Fondamentaux du langage C

- Variables / Types
- Entrées / Sorties
- Structures conditionnelles

Asma Gabis
asma.gabis@efrei.fr

L1, L1-BN, L1-INT, L1-Plus, L1-BDX – 2023/2024

01

Le langage C

01

Pourquoi le langage C ?

- C est un langage structuré
- C est un langage concis (seulement 32 mots clef).
- Actuellement, le langage le plus utilisé dans les systèmes embarqués
- C est stable
- Assembleur de haut niveau
- Très portable
- Compilation facile à comprendre
- Produit du code efficace
- C est la base d'autres langages (Java, C++, awk, Perl).

Jargon du programmeur

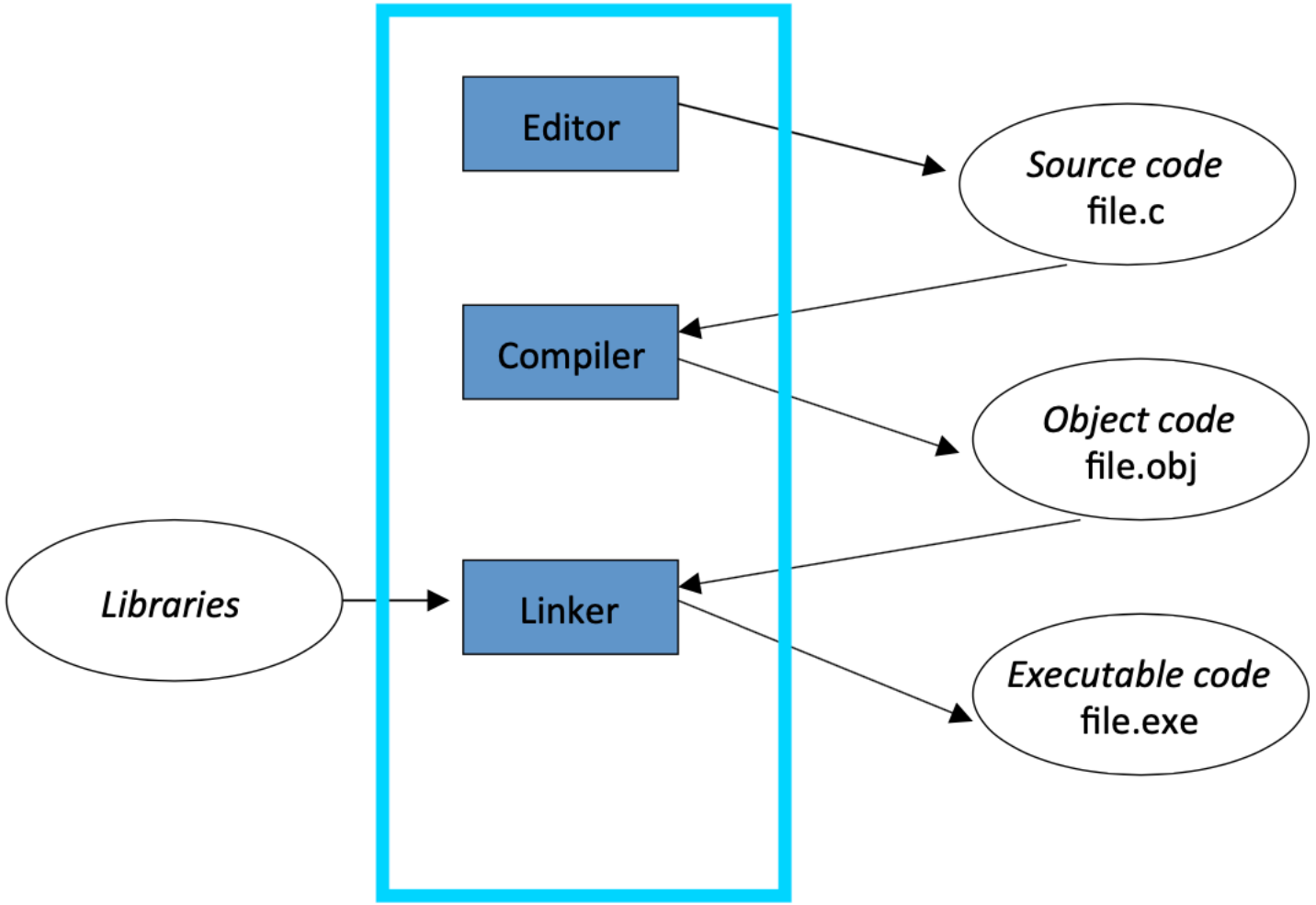
Quelques mots seront souvent utilisés :

- **Code source:** Ce que vous tapez sur la machine. Le programme que vous écrivez.
- **Compiler (build)**: À partir du code source, construire un programme compréhensible par une machine.
- **Exécutable**: Le programme compilé qu'une machine peut exécuter.
- **Bibliothèque**: Fonctions prédéfinies du langage C.
- **Fichier entête**: Fichiers ayant l'extension **.h** qui sont inclus au début du code source

