

# ***INFO 505 - Programmation C II*** ***L3 – 2024-25***

## **CM1 : Introduction et rappels sur les bases du C**

J.Y. RAMEL

Bureau 2D-204 - Polytech'Savoie - Bourget du Lac

# Au programme



- **Séances de cours et TD puis TP**
  - En prolongation du cours Programmation C I (revisité)
  - CM (5x1.5h) + TD (5x1.5h) + TP (4x3h)
  - Moodle incrémentalement...
  - IDE VSCode (ou autre) sous Linux (ou windows)
- **Evaluation**
  - Examen semaine 48 (lundi 25/11) + note de TP (individualisée ou en binôme)
  - Autre si besoin
- **Contenu**
  - CM1. Introduction, Rappels des bases du C
  - CM2. Adressage, Portées, Pointeurs, Tableaux
  - CM3. Structures de données et gestion mémoire dynamique, listes chaînées
  - CM4. Structures++, Pointeurs de fonctions, compilation avancée (pré-compilation, librairies)
  - CM5. Gestion des erreurs, GUI (GTK+), Thread, ...
- **Sources et bibliographie**
  - **Ce support de cours a été construit à partir du cours Programmation C de PolytechTours – Ronan Bocquillon, Pierre Gaucher – 2023**

# On se réveille...



- Histoire du C ?
- Spécificités du C par rapport aux autres langages ?
- Structure d'un programme en C ?
  - Faire par exemple → Conversion fahrenheit vers celsius
  - $C = (F - 32) / 1,8$

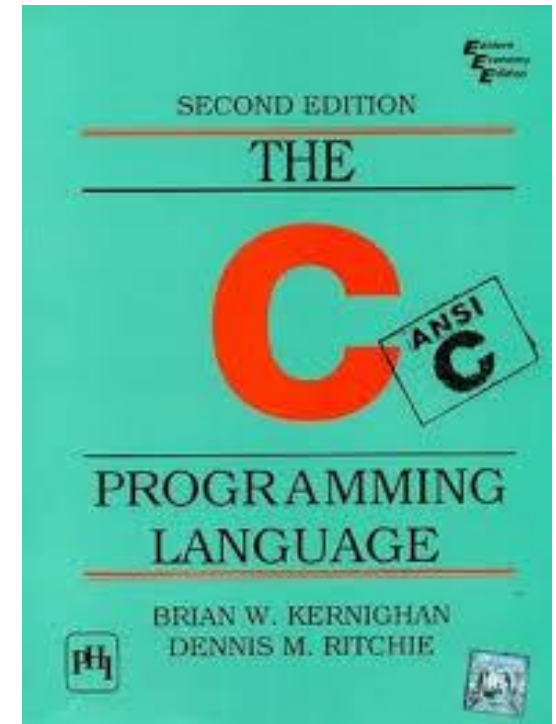
# Introduction

---

- Eléments de normalisation

- 1978 : *Brian W. Kernighan and Dennis M. Ritchie. The C Programming Language. Prentice-Hall, Englewood Cliffs, New Jersey.*
- Donner une définition complète et stable du langage → normalisation

- 1989 : C-ANSI ou **C89** (ANSI : American National Standards Institute)
- 1990 : ISO **C90** (Normalisation ISO : International Organization for Standardization)
- 1999 : **ISO C99**
- 2011 : ISO **C11**
- 2017-2018 : **ISO C17** et ISO **C18**
- <https://koor.fr/C/Index.wp>



# Introduction

---

## Structure d'un programme C et compilation

- Constitué de fichiers sources -> extension **.c** et **.h**
- Créer avec un éditeur de base gedit, notepad++ ou IDE

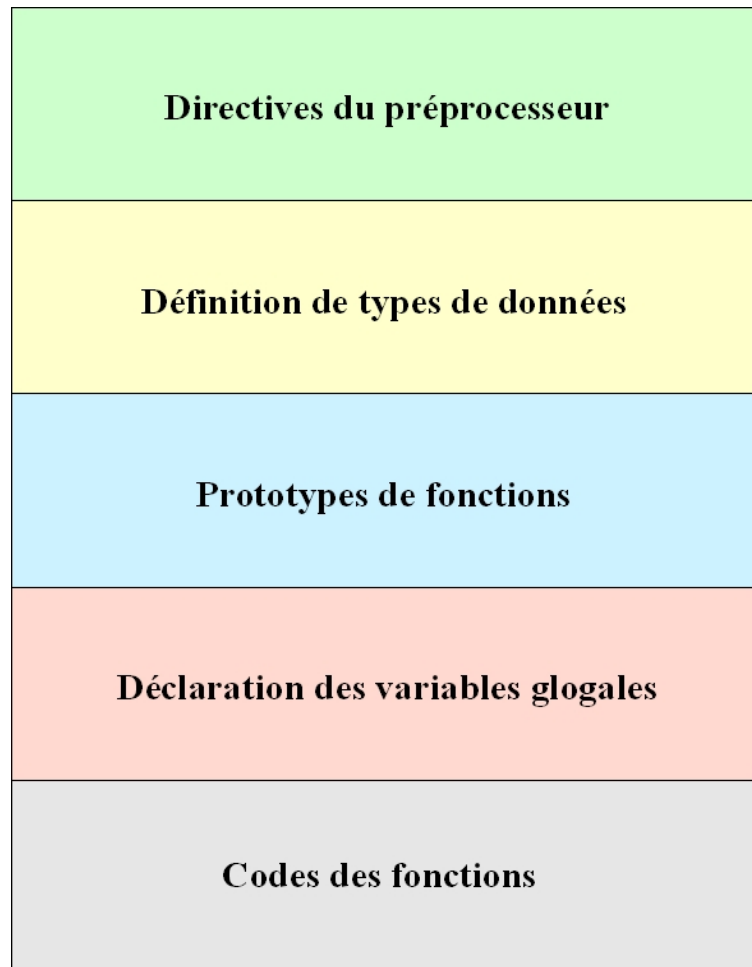
```
// hello.c
#include <stdio.h>
int main () {
    printf("Hello_world!");
    return 0;
}
```

- Transformés en exécutable à l'aide d'un compilateur comme gcc ou clang, ...

