

# Language C



```
1 /* This line basically imports the "stdio" header file, part of
2  * the standard library. It provides input and output functionality
3  * to the program.
4  */
5 #include <stdio.h>
6
7 /*
8  * Function (method) definition. This outputs "Hello, world" to
9  * standard output.
10 */
11 void sayHello() {
12     // printf() in the specified text (with optional
13     // formatting options)
14     printf("Hello, world\n");
15 }
16
17 /*
18  * This is a "main function". The compiled program will run the code
19  * defined here.
```

Dr El Hadji Bassirou TOURE  
Ecole Supérieure Polytechnique (DGI-ESP-UCAD)  
2020 - 2021

# C : Hello, world !

C helloWorld.c X

C helloWorld.c > ...

```
1  #include<stdio.h>
2
3  int main()
4  {
5      printf("Hello, world");
6  }
```

# C : Hello, world !



The image shows a code editor window with a tab labeled 'helloWorld.c'. The code is as follows:

```
1 #include<stdio.h>
2
3 int main()
4 {
5     printf("Hello, world");
6 }
```

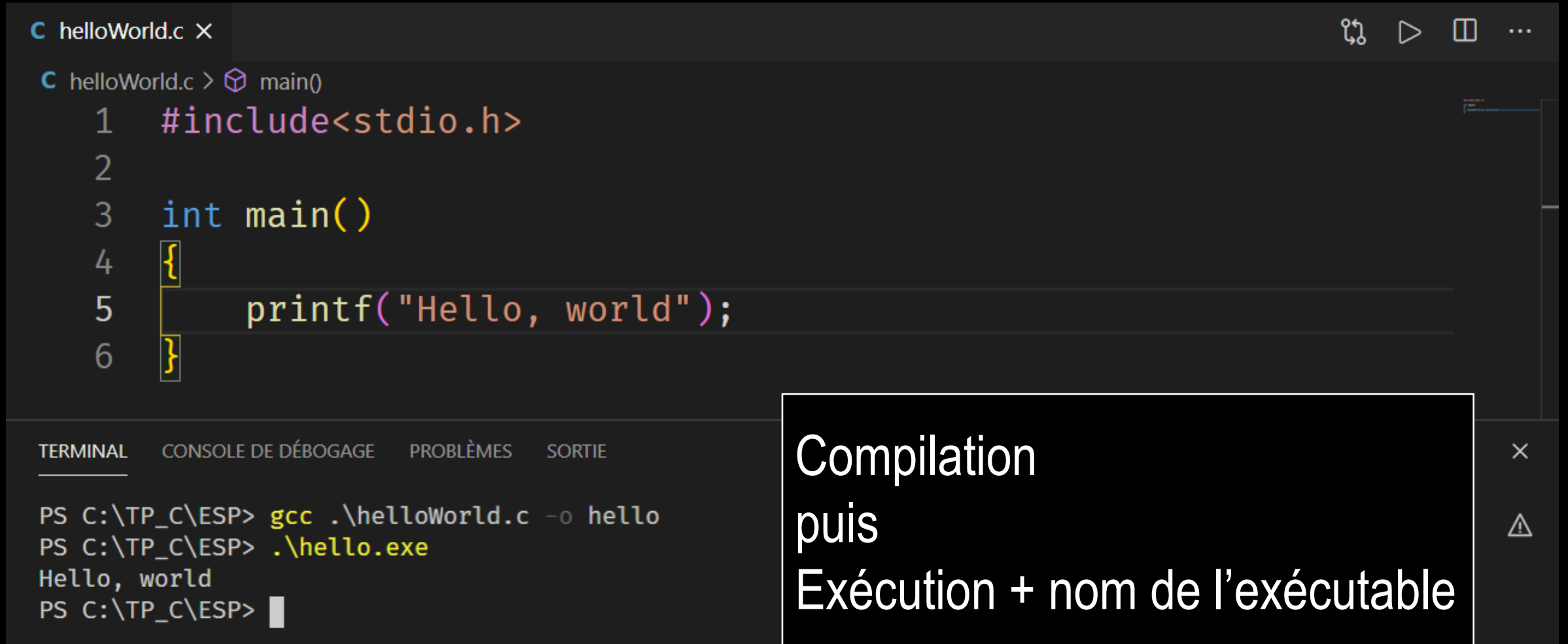
Below the code editor is a terminal window with tabs for 'TERMINAL', 'CONSOLE DE DÉBOGAGE', 'PROBLÈMES', and 'SORTIE'. The terminal shows the following commands and output:

```
PS C:\TP_C\ESP> gcc .\helloWorld.c
PS C:\TP_C\ESP> .\a.exe
Hello, world
PS C:\TP_C\ESP> 
```