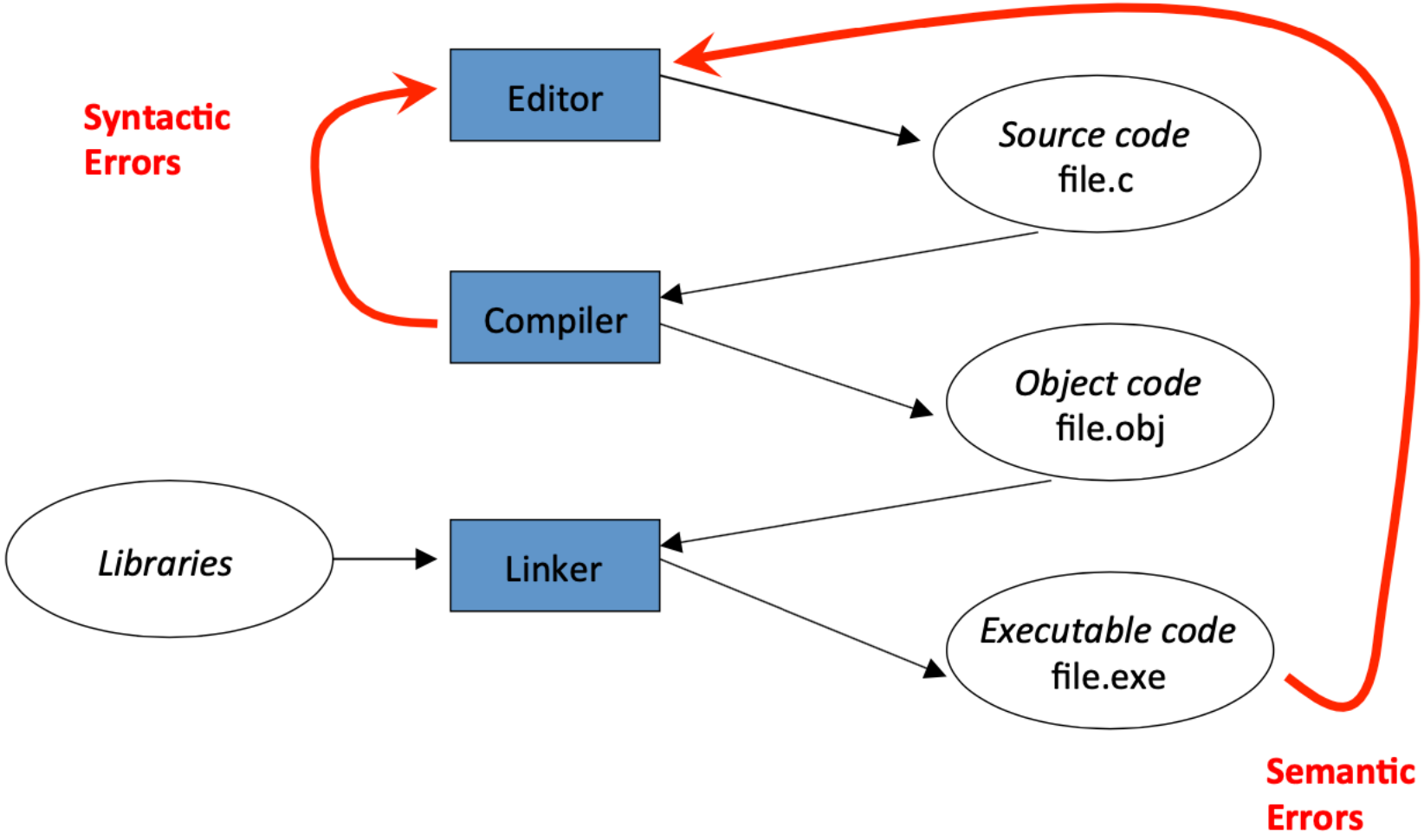
Quelques règles du langage C

- Toute instruction se termine par un point-virgule **';'**
- L'indentation est importante pour la lisibilité du code
- Espaces et indentations sont ignorés par le compilateur
- C est sensible à la casse
  - Tous les mots clefs en C sont minuscules
  - Taper INT, Int, etc au lieu de int est une erreur détectée par le compilateur
- Un retour à la ligne est représenté par **\n**

Compiler et exécuter un programme C



Debuggage


Syntactic
Errors
Editor
Source code
file.c
Compiler
Object code
file.obj
Libraries
Linker
Executable code
file.exe
Semantic
Errors