# Introduction

## Structure d'un programme C et compilation

### Blocs fonctionnels fichiers .c + .h



#### Directives du préprocesseur

Instructions particulières, prises en compte par le préprocesseur

Sont caractérisées par le caractère **#**

Deux directives importantes :

Directives d'inclusion de fichiers : **#include**

Directives de définition de symboles ou de macros :

**#define**

#### Définition de type de données

Nouveaux types de données composés

Nouveaux noms de type de données

#### Prototypes de fonctions : permet de préciser

Type de retour de la fonction

Nombre d'arguments et leur type

#### Déclaration variables globales

#### Code des fonctions

En-tête de la fonction

Code délimité par un bloc { }

Une fonction particulière : **main**, toujours présente

# Introduction

---

## Structure d'un programme C et compilation

- Constitué de plusieurs fichiers sources -> extension .c + extension.h pour une organisation **en modules**

```
// main.c
void greeting();
int main () {
    greeting();
    return 0;
}
```

```
// messages.c
#include <stdio.h>
void greeting () {
    printf("Hello_world!");
}
```

# Introduction

---

## Structure d'un programme C et compilation

- Une fonction (son prototype) doit être déclarée avant d'être appelée.
- Si une fonction est utilisée dans plusieurs fichiers, il peut être pratique d'écrire sa déclaration une fois dans un fichier d'en-tête, puis de l'inclure là où c'est nécessaire.

```
// main.c
#include "messages.h"
void main () {
    greeting();
}
```

```
// messages.h
void greeting();
```

```
// messages.c
#include <stdio.h>
void greeting () {
    printf("Hello_world!");
}
```



# Introduction

## Structure d'un fichier source .c :

- exemple de main.c

```

1  /**
2   * @brief Table de conversion °F vers °C
3   * @file main.c
4   */
5
6  // Directives du préprocesseur pour include des fichiers d'en tête
7  #include <stdlib.h>
8  #include <stdio.h>
9  #include "ConvertFtoC.h"
10
11 // Fonction main()
12 int main(void) {
13     double f_min = 0.0, f_pas = 5.0, f = 0.0;
14
15     for (int i = -5; i < 15; i++) {
16         f = f_min + i * f_pas;
17         printf ("%+0.2lf f \t=> %.2lf C\n", f, ConvertFtoC (f));
18     }
19     return EXIT_SUCCESS;
20 }

```

- Eléments constitutifs d'un fichier .c (ou .h)
- Des **mots réservés** (keywords, directives)  
#include, double, void, for, return
  - Des **délimiteurs** pour structurer le code  
{}, [], ( ),...
  - Des **commentaires**  
/\* .....\*/ ou // ou /\*\*  
et \*/
  - Des **littéraux**  
Items faisant référence à des éléments  
de valeur fixe dans le code source
  - Des **identifiants**  
Variables, type de donnée,  
nom de fonction
  - Des **fonctions**  
printf, **main**
  - Des **opérateurs**  
=, <, ++

# Introduction

## Structure d'un fichier d'en-tête .h :

- exemple de ConvertFtoC.h

```

1  /**
2   * @brief prototype de fonction
3   * @file ConvertFtoC.h
4   */
5
6  #ifndef _CONVERTFTOC_H
7  #define _CONVERTFTOC_H
8
9  /**
10 * @brief Conversion de °Fahrenheit vers ° Celsius
11 * @param temp_f température en ° Fahrenheit à convertir en ° Celsius
12 * @return la température convertie en °Celsius
13 */
14
15 double ConvertFtoC (double temp_f);
16
17 #endif

```

- Eléments constitutifs d'un fichier .c (ou .h)
- Des **mots réservés** (keywords, directives)  
#include, #ifndef, double, void...
  - Des **délimiteurs** pour structurer le code  
{, [], ( ), ...
  - Des **commentaires**  
/\* .....\*/ ou // ou /\*\*  
et \*/
  - Des **littéraux**  
Items faisant référence à des éléments  
de valeur fixe dans le code source
  - Des **identifiants**  
Paramètres, type de donnée  
nom de fonction
  - Des **prototypes de fonctions**

# On se réveille...

---



- Création d'un exécutable ?
- Options de compilation ?
- Création de bibliothèques ?

