constante numérique (associée à un nombre) sans aucune partie fractionnaire ou exponentielle. Il existe trois types de constantes entières en programmation C:
  1. Constante décimale (base 10)
  2. Constante octale (base 8)
  3. Constante hexadécimale (base 16)

# Constants

## Floating-point constants

- Une constante à virgule flottante est une constante numérique qui a une forme fractionnaire ou une forme exposant. Par exemple: 2.0      0.0000234      -0.22E-5

## Constantes de caractère

- Une constante de caractère est une constante qui utilise des guillemets simples autour des caractères. Par exemple: 'a'      'l'      '@'      'F'

## Constantes de chaîne


- Les constantes de chaîne sont les constantes qui sont entourées d'une paire de guillemets doubles. Par exemple: "bien"      "x"      "La terre est ronde \n"

# Constantes du caractère et de chaînes

- 'c', un seul caractère entre guillemets simples est stocké sous forme de char..  
Certains caractères spéciaux sont représentés par deux caractères entre guillemets simples.  
'\n' = nouvelle ligne, '\t' = tab, '\\ ' = Tabulation, '\"' = guillemets doubles.  
Les constantes **char** peuvent également être écrites en fonction de leur code ASCII.  
'\060' = '0' (code décimal est 48).
- Une séquence de caractères entre guillemets est appelée constante de chaîne ou littéral de chaîne. Par exemple :  
"Boutalline"  
"A"  
"3/9"  
"x = 5"

# Espaces blancs( White Spaces)

- For example: \n is used for newline. The backslash ( \ ) causes "escape" from the normal way the characters are interpreted by the compiler.Escape
- Par exemple: \n est utilisé pour la nouvelle ligne. La barre oblique inverse ( \ ) provoque un "échappement" de la manière normale dont les caractères sont interprétés par le compilateur.

Séquence	caractère
\b	Retour arrière
\t	Tabulation horizontale
\n	Nouvelle ligne
\r	Début de la ligne 
\\	Barre oblique inverse
\'	Guillemet simple
\"	Guillemet double
\?	Point d'interrogation
\0	caractère NULL



# Opérateurs en C

- Un opérateur est un symbole qui opère sur une valeur ou une variable. Par exemple: + est un opérateur pour effectuer une addition.
- La programmation en langage C dispose d'un large éventail d'opérateurs pour effectuer diverses opérations. Pour une meilleure compréhension des opérateurs, ces opérateurs peuvent être classés comme suit:
  - ✓ **Opérateurs arithmétiques**
  - ✓ **Opérateurs d'incrément et de décrémentation**
  - ✓ **Opérateurs d'affectation**
  - ✓ **Opérateurs relationnels**
  - ✓ **Opérateurs logiques**
  - ✓ **Opérateurs conditionnels**
  - ✓ **Opérateurs au niveau du bit**
  - ✓ **Opérateurs spéciaux**

# Opérateurs arithmétiques

Opérateur	Description
+	addition ou unaire plus
-	soustraction ou moins unaire
*	multiplication
/	division
%	reste après division (division modulo)

- **%** est un opérateur de modulo.  $x \% y$  donne le reste lorsque  $x$  est divisé par  $y$  et vaut zéro lorsque  $x$  est divisible par  $y$ .
- Ne peut pas être appliqué aux variables flottantes ou doubles.

- Exemple

```
if ( num % 2 == 0 )  
    printf("%d is an even number\n", num);  
else  
    printf("%d is an odd number\n", num);
```

**+** et **-** ont la même priorité et associent de gauche à droite.

$3 - 5 + 7 = ( 3 - 5 ) + 7 \neq 3 - ( 5 + 7 )$

$3 + 7 - 5 + 2 = ( ( 3 + 7 ) - 5 ) + 2$

**\***, **/**, **%** ont la même priorité et associent de gauche à droite.  
le groupe **+**, **-** a une priorité inférieure à celle du groupe **\***, **/** et **%**.