02

Les bases de la programmation C

02

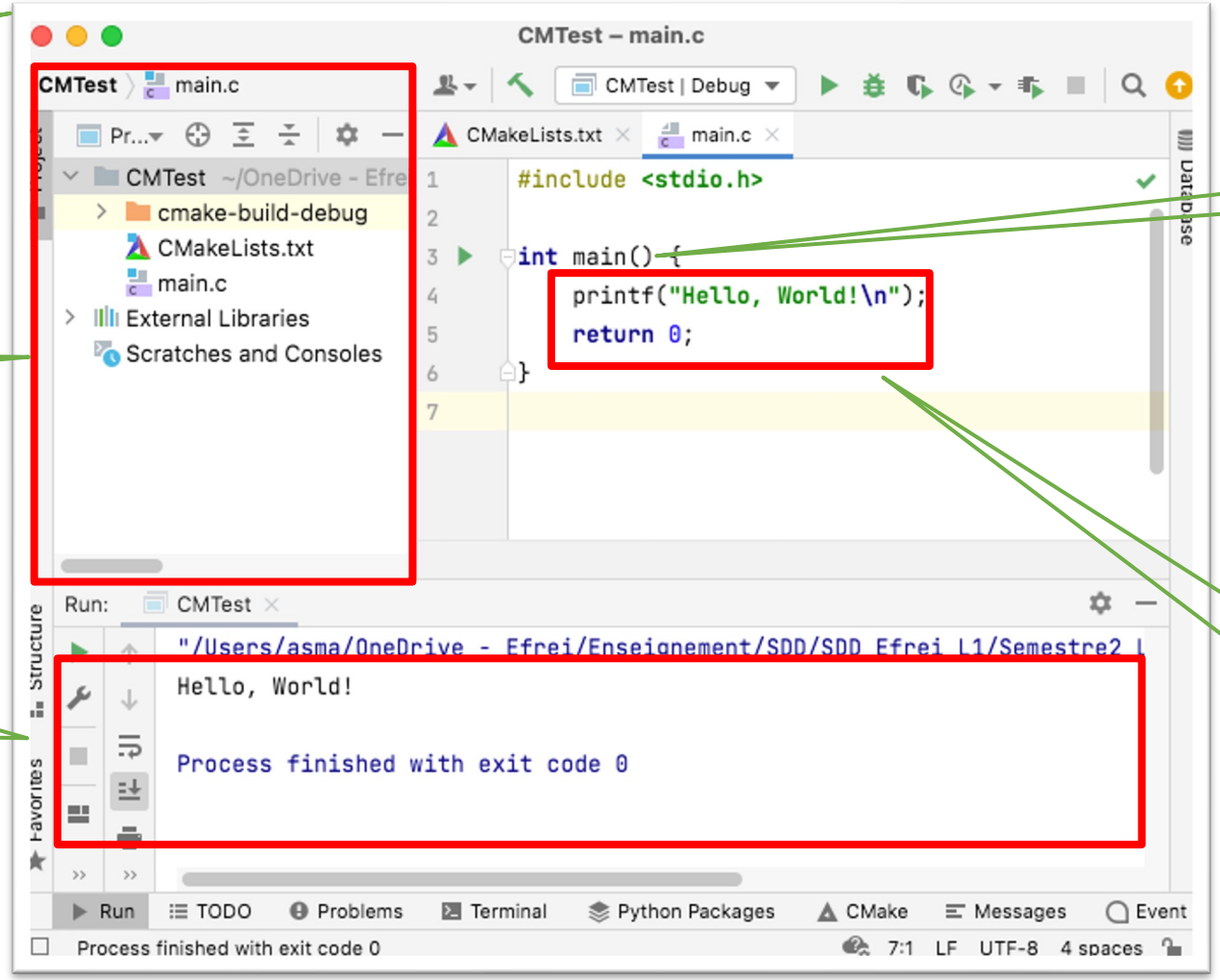
Les bases de la programmation C

Structure d'un programme C

IDE: CLion

Explorateur de projet

Console : affichage du résultat ou des messages d'erreur



Fonction principale

Exemple de programme en C

Mon premier programme C

- L'unique porte d'entrée de tout programme C est la fonction **main**.
- Les instructions du programme sont ajoutées dans le bloc de la méthode main.
- Un bloc d'instructions commence par une accolade ouvrante **{** et se termine par une accolade fermante **}.**
- La fonction main se termine toujours par l'instruction **return 0;**

```c
int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Programme principal
(Fonction principale)
Liste des instructions.
Ici, instruction
d'affichage

Les bases de la programmation C

Les commentaires

- Texte écrit dans le langage naturel et qui est ignoré par le compilateur.
- Servent à ajouter des informations permettant de mieux comprendre le code.
- Le langage C supporte (au moins) deux types de commentaires:
  - Commenter une seule ligne en la précédant par « // »
  - Commenter un bloc d'instructions en l'entourant par « /* .... */ »

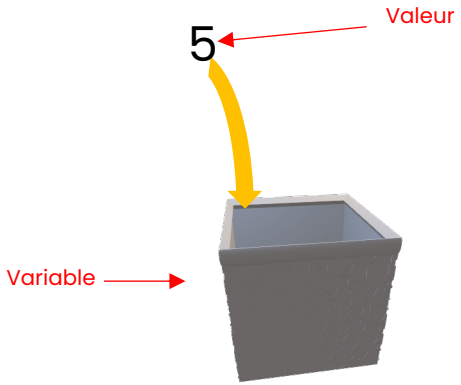
```c
int a, b; // variables permettant de stocker
des valeurs