# Introduction

## Structure d'un programme C et compilation

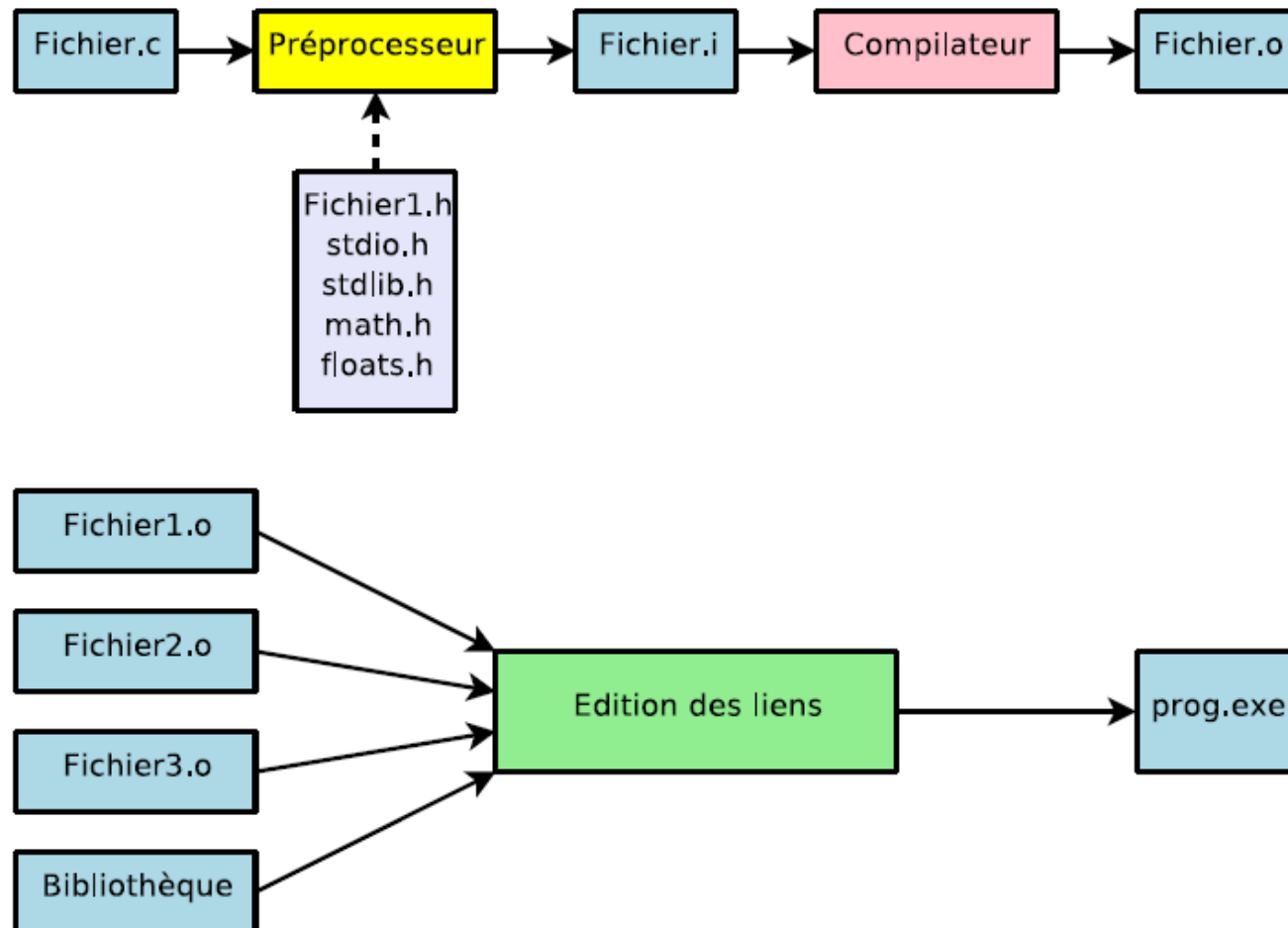
- Chaque fichier .c est **compilé** indépendamment des autres
- La compilation produit des fichiers objet (.o) qui doivent être assemblés par un **linker**
- Pas besoin de tous re-compiler à chaque modification
- Et possibilité d'utiliser des **librairies** déjà existante
- Comme la **C library standard** comprenant stdio.h
- Et possibilité de créer **ses propres librairies**

```
$ gcc -c main.c -o main.o # compile main.c
$ gcc -c messages.c -o messages.o # compile messages.c
$ gcc main.o messages.o -o hello # link
$ ./hello # execute
Hello world!
```