$3 - 5 * 7 / 8 + 6 / 2$

$3 - 35 / 8 + 6 / 2$

$3 - 4.375 + 6 / 2$

$3 - 4.375 + 3$

$-1.375 + 3$

$1.625$



# Opérateurs d'incrémentation et de décrémentation

- La programmation C a deux opérateurs incrément **++** et décrément **--** pour changer la valeur d'un opérande de 1.
- **++** augmente la valeur de 1 tandis que **--** diminue la valeur de 1.
- Ces deux opérateurs sont des opérateurs unaires, c'est-à-dire qu'ils n'opèrent que sur un seul opérande.

## Exemple 1 :

```
int a=10, b=100 ;  
++a = 11  
--b = 99  
a++ et ++a sont équivalents à a += 1.  
a-- et --a sont équivalents à a -= 1.
```

## Exemple 2 :

```
soit b = 10 alors  
(++b)+b+b = 33  
b+(++b)+b = 33  
b+b+(++b) = 31  
b+b*(++b) = 132
```

*Attention à la priorité*

# Opérateurs relationnels en C

- Un opérateur relationnel vérifie la relation entre deux opérandes. Si la relation est vraie, elle renvoie 1; si la relation est fausse, elle renvoie la valeur 0.
- Les opérateurs relationnels sont utilisés dans la prise de décision et les boucles:

Opérateur	Signification	Exemple
==	Égal à	5 == 3 renvoie 0
>	Supérieur à	5 > 3 renvoie 1
<	Inférieur à	5 < 3 renvoie 0
!=	Différent de	5 != 3 renvoie 1
>=	Supérieur ou égal	5 >= 3 renvoie 1
<=	Inférieur ou égal	5 <= 3 renvoie 0

- Prenons la valeur 1 (int) si la relation est vraie et 0 (int) si la relation est fausse.
- Exemple :

```
int a = 25, b = 30, c, d;
```

```
c = a < b;
```

```
d = a > b;
```

*la valeur de c sera 1 et celle de d sera 0.*

# Opérateurs logiques

- `&&`, `||` et `!` sont les trois opérateurs logiques.
- `expr1 && expr2` a une valeur 1 si `expr1` et `expr2` sont toutes deux différentes de zéro.
- `expr1 || expr2` a une valeur 0 si `expr1` et `expr2` sont toutes deux égaux de zéro.
- `! expr1` a une valeur 1 si `expr1` est zéro sinon 0.

Exemples :

- ```
if (notes >= 40 && presence >= 75)
    mention= "Bien";
```
- ```
Si (notes <40 || presence <75)
    mention= "Passable";
```

# Opérateurs d'affectation

- La forme générale d'un opérateur d'affectation est

**v op= expression ;**

- Where v is a variable and op is a binary arithmetic operator. This statement is equivalent to
- Où **v** est une variable et **op** est un opérateur arithmétique binaire. Cette déclaration équivaut à

**v = v op (expression) ;**

a = a + b	peut être écrit comme	a += b
a = a * b	peut être écrit comme	a *= b
a = a / b	peut être écrit comme	a /= b
a = a - b	peut être écrit comme	a -= b