int sum; // variable permettant de stocker le
résultat de la somme
```

```c
/*
Dans ce programme l'utilisateur va saisir deux entiers a, b
Il effectue la somme et la stocke dans la variable sum
 */
int a, b;
int sum;
```

Les commentaires peuvent être utilisés sur des portions de code pour éviter de les inclure dans la compilation.

Variables / Types

Variable : Conteneur (boîte) pour y stocker une valeur :



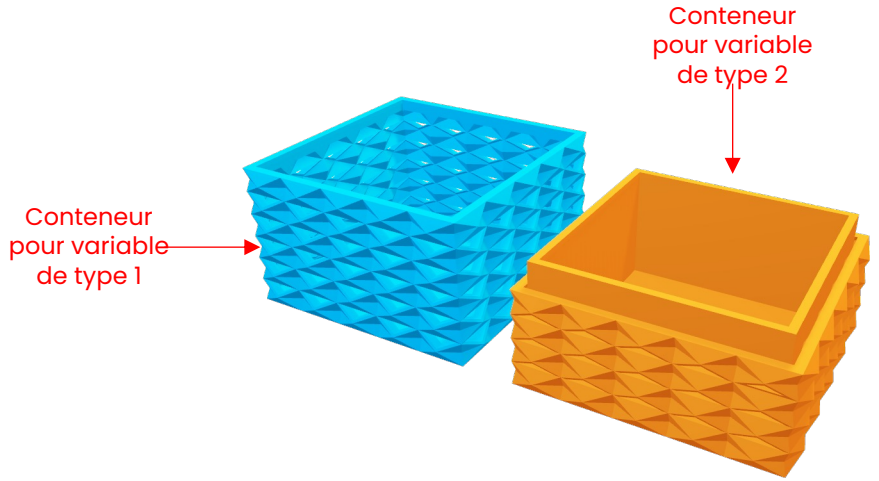
- Nombre : pour faire des opérations mathématiques (additions, calcul TVA, ..)
- Mot (chaîne de caractères) : pour former des phrases.

Il est possible de mettre dans un même conteneur plusieurs valeurs à des moments différents.

La nouvelle valeur à mettre **écrase** (annule) la valeur précédente.

Type : Détermine la nature de la variable :

- Une variable d'un type donné peut prendre plusieurs valeurs mais **pas** plusieurs types
- Plusieurs types de variables sont présent.



Comment ça marche en C ?

Les bases de la programmation C

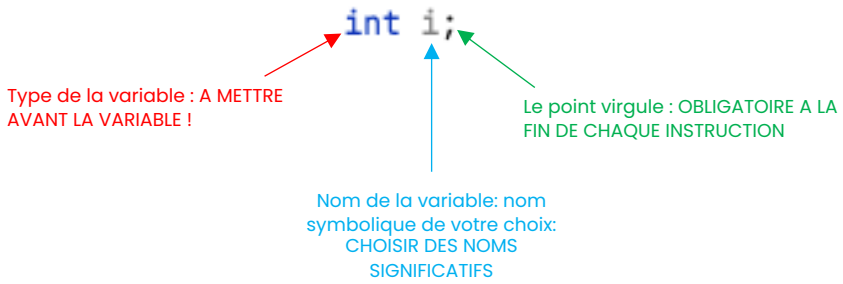
Les types fréquents

Primitive	Signification	Taille (en octets)
char	Caractère	1
int	Entier	4
long	Entier long	8
float	flottant (réel)	4
double	flottant double	8

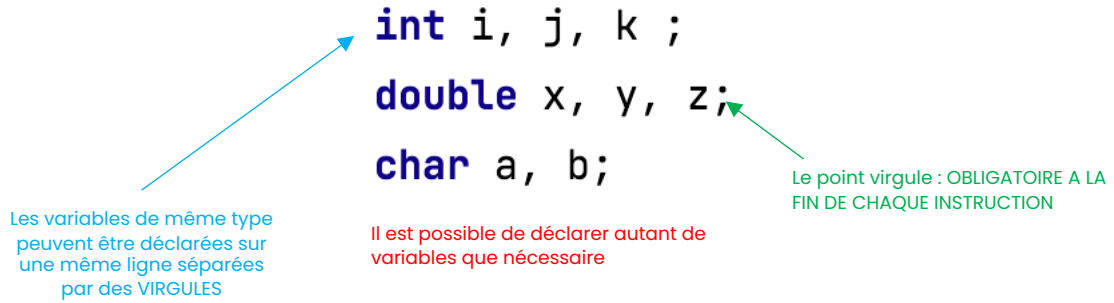
Les bases de la programmation C

Manipulation des variables

Définition d'une variable



Définition de plusieurs variables



Initialisation d'une variable

```
int i;
i = 3 ; // Initialisation de la variable bien après sa définition

double x = 2.5; // Initialisation de la variable en même temps que sa définition

char b = 'b', a = 'a'; // Initialisation de plusieurs variables au moment de leurs définitions
```

Modification d'une variable

```
int i = 3; // Définition et initialisation de i: valeur de i = 3

i = 5 ; // Affectation d'une nouvelle valeur à i : 5 annule 3

i = i * 2; // Modification en utilisant une expression : 10 annule 5
```

Les constantes

- Les constantes sont proches des variables mais elles ne peuvent pas changer de valeur au cours du programme.

- Avec la directive **#define**, il est possible de donner un nom symbolique à une constante.

Définition des constantes