# Introduction

## Structure d'un programme C et compilation



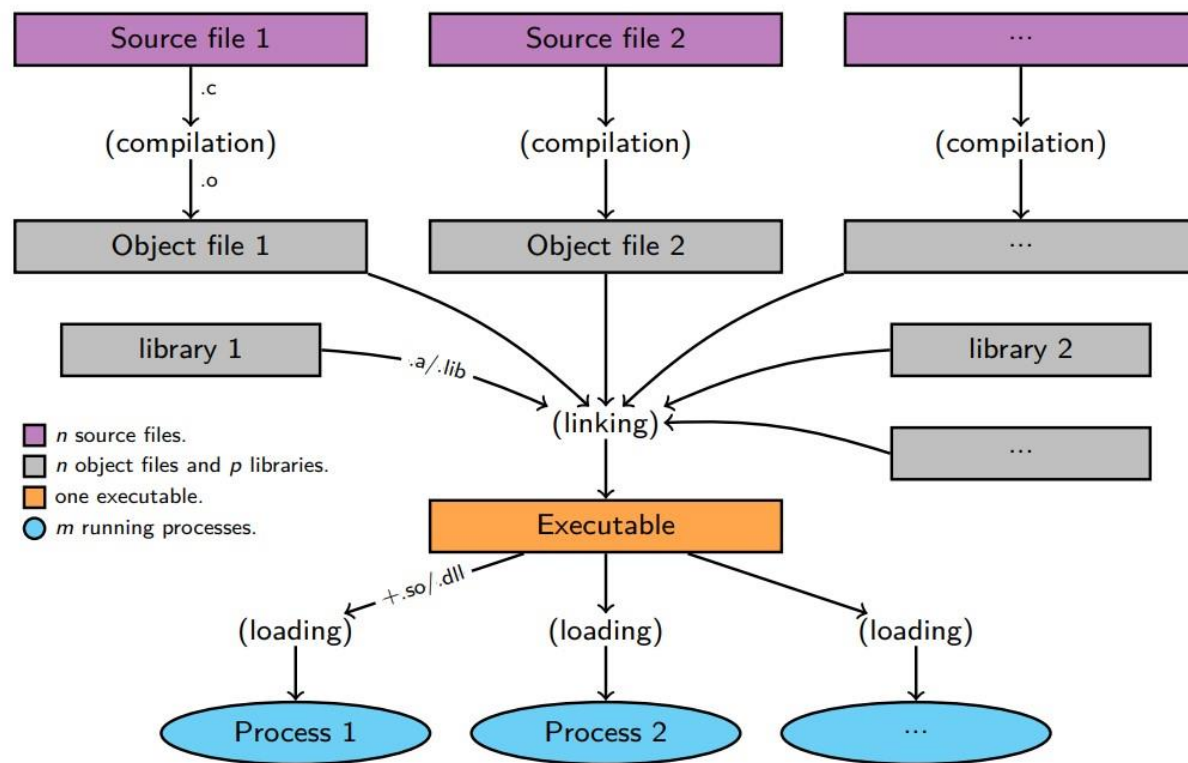
# Introduction

## Structure d'un programme C et compilation



- L'ensemble de ces fichiers sont gérés dans un **projet**, défini au sein d'un **espace de travail**.
- La génération du code exécutable est régie au sein du projet par des **options de compilation** et **d'édition de liens**.

### Compilation process

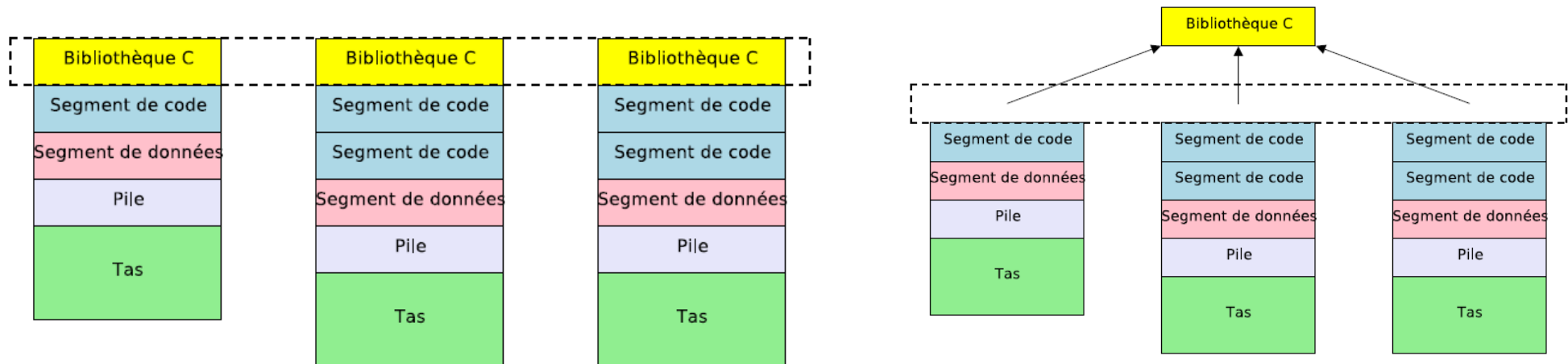


# Introduction



## Utilisation et création de bibliothèques

- Statiques : dupliquer dans chaque exécutable
- Dynamique : Le code de la bibliothèque n'est pas intégré dans l'exécutable → Une seule copie en mémoire !
- libc sous linux : 2.5 Mo x 100 programmes utilisant → 250 Mo de libc !



# Introduction

---

## Librairies statiques (.a) et dynamiques (.so)

Les librairies en C sont par convention toutes préfixées par lib → libmath.a ou libmath.so

Les outils de compilation et de linkage, comme gcc et ld, suivent cette convention pour rechercher automatiquement les bibliothèques lors de la phase de linkage (cf flag -l de gcc).

### Rappels des options principales de gcc

- gcc -c main.c mesfct.c // compilation
- gcc -Wall main.o mes\_fct.o -o myprogram // compilation + linkage avec warning
- gcc -I/my/include/path main.c -o myprogram // -I include path
- gcc -g buggy.c -o buggy
- gcc -fPIC -c mes\_fct.c // génère du Position Independent Code à utiliser notamment pour les bibliothèques partagées (shared libraries).



### Bibliothèques statiques

Une bibliothèque statique en C est simplement la concaténation d'un certain nombre de fichiers objet

**ar -crs libmylib.a file1.o file2.o** // tiret optionnel - c:create – r : replace – s : index



# Introduction

## Bibliothèques dynamiques

En général en C, le fichier pour une bibliothèque dynamique XXX s'appelle libXXX.so

**gcc -shared -fPIC -o libmylib.so file1.o file2.o // -fPIC cf avant**

Utilisation du flag gcc `-lXXX` pour inclure la librairie.