# Variables globales et locales

## Variables globales

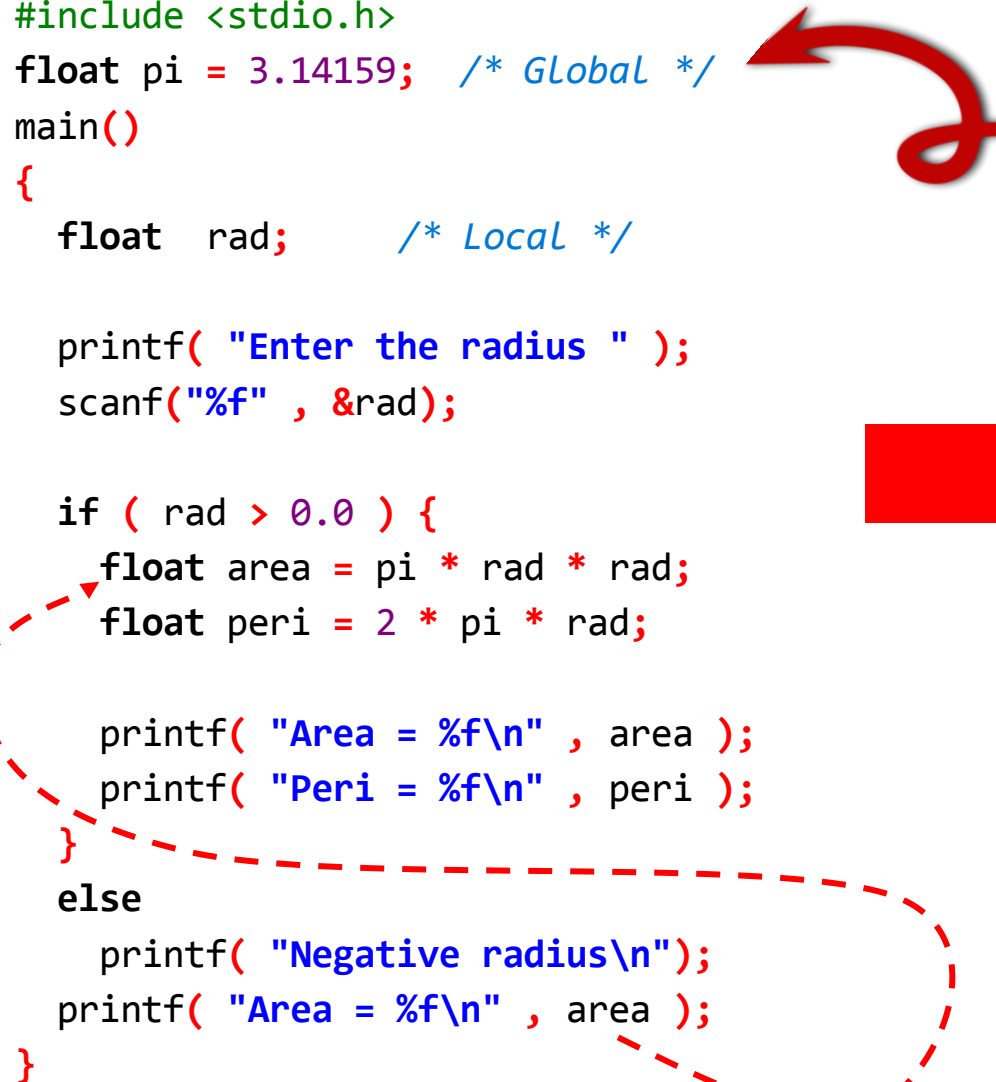
- Ces variables sont déclarées en dehors de toutes les fonctions.
- La durée de vie d'une variable globale correspond à toute la période d'exécution du programme.
- Accessible par toute fonction définie sous la déclaration, dans un fichier.

```
/* Compute Area and Perimeter of a circle */
#include <stdio.h>
float pi = 3.14159; /* Global */
main()
{
    float rad;      /* Local */

    printf( "Enter the radius " );
    scanf("%f" , &rad);

    if ( rad > 0.0 ) {
        float area = pi * rad * rad;
        float peri = 2 * pi * rad;

        printf( "Area = %f\n" , area );
        printf( "Peri = %f\n" , peri );
    }
    else
        printf( "Negative radius\n");
    printf( "Area = %f\n" , area );
}
```