```
#define TVA 19.6

#define PI 3.14

#define CUBE_FACES 6
```

Exemple

```c
#include <stdio.h>

// Définition d'une constante
#define PI 3.14

int main()
{
    // définition des variables
    double rayon = 2.5;
    double aireCercle;

    // Calcul de l'aire d'un cercle
    aireCercle = PI * rayon * rayon;

    return 0;
}
```

Conventions de nommage

Variable

- Il est d'usage de toujours commencer le nom de la variable par une une majuscule.
- Le caractère '_' peut être utilisé comme un séparateur
- Le langage C est sensible à la casse.
- Les noms de variables doivent être significatifs

```c
int Moteur_voiture;

double Moyenne_etudiant;

char Lettre_alphabet;
```

Constante

- Le nom est complètement en majuscules avec les mots séparés par des tirets de soulignement (_).
- Le nom d'une constante doit être significatif.

```c
#define TVA 19.6

#define PI 3.14

#define CUBE_FACES 6
```

Les opérateurs

- Les opérateurs sont des symboles qui permettent de manipuler des variables.

- Le langage C supporte plusieurs types d'opérateurs
  - Les opérateurs arithmétiques : ils permettent (en général) de modifier mathématiquement la valeur d'une variable
  - Les opérateurs d'affectation: Ils permettent de chaîner plusieurs opérations en une seule.
  - Les opérateurs d'incrémentation : Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable.
  - Les opérateurs relationnels : ils sont utilisés pour effectuer des comparaisons entre deux variables
  - Les opérateurs logiques : ils permettent d'évaluer la véracité (ou pas) de plusieurs conditions

Les bases de la programmation C

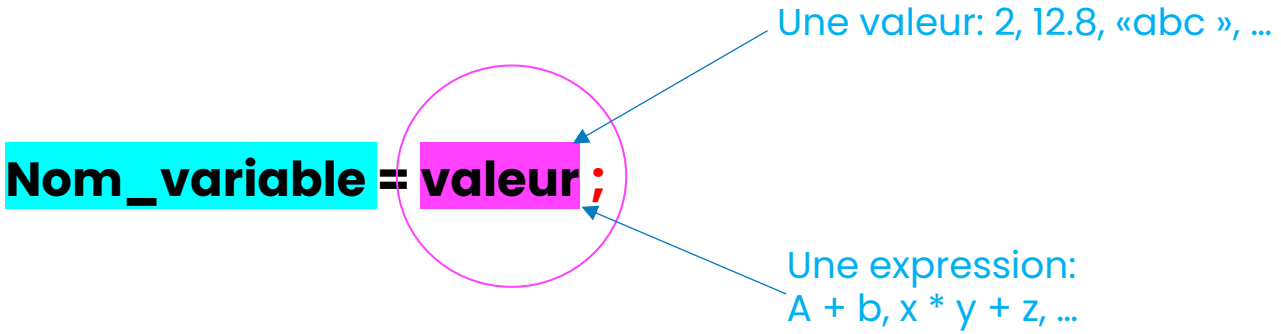
Les opérateurs arithmétiques

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
+	opérateur d'addition	additionne deux valeurs	x+3	10
-	opérateur de soustraction	Soustrait deux valeurs	x-3	4
*	opérateur de multiplication	Multiplie deux valeurs	x*3	21
/	opérateur de division	Divise deux valeurs	x/3	2.3333333
%	Opérateur du modulo	Reste de la division	x%3	1
=	opérateur d'affectation	Affecte une valeur à une variable	x=3	Met la valeur 3 dans la variable x

Les bases de la programmation C

Les opérateurs

L'affectation : Syntaxe Vs erreurs à éviter



MAIS

5 = y	✗
a + b = y	✗
a + 1 = b	✗
a, b = 3, 4;	✗

Équivalent à
`a = (b = 3)`

MAIS `a = b = 3 ;` ✓

Les bases de la programmation C

Les opérateurs d'affectation

Opérateur	Effet	Syntaxe	Instruction équivalente
+=	additionne deux valeurs et stocke le résultat dans la variable (à gauche)	X += Y	X = X + Y
-=	soustrait deux valeurs et stocke le résultat dans la variable	X -= Y	X = X - Y
***=**	multiplie deux valeurs et stocke le résultat dans la variable	X *= Y	X = X * Y
/=	divise deux valeurs et stocke le résultat dans la variable	X /= Y	X = X / Y

Les bases de la programmation C

Les opérateurs d'incrémentation

Opérateur	Dénomination	Effet	Syntaxe	Instruction équivalente	Résultat (avec x valant 7)
++	Incrémentation	Augmente d'une unité la variable	x++	x = x + 1	8
--	Décrémentation	Diminue d'une unité la variable	x--	x = x - 1	6

Les opérateurs d'incrémentation

En fonction de la position de l'opérateur d'incrémentation par rapport à la variable, on applique les opérations **post-incrémentation (post-décrémentation)** et **pré-incrémentation (pré-décrémentation)**.

Post-incrémentation

- L'opérateur est **post positionné** (après la variable)
- S'applique sur instruction fusionnant au moins deux opérations .
- L'incrémentation est effectuée **après** l'évaluation de l'expression.