### Attention:

- A la compilation /linkage, la position du flag `-l` et de la liste des librairies est importante pour gcc  
→ a mettre à la fin juste avant `-o` et si libA dépend de libB, vous devez écrire `-lB -lA`.

- A l'exécution, le programme doit pouvoir trouver la localisation de la librairie dynamique

→ `LD_LIBRARY_PATH=/path/to/my/libs ./myprogram`

→ `export LD_LIBRARY_PATH=/path/to/my/libs:$LD_LIBRARY_PATH`

→ Option `-Wl,-rpath` → Syntaxe : `-Wl,-rpath,<chemin>`

- Voir aussi **pkg-config** sous Linux → `gcc -o my_program my_program.c `pkg-config --cflags --libs gtk4``

## (Suite des) Rappels des options principales de gcc (linkage)

- `gcc main.c -L./lib -lmylib1 -lmylib2 -o myprogram // a la compilation, L = library path -l nom de la librairie`
- `gcc main.c -L./lib -lmylib -Wl,-rpath=./lib -o myprogram // a l execution -Wl,-rpath`

`gcc main.c -Wall -Wextra -O2 -g -fPIC -std=c11 -I/include/path -L/library/path -lcustomlib -Wl,-rpath,/library/path -o myprogram`

`LD_LIBRARY_PATH=. ./myprogram`



# On se réveille...

---



- Outils pour faire du C ?
- Makefile ?
- Débugueur ?
- Documentation ?
- DevOps ?

# Introduction

## Gestion des ressources d'un projet C



### Aide à la compilation → **makefile**

Un fichier texte Makefile spécifiant des cibles parsées par l'utilitaire make

```
# avec des variables
```

```
CC = gcc
```

```
CFLAGS = -Wall -g
```

```
OBJ = main.o math_ops.o
```

```
program: $(OBJ)
```

```
    $(CC) $(OBJ) -o program
```

```
main.o: main.c math_ops.h
```

```
    $(CC) $(CFLAGS) -c main.c
```

```
math_ops.o: math_ops.c math_ops.h
```

```
    $(CC) $(CFLAGS) -c math_ops.c
```

```
clean:
```

```
    rm -f *.o program
```

```
# Plus sophistiqué
```

```
# Liste des fichiers sources et objets
```

```
SOURCES = $(wildcard *.c)
```

```
OBJECTS = $(SOURCES:.c=.o)
```

```
# Cible principale
```

```
all: myprogram
```

```
# Règle pour construire l'exécutable
```

```
myprogram: $(OBJECTS)
```

```
    gcc -o $@ $^
```

```
# Règle générique pour compiler les fichiers .c en fichiers .o
```

```
%.o: %.c
```

```
    gcc -c $< -o $@
```

```
# Cible pour nettoyer les fichiers objets et l'exécutable
```

```
clean:
```

```
    rm -f $(OBJECTS) myprogram
```

# Introduction

---

## Gestion des ressources d'un projet C

### Aide au codage → IDE

- Constitué de fichiers sources -> extension **.c**
  - Regroupés dans un répertoire **src** (par exemple)
- Constitués de fichiers d'en tête -> extension **.h**
  - Regroupés dans un répertoire **Include** (par exemple)
- Utilisation d'un IDE : CodeLite, VSCode, CodeBlocks, ...
- OS : Windows ou Linux
- Selon structure du projet -> mise en place d'options particulières pour :
  - La compilation
  - L'édition de liens

# Introduction

The screenshot shows a C++ IDE with a workspace named "ProgramationLangageC". The main window displays the source code for "Convert\_F\_to\_C/main.c":

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "ConvertFtoC.h"
4
5 int main(void) {
6     double f_min = 0.0, f_pas = 5.0, f = 0.0;
7
8     for (int i = -5; i < 15; i++) {
9         f = f_min + i * f_pas;
10        printf ("%+0.2lf f \t=> %.2lf C\n", f, ConvertFtoC (f));
11    }
12    return EXIT_SUCCESS;
13 }
14
```

The left sidebar shows a project tree with folders like "Errno\_gestion\_erreurs", "Gestion\_fichier", and "SDL2\_ttf". The bottom panel shows the "Output View" with the following compilation log:

```
mingw32-make[1]: Entering directory 'D:/Documents/ProgrammationLangageC/Convert_F_to_C'
C:/msys64/mingw64/bin/gcc.exe -c "d:/Documents/ProgrammationLangageC/Convert_F_to_C/main.c" -gdwarf-2 -O0 -Wall -o ../build-Debug/Convert_F_to_C/main.c.o -I. -I. -I.\Include
C:/msys64/mingw64/bin/g++.exe -o ../build-Debug/bin/Convert_F_to_C.exe @../build-Debug/Convert_F_to_C/ObjectsList.txt -L.
mingw32-make[1]: Leaving directory 'D:/Documents/ProgrammationLangageC/Convert_F_to_C'
=== build completed successfully (0 errors, 0 warnings) ===
```

The status bar at the bottom indicates "Ln 1, Col 0" and "SPACES LF C++ UTF-8". The Windows taskbar is visible at the very bottom.

# Introduction

## Aide au codage IDE

## Exemple de ConvertFtoC.exe

C is a **compiled** programming language  
 A correct C program is **portable** between different platforms  
 A C program should compile cleanly **without warnings**

(Ref : Jens Gustedt, Modern C. Manning 2019, 9781617295812. hal-02383654)

The screenshot displays the Visual Studio IDE interface. The main editor shows the source code for `ConvertFtoC.c`. The code includes a comment in French, standard C headers (`<stdlib.h>` and `<stdio.h>`), and a custom header `ConvertFtoC.h`. The `main` function iterates from -5 to 70, converting Fahrenheit values to Celsius using the `ConvertFtoC` function.