Overlaid on the bottom right of the terminal window is a white box with the text:

Compilation  
puis  
Exécution

# C : Hello, world !



The image shows a code editor window with a tab labeled 'C helloWorld.c'. The code is as follows:

```
C helloWorld.c > main()
1  #include<stdio.h>
2
3  int main()
4  {
5      printf("Hello, world");
6  }
```

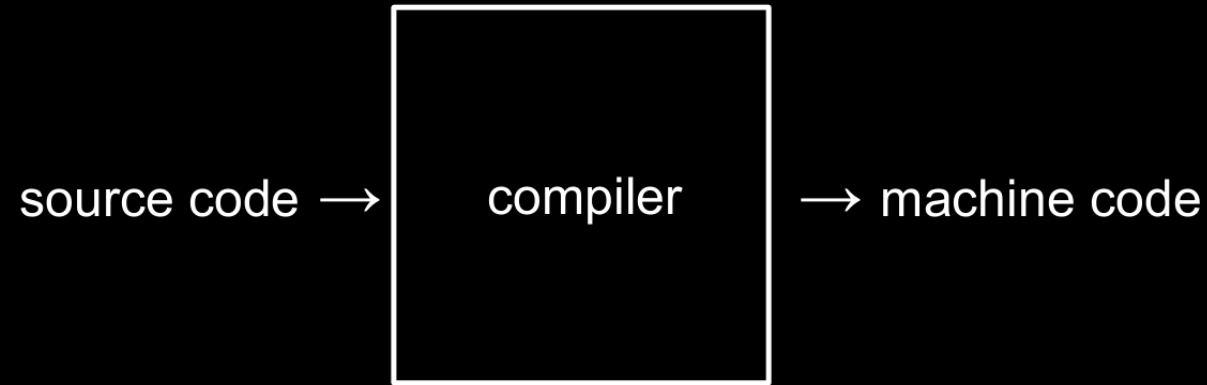
Below the code editor is a terminal window with tabs for 'TERMINAL', 'CONSOLE DE DÉBOGAGE', 'PROBLÈMES', and 'SORTIE'. The terminal shows the following commands and output:

```
PS C:\TP_C\ESP> gcc .\helloWorld.c -o hello
PS C:\TP_C\ESP> .\hello.exe
Hello, world
PS C:\TP_C\ESP> █
```

Overlaid on the bottom right of the terminal window is a white-bordered box containing the following text:

Compilation  
puis  
Exécution + nom de l'exécutable

# Programme



```
helloWorld.c x
helloWorld.c > ...
1  #include<stdio.h>
2
3  int main()
4  {
5      printf("Hello, world");
6  }
```



```
01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000010 00000000 00111110 00000000 00000001 00000000 00000000 00000000
10110000 00000101 01000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11010000 00010011 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 01000000 00000000 00111000 00000000
00001001 00000000 01000000 00000000 00100100 00000000 00100001 00000000
00000110 00000000 00000000 00000000 00000101 00000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 01000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 01000000 00000000 00000000 00000000 00000000 00000000
11111000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
11111000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
00001000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000011 00000000 00000000 00000000 00000100 00000000 00000000 00000000
00111000 00000010 00000000 00000000 00000000 00000000 00000000 00000000
...
```

# Compilateur gcc

- Compilation : `gcc nomProg.c`
- Exécution : `a.exe` (windows) ou `./a.out` sur UNIX
- Alternatives
  - Compilation : `gcc nomProg.c -o nomDeMonExe`
  - Exécution : `nomDeMonExe.exe` (windows) ou `./nomDeMonExe` (UNIX)

# Le C

- Langage de programmation rapide, petit, à usage général, et est indépendant de la plateforme.
- Utilisé pour la programmation système ( compilateurs et interpréteurs, SE, BD, microcontrôleurs, etc.)
- Statique (compilé), typé, structuré et impératif.
- « C is quirky, flawed, and an enormous success. » –Ritchie

# Les bases

- Déclaration de variables : `int i ; float f;`
- Initialisation: `char c='A'; int x = y = 10;`
- Operateurs: `+, -, *, /, %`
- Expressions: `int x, y, z; x = y*2+z*3 ;`
- Fonction: `int factorial (int n);`  
`/*Une fonction prenant un int , retournant un int */`



# Définition

- Type de données
  - Ensemble de valeurs
  - Spécifie la taille occupée par la variable en mémoire
  - Détermine les opérateurs pouvant être appliqués à la variable
  - C est un langage (faiblement) typé
    - Conversion implicite,
    - Conversion explicite (potentiellement dangereux)
- Opérateur
  - Spécifie comment un objet doit être manipulé
  - Peut être binaire (%), unaire (++), ternaire (?)