```
int b , x = 3;
b = x++;
```

⟺

```
int b , x = 3;
b = x;
x = x + 1 ;
```

```
b = 3 , x = 4.
```

Pré-incrémentation

- L'opérateur est **pré positionné** (avant la variable)
- S'applique sur instruction fusionnant au moins deux opérations .
- L'incrémentation est effectuée **avant** l'évaluation de l'expression.

```
int b , x = 3;
b = -- x;
```

⟺

```
int b , x = 3;
x = x - 1;
b = x;
```

```
b = 2 , x = 2.
```

Les bases de la programmation C

Les opérateurs relationnels

Opérateur	Dénomination	Effet	Exemple	Résultat
== **A ne pas confondre avec le signe d'affectation (=)!!**	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	x==3	Retourne 1 si X est égal à 3, sinon 0
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	x<3	Retourne 1 si X est inférieur à 3, sinon 0
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	x<=3	Retourne 1 si X est inférieur ou égal à 3, sinon 0
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	x>3	Retourne 1 si X est supérieur à 3, sinon 0
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	x>=3	Retourne 1 si X est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	x!=3	Retourne 1 si X est différent de 3, sinon 0

Les bases de la programmation C

Les opérateurs logiques

Opérateur	Dénomination	Effet	Syntaxe
||	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur True si la variable vaut False, False si elle vaut True)	(!condition)

Les caractères et les chaînes de caractères

- Une chaîne de caractères est représentée en C par les doubles quotes (guillemets).

```
cours = "langageC";
nom = "John";
```

- Un caractère seul est représenté en C par de simples quotes.

```
char a , b ;
a = 'A';
b = 'B';
```

Conversion de types de données

- C'est le fait de modifier le type d'une donnée en un autre.

Conversion implicite

- Affecter une valeur d'un type donné à une variable déclarée dans un type différent.
- Modification effectuée par le compilateur sans retourner d'erreur.

```c
int x;
double y;
x = 8.324; // x prendra la partie entière sans arrondi de la valeur réelle.

y = 4; // conversion automatique avec une partie décimale nulle.

printf("x = %d \ny = %f.", x, y);
```

```
x = 8
y = 4.000000.
```

Conversion explicite (cast)

- Modification du type de donnée forcée.
- Mettre le type souhaité entre parenthèses avant la variable ou l'expression à convertir.

```c
int x ;
double y;

x = (int) 8.324;
y = (double) 4;

printf("x = %d \ny = %f.", x, y);
```

```
x = 8
y = 4.000000.
```

Le type booléen

- Le langage C permet de calculer et de manipuler des données booléennes **mais** ne définit pas le type booléen par défaut.
- Par convention :
  - La valeur nulle représente la constante booléenne **faux** ou **false.**
  - Toute autre valeur sera assimilée à la constante **vrai** ou **true.**

- En utilisant la directive **#define**, il est possible de créer un type pseudo booléen.

```c
#include <stdio.h>

// Définition d'une constante
#define BOOL int
#define TRUE 1
#define FALSE 0

int main()
{
    BOOL verif; // Utiliser le type booléen intuitivement
    verif = TRUE; // affectation intuitive des valeurs booléennes.
    printf("verif = %d.", verif);
}
```