```

1  /**
2   * @brief Table de conversion °F vers °C
3   * @file main.c
4   */
5
6  // Directives du préprocesseur pour include des fichiers d'en tête
7  #include <stdlib.h>
8  #include <stdio.h>
9  #include "ConvertFtoC.h"
10
11 // Fonction main()
12 int main(void) {
13     double f_min = 0.0, f_pas = 5.0, f = 0.0;
14
15     for (int i = -5; i < 15; i++) {
16         f = f_min + i * f_pas;
17         printf ("%+.21f f \t=> %.21f C\n", f, ConvertFtoC (f));
18     }
19     return EXIT_SUCCESS;
20 }
  
```

The Output View at the bottom shows the compilation process:

```

C:/msys64/mingw64/bin/mingw32-make.exe -j12 -e -f Makefile
-----Building project:[ Convert_F_to_C - Debug ]-----
mingw32-make[1]: Entering directory 'D:/Documents/ProgrammationLangageC/Convert_F_to_C'
mingw32-make[1]: Leaving directory 'D:/Documents/ProgrammationLangageC/Convert_F_to_C'
=== build completed successfully (0 errors, 0 warnings) ===
  
```

The console window shows the execution output of `Convert_F_to_C.exe`:

```

-25.00 f => -31.67 C
-20.00 f => -28.89 C
-15.00 f => -26.11 C
-10.00 f => -23.33 C
-5.00 f => -20.56 C
+0.00 f => -17.78 C
+5.00 f => -15.00 C
+10.00 f => -12.22 C
+15.00 f => -9.44 C
+20.00 f => -6.67 C
+25.00 f => -3.89 C
+30.00 f => -1.11 C
+35.00 f => 1.67 C
+40.00 f => 4.44 C
+45.00 f => 7.22 C
+50.00 f => 10.00 C
+55.00 f => 12.78 C
+60.00 f => 15.56 C
+65.00 f => 18.33 C
+70.00 f => 21.11 C

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.047 (MM:SS.MS)
Press any key to continue...
  
```

# Introduction

---

## Aide au codage

### Convention de codage

```
// B=6945503773712347754LL, I=5859838231191962459LL, T=0 and S=7
// must be #define'd at compile-time using -D gcc's option.
int main (int b, char**i) {
    long long n=B,
        a=I^n,
        r=(a/b&a)>>4,
        y=atoi(*++i), _=(((a^n/b)*(y>>T)|y>>S)&r)|(a^r);
    printf("%.8s\n", (char*)&_);
}
```

*Byte to binary, no loops, one-liner,  
27th International Obfuscated C Code Contest (2020).*

- KISS principle (“keep it simple, stupid!”)
- Il n'y a pas une seule bonne façon d'écrire du code, mais essayez d'être cohérent.
- Dans le cas de projets collaboratifs, essayez toujours de définir et de suivre des règles communes

# Introduction

## Aide au codage : commentaires et documentation

- Commentaires utiles → Les commentaires doivent expliquer le pourquoi...

```
// Currently, key can be spread in as a prop. This causes a potential
// issue if key is also explicitly declared (ie. <div {...props} key="Hi"/>
// or <div key="Hi" {...props}/>). We want to deprecate key spread,
// but as an intermediary step, we will use jsxDEV for everything except
// <div {...props} key="Hi"/>, because we aren't currently able to tell if
// key is explicitly declared to be undefined or not.
if (maybeKey !== undefined) {
    key = '' + maybeKey;
}
```

*A good comment [React, a well-known open-source JavaScript library].*

- Avec un outil approprié, tel que **Doxygen**, une documentation de haute qualité peut être générée à partir d'un ensemble de fichiers source annotés.