# Définition

- Expression
  - Combinaison de valeurs, variables, opérateurs et fonctions
  - Produit une valeur
- Variable
  - Nom faisant référence à une valeur stockée en mémoire ou à une expression
- Soit: `int x = 0, y = 0; y = x + 2;`
  - `x` et `y` sont des variables
  - `y = x + 2` est une expression
  - `+` est un opérateur.

# Types de données

- C a une petite catégorie de types
  - Types numériques (int, float, double)
  - Types caractères (char)
  - Types définis par l'utilisateur (struct, union)

	signed	unsigned
short	<b>short int</b> x; <b>short</b> y;	<b>unsigned short</b> x; <b>unsigned short int</b> y;
default	<b>int</b> x;	<b>unsigned int</b> x;
long	<b>long</b> x;	<b>unsigned long</b> x;
float	<b>float</b> x;	N/A
double	<b>double</b> x;	N/A
char	<b>char</b> x; <b>signed char</b> x;	<b>unsigned char</b> x;

# Types de données

- Les tailles de ces différents types sont dépendants de la machine/compilateur
- Cependant, on a toujours :
  - `sizeof(char) < sizeof(short) <= sizeof(int) <= sizeof(long)` and
  - `sizeof(char) < sizeof(short) <= sizeof(float) <= sizeof(double)`

```
#include <limits.h>
#include <float.h>

int main(void)
{
    printf("min int = %i\n", INT_MIN);
    printf("max int = %i\n", INT_MAX);
    printf("min char = %i\n", CHAR_MIN);
    printf("max char = %i\n", CHAR_MAX);
    printf("min float = %.3e\n", FLT_MIN);
    printf("max float = %.3e\n", FLT_MAX);
    printf("min double = %.3e\n", DBL_MIN);
    printf("max double = %.3e\n", DBL_MAX);
    printf("min short is %i\n", SHRT_MIN);
    printf("max short is %i\n", SHRT_MAX);
}
```

# Constantes

- Constantes symboliques

- Exemple

- `#define PI 3.14159`

- `printf("Perimètre (rayon: 1.5) = %f ", 2*PI*1.5);`

- Déclaration de variables comme constante

- Exemple

- `const int class effectif = 40;`

- `printf("Moyenne classe (cumul notes: 100) = %f", (float)100/40);`

# Déclarations

- Déclaration de variables :
  - `type nom-de-variable [=value];`
    - `char x; /* non initialisée */`
    - `char x='A'; /* + intialisation à 'A'*/`
    - `char x='A',y='B';`
- Déclaration de fonctions :
  - `type nom-de-fonction ([liste de paramètres]);`
    - `int factoriel(int); ⇔ int factoriel(int x);`
    - `void afficher(int ); ⇔ void afficher(int x);`
    - `int puissance10();`

# Opérateurs

## ■ Opérateurs arithmétiques

operator	meaning	examples
+	addition	<code>x=3+2; /*constants*/</code> <code>y+z; /*variables*/</code> <code>x+y+2; /*both*/</code>
-	subtraction	<code>3-2; /*constants*/</code> <code>int x=y-z; /*variables*/</code> <code>y-2-z; /*both*/</code>
*	multiplication	<code>int x=3*2; /*constants*/</code> <code>int x=y*z; /*variables*/</code> <code>x*y*2; /*both*/</code>

# Opérateurs

## ■ Opérateurs arithmétiques (2)

operator	meaning	examples
/	division	<code>float x=3/2; /*produces x=1 (int /) */</code> <code>float x=3.0/2 /*produces x=1.5 (float /) */</code> <code>int x=3.0/2; /*produces x=1 (int conversion)*/</code>
%	modulus (remainder)	<code>int x=3%2; /*produces x=1*/</code> <code>int y=7;int x=y%4; /*produces 3*/</code> <code>int y=7;int x=y%10; /*produces 7*/</code>



# Opérateurs

## ■ Opérateurs relationnels (comparaison)

operator	meaning	examples
>	greater than	3>2; /*evaluates to 1 */ 2.99>3 /*evaluates to 0 */
>=	greater than or equal to	3>=3; /*evaluates to 1 */ 2.99>=3 /*evaluates to 0 */
<	lesser than	3<3; /*evaluates to 0 */ 'A' < 'B' /*evaluates to 1*/
<=	lesser than or equal to	3<=3; /*evaluates to 1 */ 3.99<3 /*evaluates to 0 */

operator	meaning	examples
==	equal to	3==3; /*evaluates to 1 */ 'A' == 'a' /*evaluates to 0 */
!=	not equal to	3!=3; /*evaluates to 0 */ 2.99!=3 /*evaluates to 1 */