```
verif = 1.
```

03

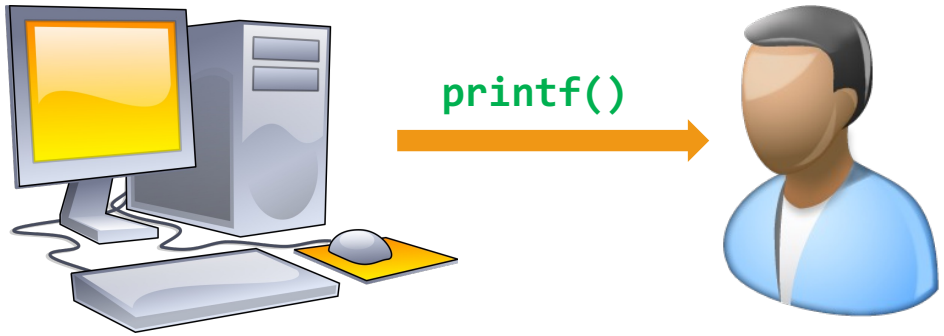
Les entrées / sorties en langage C

03

Les entrées / sorties en C

Affichage à l'écran

- `printf()`: permet d'effectuer un affichage à l'écran. Il peut être:
  - C'est une fonction prédéfinie de la bibliothèque standard **stdio.h**
  - L'utilisation de printf nécessite la présence de **#include<stdio.h>**
  - printf offre un affichage formaté.

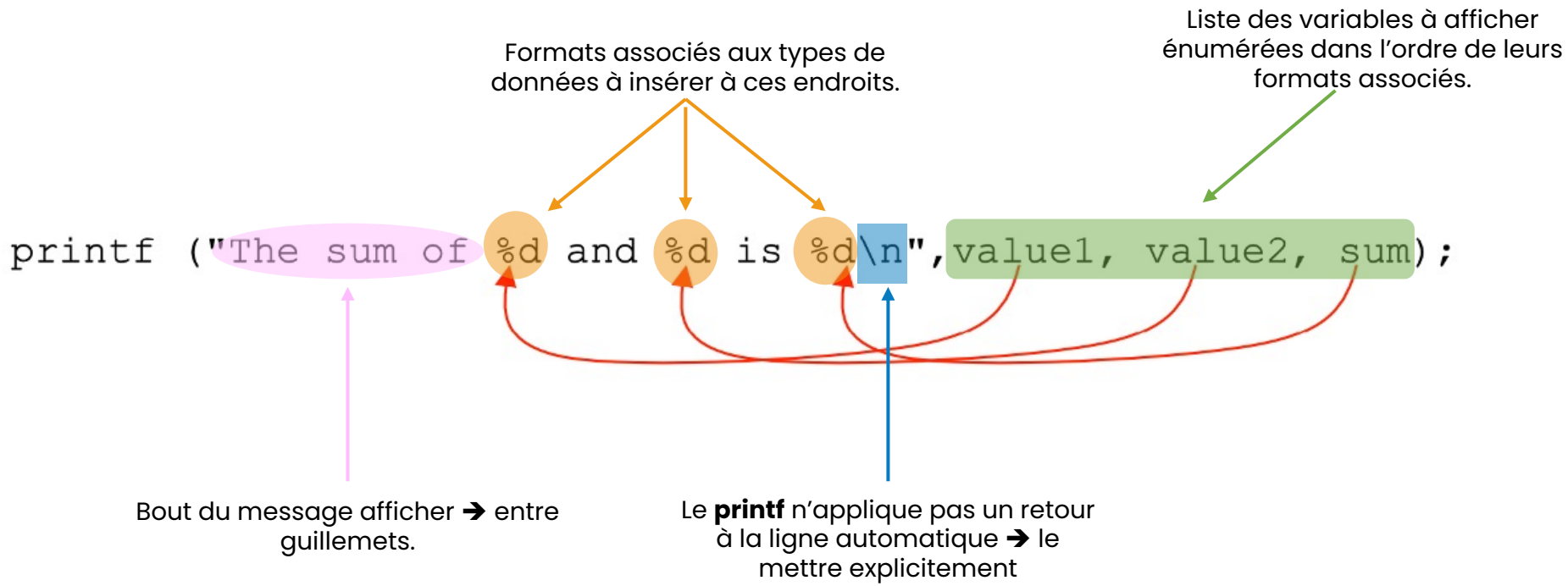



```c
#include <stdio.h>

int main()
{
    printf("Hello World !\n");
    printf("Mon age est %d ans.", 19); // %d est le format d'un entier
}
```

```
Hello World !
Mon age est 19 ans.
```

Les entrées / sorties en C

Affichage à l'écran: Syntaxe



Les entrées / sorties en C

Affichage à l'écran: Options de formatage

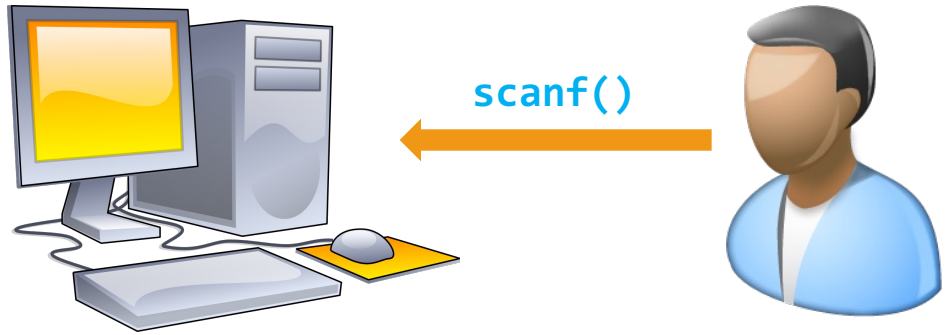
- %d –> entier décimal
- %ld –> entier décimal long
- %x –> entier hexadécimal
- %c –> caractère

  %s –> chaîne de carctères
- %f –> nombre réel
- %lf –> nombre réel double …

Les entrées / sorties en C

Saisie de données

- `scanf()` permet de demander à l'utilisateur de fournir des informations.
- L'utilisation de scanf nécessite la présence de **#include<stdio.h>**
- Le scanf porte deux arguments :
  - Le format de la donnée à saisir.
  - La variable où sera stockée la donnée saisie.
  - Les variables sont toutes précédées par le caractère **&** sauf si elles sont de type « chaine de caractères ».



Les entrées / sorties en C

Saisie de données : Exemples

- `scanf("%c", &nextChar);`
  - lit un seul caractère et le sauvegarde dans la variable *nextChar*

- `scanf("%f", &radius);`
  - lit un nombre réel et le sauvegarde dans la variable *radius*

- `scanf("%d %d", &length, &width);`
  - lit deux entiers décimaux (séparés par un espace), sauvegarde le premier dans *Length* et le second dans *width*

- `scanf("%d;%d", &length, &width);`
  - lit deux entiers décimaux successivement (séparés par un point virgule), sauvegarde le premier dans *Length* et le second dans *width*

04

Les structures de contrôle

04

Les structures de contrôle

Structures conditionnelles

- if
- if - else
- switch - case

Boucles

- while
- For
- Do - while

Les structures de contrôle

Les structures conditionnelles

- **if : syntaxe**

Peut être:
- Condition simple: expression de comparaison
- Condition composée: expression logique

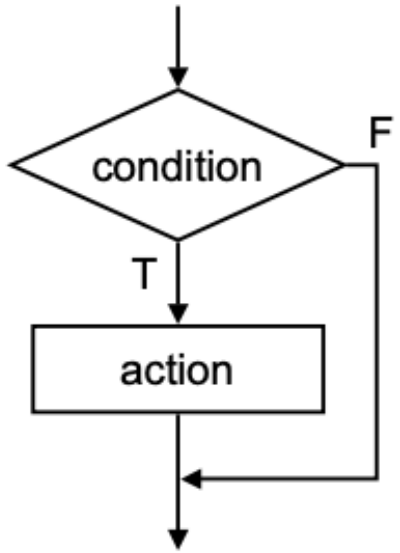
Mot clé : OBLIGATOIRE