```
/**
 * \brief Writes \a count bytes from \a buf to the filedescriptor \a fd.
 * \param fd    The descriptor to write to.
 * \param buf   The data buffer to write.
 * \param count The number of bytes to write.
 */
size_t write(int fd, const char *buf, size_t count);
```

*A typical doxygen comment. @ ↔ \*



# Introduction

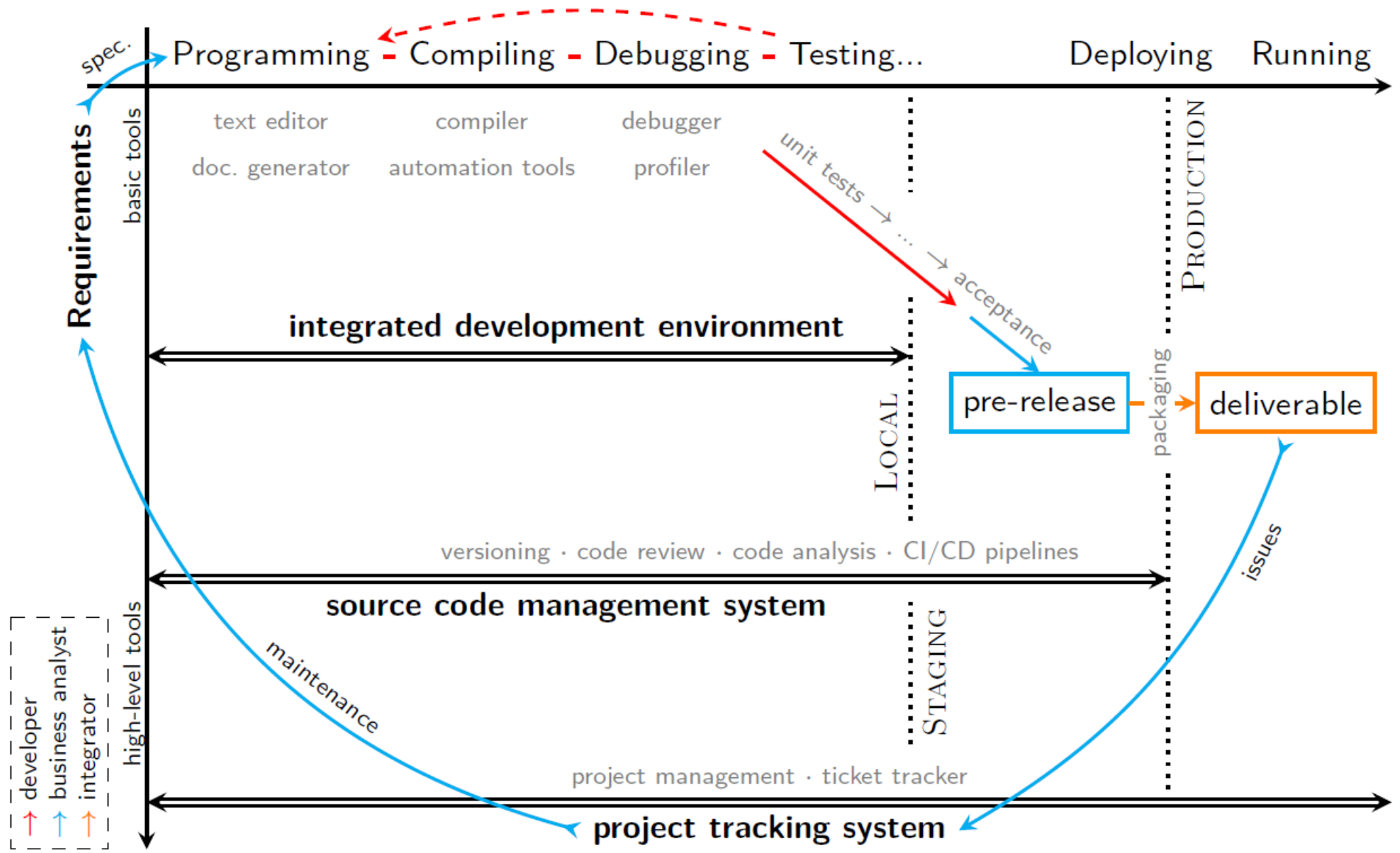
## Aide au codage : trouver l'erreur, tester, déployer

- Divers styles de débogage...
- Outils de débogage
  - gdb, ddd,, IDE...
- Procédures de tests (unitaires/intégration)
- Analyseurs de code
  - SonarQube (code dupliqué, normes de codage, couverture des tests, la complexité du code, les commentaires, problèmes de sécurité,...)
  - Valgrind (un profiler, pour suivre le temps passé dans chaque section du code, un détecteur d'erreurs et de fuites de mémoire)
- Gestionnaires de codes/versions, collaboration : GIT
- Approche DevOps : vise à rapprocher les développeurs (dev) et les opérateurs (ops)  
→ Automatisation des procédures de construction, de test et de déploiement afin d'améliorer et de raccourcir le cycle de vie du développement