Testez le code

# Global and Local Variables

## Local Variables

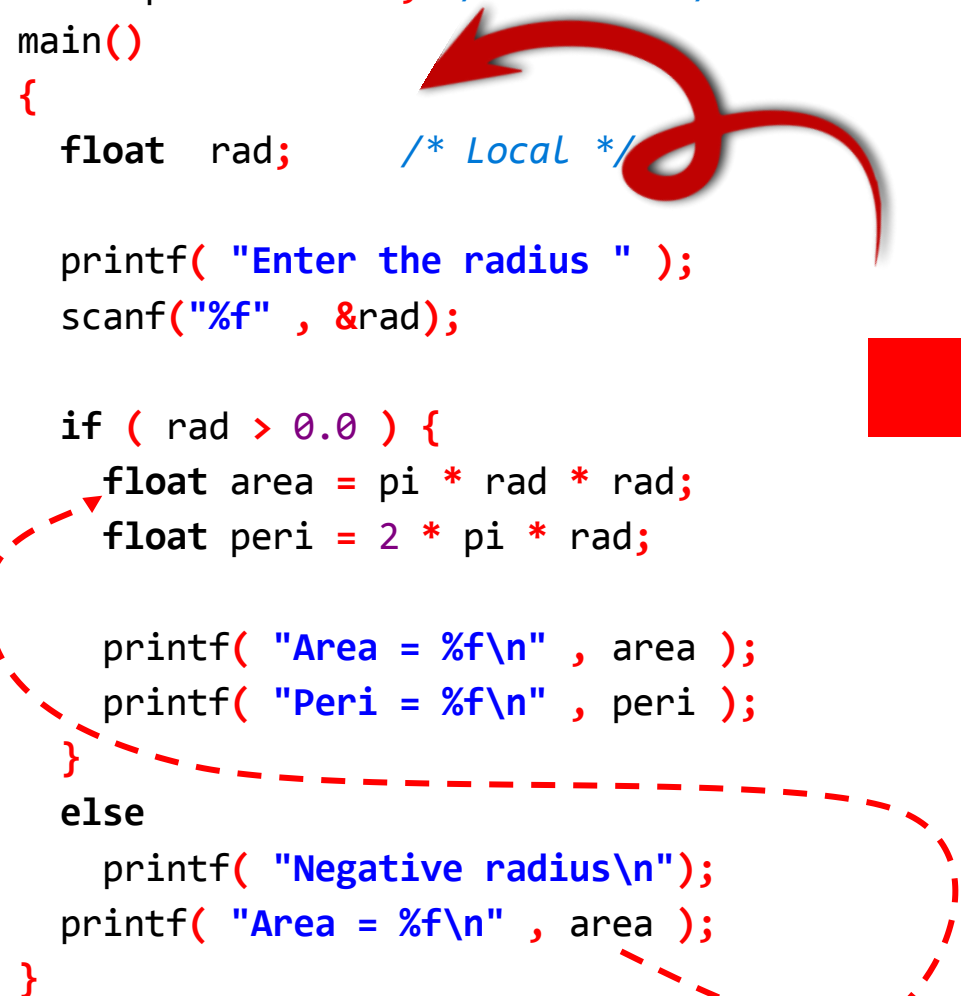
- Ces variables sont déclarées dans certaines fonctions.
- La durée de vie d'une variable locale correspond à toute la période d'exécution de la fonction dans laquelle elle est définie.
- Ne peut être accédé par aucune autre fonction.
- En général, les variables déclarées à l'intérieur d'un bloc ne sont accessibles que dans ce bloc.

```
/* Compute Area and Perimeter of a circle */
#include <stdio.h>
float pi = 3.14159; /* Global */
main()
{
    float rad; /* Local */

    printf( "Enter the radius " );
    scanf("%f" , &rad);

    if ( rad > 0.0 ) {
        float area = pi * rad * rad;
        float peri = 2 * pi * rad;

        printf( "Area = %f\n" , area );
        printf( "Peri = %f\n" , peri );
    }
    else
        printf( "Negative radius\n" );
    printf( "Area = %f\n" , area );
}
```



*Testez le code*



# Conversions de type

- Les opérandes d'un opérateur binaire doivent avoir le même type et le résultat est également du même type.
- **Division entière:** `c = (9 / 5)*(f - 32) ;`
- Les opérandes de la division sont à la fois **int** et donc le résultat serait également **int**. Pour des résultats corrects, on peut écrire  
`c = (9.0 / 5.0)*(f - 32) ;`
- Dans le cas où les deux opérandes d'un opérateur binaire sont différents, mais compatibles, ils sont convertis dans le même type par le compilateur. Le mécanisme (ensemble de règles) est appelé **conversion automatique de type**.  
`c = (9.0 / 5)*(f - 32) ;`
- Il est possible de forcer la conversion d'un opérande. C'est ce qu'on appelle la **conversion de type explicite**.  
`c = ((float) 9 / 5)*(f - 32) ;`

## Conversion automatique de type

### Hiérarchie

Double

float

long

Int

Short and char

- **char** et les opérandes courts sont convertis en **int**
- Les types de données inférieurs sont convertis en types de données supérieurs et le résultat est de type supérieur.
- Les conversions entre les types non signés et signés peuvent ne pas donner de résultats intuitifs.