# Opérateurs

## ■ Opérateurs logiques

operator	meaning	examples
&&	AND	<code>((9/3)==3) &amp;&amp; (2*3==6); /*evaluates to 1 */</code> <code>('A'=='a') &amp;&amp; (3==3) /*evaluates to 0 */</code>
	OR	<code>2==3    'A'=='A'; /*evaluates to 1 */</code> <code>2.99&gt;=3    0 /*evaluates to 0 */</code>
!	NOT	<code>!(3==3); /*evaluates to 0 */</code> <code>!(2.99&gt;=3) /*evaluates to 1 */</code>

## ■ Court-circuit

- `(3==3) || (2/0)` : 2e expression pas évaluée.
- `(0) && ((x=x+1)>0)` : 2e expression pas évaluée.

# Opérateurs

- Opérateurs d'incrementation (resp. décrémentation)
- Postfix :  $x++$ 
  - $x++$  est un raccourci de  $x=x+1$
  - $y=x++$  est un raccourci de  $\{ y=x; x=x+1 \}$
  - Incrémentation est faite après (post) évaluation de l'expression.
- Préfix :  $++x$ 
  - $++x$  est un raccourci de  $x=x+1$
  - $y=++x$  est un raccourci de  $\{ x=x+1; y=x \}$
  - Incrémentation est faite avant (post) évaluation de l'expression.

# Opérateurs

## ■ Opérateurs bit à bit

operator	meaning	examples
&	AND	0x77 & 0x3; /*evaluates to 0x3 */ 0x77 & 0x0; /*evaluates to 0 */
	OR	0x700   0x33; /*evaluates to 0x733 */ 0x070   0 /*evaluates to 0x070 */
^	XOR	0x770 ^ 0x773; /*evaluates to 0x3 */ 0x33 ^ 0x33; /*evaluates to 0 */
«	left shift	0x01<<4; /*evaluates to 0x10 */ 1<<2; /*evaluates to 4 */
»	right shift	0x010>>4; /*evaluates to 0x01 */ 4>>1 /*evaluates to 2 */

- & est vrai si et seulement si les deux opérandes sont vrais.
- | est vrai si l'un des opérandes est vrai.
- ^ est vrai si et seulement si l'un des opérandes est vrai.

# Opérateurs

- Opérateurs d'affectation compactés

- $x+=1 /* \Leftrightarrow x=x+1*/$

- $x-=1 /* \Leftrightarrow x=x-1*/$

- $x*=10 /* \Leftrightarrow x=x*10 */$

- $x/=2 /* \Leftrightarrow x=x/2$

- $x\%=2 /* \Leftrightarrow x=x\%2$

# Opérateurs

## ■ Opérateur ? (conditionnel ternaire)

```
if(x>0)
```

```
    sign = 1;
```

```
else
```

```
    sign = -1;
```

```
⇔    sign = x>0 ? 1 : -1;
```

```
if(x%2 == 1)
```

```
    estPair = 1;
```

```
else
```

```
    estPair = 0;
```

```
⇔    estPair = x%2 == 1 ? 1 : 0;
```

# Conversion de types

- Conversion implicite
  - Ajustement de type int -> float
    - `int x = 7; float y = x/2 ; /* y = 3.0 */`
  - Promotion numérique char -> int
    - `char x = 'c'; int a = x + 2 ; /* 101 */`

# Conversion de types

- Conversion avec perte (à éviter)
  - Exemple : float -> int
    - `float x = 2.5; int y = x; /* y = 2; */`
    - `float x = 7.0; int y = x/2 ; /* y = 3 */`



# Conversion de types

- Conversion explicite
  - En utilisant l'opérateur de cast
    - `int x = 7; float y = (float)x/2 ;`
      - `/* y = 3.5 : conversion de x en float puis division */`
    - `int x = 7; float y = (float)(x/2) ;`
      - `/* y = 3.0 : division entière suivie de conversion du résultat*/`

# Précédence et ordre d'évaluation

- `++,--, (cast), sizeof` ont la plus grande priorité
- `*,/, %` ont une plus grande priorité que,
- `==, !=` ont une plus grande priorité que `&&, ||`
- `=` a la plus petite priorité
- Utiliser des `()` pour plus de clarté ou éviter les effets de bord des opérateurs.
  - $y = x * 3 + 2 /* \Leftrightarrow y = (x * 3) + 2 /*$
  - $x != 0 \&\& y == 0 /* \Leftrightarrow (x != 0) \&\& (y == 0) /*$
  - $d = c >= 'o' \&\& c <= 'g' /* \Leftrightarrow d = (c >= 'o') \&\& (c <= 'g') /*$