```
if (condition)
{
    action(s)
}
```

Les parenthèses autour de la condition sont obligatoires

Utiliser les accolades pour délimiter le bloc d'instructions du if .

Une ou plusieurs action(s) exécutée(s) uniquement si condition est vraie



Les structures de contrôle

Les structures conditionnelles

- **if : exemples**

```
if (x <= 10)
{
    y = x * x + 5;
    z = (2 * y) / 3;
}
```

Ces deux instructions seront exécutées si la condition du « if » est vraie

```
if (x <= 10)
    y = x * x + 5;
z = (2 * y) / 3;
```

Seule cette instruction sera exécutée si la condition du « if » est vraie car pas d'accolades

```
if (0 <= age && age <= 11)
    kids = kids + 1;
```

Toutes les conditions du « if » doivent être vraies pour exécuter cette instruction

```
if (mois==4 || mois==6 || mois==9 || mois==11)
    printf("Le mois contient 30 jours");
```

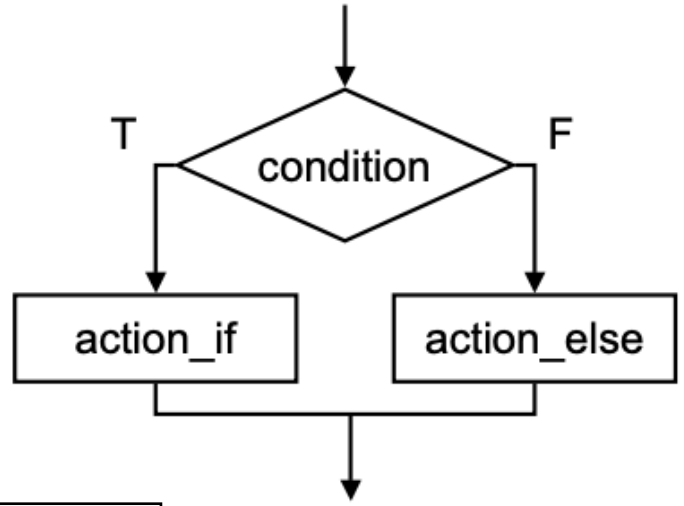
Il suffit d'une seule condition du « if » à vrai pour exécuter cette instruction.

Les structures de contrôle

Les structures conditionnelles

- **if - else: syntaxe et exemple**

```
if (condition)
{
   instructions si condition vérifiée
}
else
{
   instructions si condition non vérifiée
}
```




```c
if (delta < 0)
{
    printf("Aucune solution possible");
}
else
{
    printf("Il existe une ou deux solutions");
}
```

Les structures de contrôle

L'instruction conditionnelle

- **Instructions de test imbriquées**

```
if (condition 1)
{
   if (condition 2)
   {
      ….
   }
}
else
{
   if (condition 3)
   {
      …
   }
   else
   {
      ….
   }
}
```

```
if (mois == 1)
{
    printf("Janvier");
}
else
{
    if (mois == 2)
    {
        printf("Fevrier");
    }
    else
    {
        if (mois == 3)
        {
            printf("Mars");
        }
        else
        {
            ....
        }
    }
}
```

Les structures de contrôle

Le switch … case

- Une façon plus élégante et plus lisible d'écrire des tests multiples imbriqués.

```
switch (variable)
{
   case 0:
   {
      instruction pour cas 0
      break;
   }
   case 1:
   {
      instructions pour cas 1
      break;
   }
    …
   default : instructions pour le cas par défaut
}
```

```
switch (mois)
{
     case 1:
     {
          printf("Janvier");
          break;
     }
     case 2:
     {
          printf("Fevrier");
          break;
     }
 //…
     default : printf("Ce numéro de mois n'existe pas");
}
```