### Exemple

```
float f; double d; long l;  
int i; short s;  
d + f    /* f sera converti en double */  
i / s    /* s seront convertis en int */  
l / i    /* i est converti en long;  
résultat  
        long */
```

## Conversion de type explicite

```
c = (float)9 / 5 * ( f - 32 ) ;
```

- La forme générale d'un opérateur de conversion de type est

**(type-nom) expression**

- Il est généralement recommandé d'utiliser des transtypes explicites plutôt que de s'appuyer sur des conversions de type automatiques.

### Exemple

```
C = (flottant) 9/5 * (f - 32)
```

- La conversion **float** en **int** provoque la perte de la partie fractionnaire
- la conversion **double** en **float** provoque l'arrondi des chiffres
- **long int** à **int** provoque la suppression des bits d'ordre supérieur.

# Priorité et associativité des opérateurs en C

Symbole	Type d'opération	Associativité
[ ] ( ) . -> ++ -- (postfixe)	Expression	De gauche à droite
sizeof & * + - ~ ! ++ -- (préfixe)	Unaire	De droite à gauche
* / %	Multiplicatif	De gauche à droite
+ -	Additif	De gauche à droite
<< >>	Décalage au niveau du bit	De gauche à droite
< > <= >=	Relationnel	De gauche à droite
== !=	Égalité	De gauche à droite
&	ET au niveau du bit	De gauche à droite
^	OU exclusif au bit	De gauche à droite
	OU inclusif au niveau du bit	De gauche à droite
&&	ET logique	De gauche à droite
	OU logique	De gauche à droite
? :	Expression conditionnelle	De droite à gauche
= *= /= %= += -= <<= >>= &= ^=  =	Affectation simple et composée	De droite à gauche
,	Évaluation séquentielle	De gauche à droite

*It's time to write your first C program.*

Sujets abordés dans cette section :

- ✓ Structure d'un programme
- ✓ Votre premier programme
- ✓ Commentaires
- ✓ Le programme d'accueil

# Structure d'un programme C

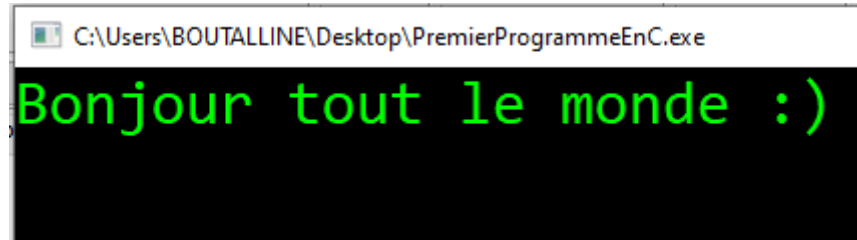
Partie des préprocesseurs (`#include`, `#define`)  
Variables globales

```
void main ( void)
{
    // Des déclarations locales
    // Partie des initialisations
    // Partie des entrées
    // Traitements
    // Partie des sorties
} // Fin de la fonction main

// autres fonctions selon les besoins
```

```
#include<stdio.h>
/* Directive de préprocesseur pour inclure
les fonctions standards d'entrée/sortie
standard dans le programme */
void main(void)
{
    printf("Bonjour tout le monde :) \n");
}
```

Sortie :



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\BOUTALLINE\Desktop\PremierProgrammeEnC.exe". The command prompt displays the output "Bonjour tout le monde :)" in green text on a black background.

## Exemple

*Écrire un programme complet qui calcule la moyenne de trois nombres entiers*

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

```
void main()
```

```
{
```

```
    // Partie des déclarations
```

```
    int nombre1, nombre2, nombre3;
```

```
    double moyenne ;
```

```
    // Partie des entrées
```

```
    printf("Entrer les valeurs des trois nombres :");
```

```
    scanf("%d%d%d",&nombre1,&nombre2,&nombre3);
```

```
    //Traitements
```

```
    moyenne = (nombre1 + nombre2 + nombre3)/3.0;
```

```
    // Partie des sorties
```

```
    printf("La moyennes des trois nombres est : %f \n",moyenne);
```

```
}
```

Sortie :

C:\Users\BOUTALLINE\Desktop\MoyenneTroisNombresEntiers.exe

```
Entrer les valeurs des trois nombres :2 3 5
La moyennes des trois nombres est : 3.333333
```