```
#include <stdio.h>
void foo (int x) {
    printf("foo: x: %d\n", x);
    ++x;
    printf("foo: x: %d\n", x);
}
void main () {
    int i = 0;
    printf("main: i: %d\n", i);
    foo(i);
    printf("main: i: %d\n", i);
}
```

*The art of print style debugging.*

# Introduction



# Introduction

## Efficacité du C

Energy		Time		Memory	
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.05	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

*Make your choice!*

(<https://dl.acm.org/doi/10.1145/3136014.3136031>)

# On se réveille...

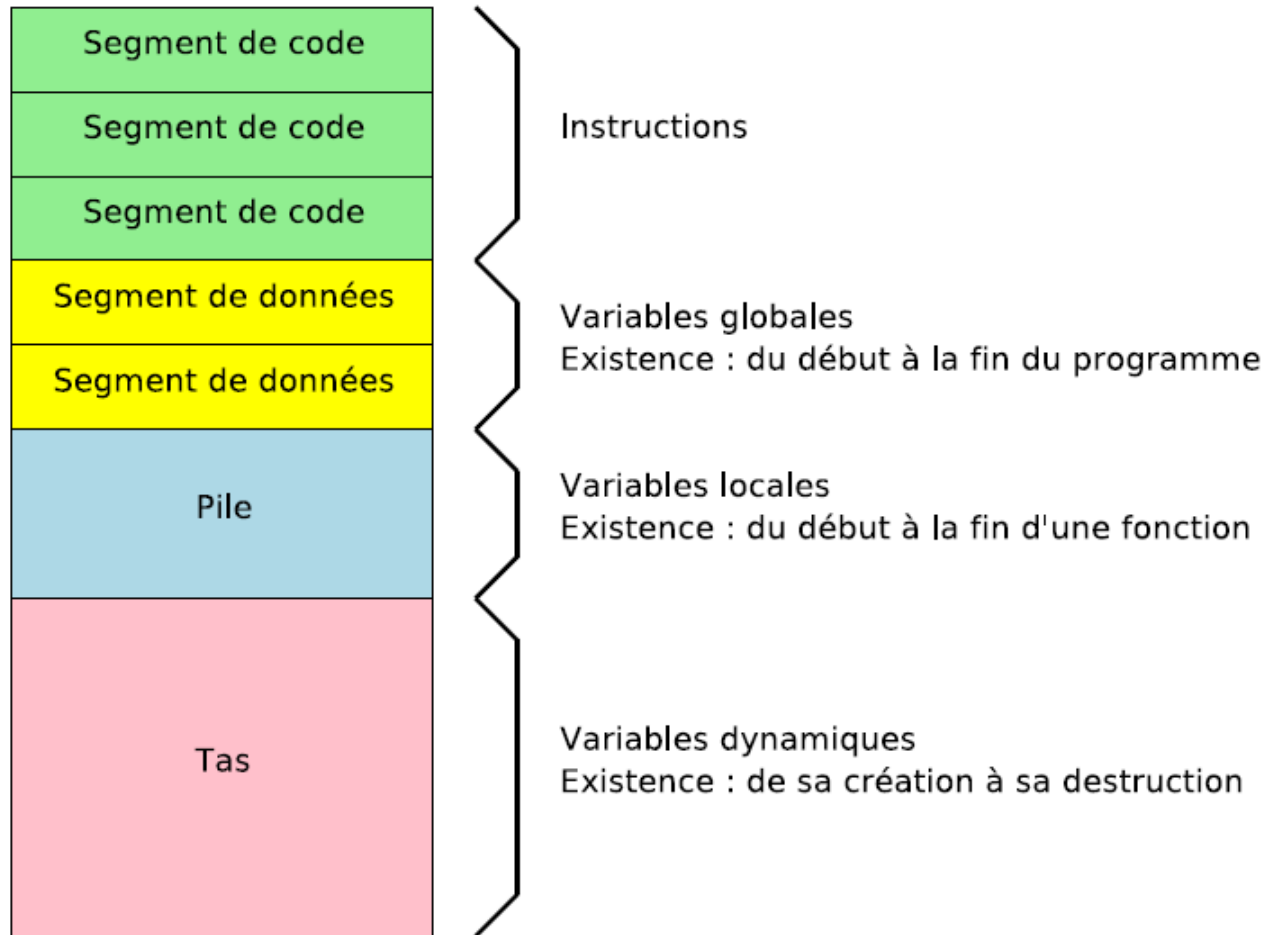
---



- Organisation en mémoire ?
- Programmation C et mémoire ?



## Gestion de la mémoire



### Segment de code = code segment

Zone contigue de la mémoire :

- en lecture seulement
- les instructions en code machine à exécuter
- un prgm peut en avoir plusieurs

### Segment de données = data segment

Zone contigue de la mémoire :

- zones réservées dans le code source
- contient les variables globales
- un prgm peut en avoir plusieurs

### Tas = heap

Zone contigue de la mémoire :

- création dynamique (à l'exécution) de zones de données pendant l'exécution
- ex : enregistrements, files, listes, arbres, . .
- un prgm peut en posséder en général 1 seule

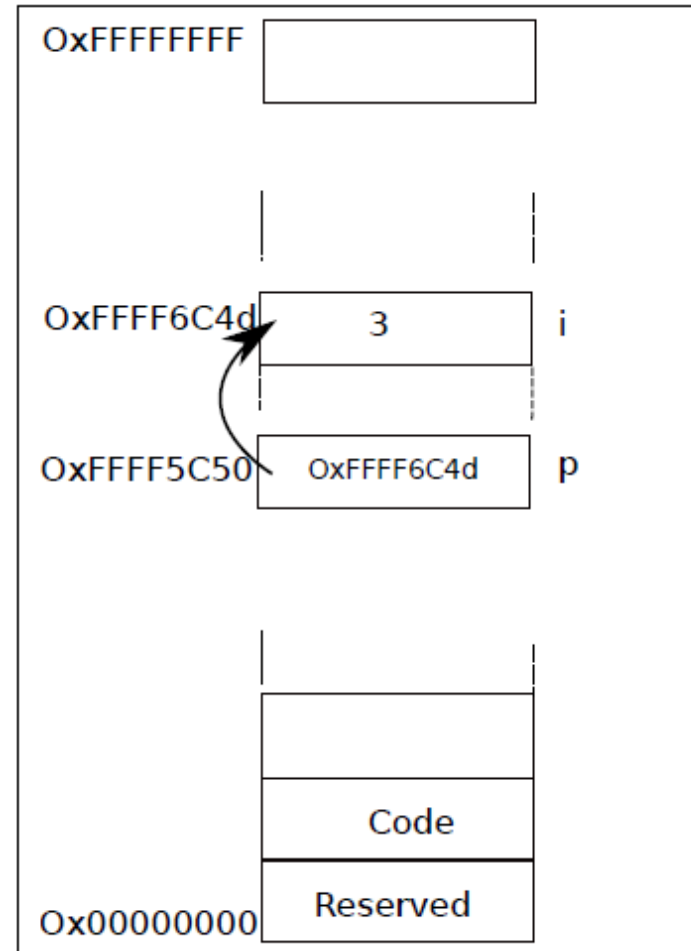
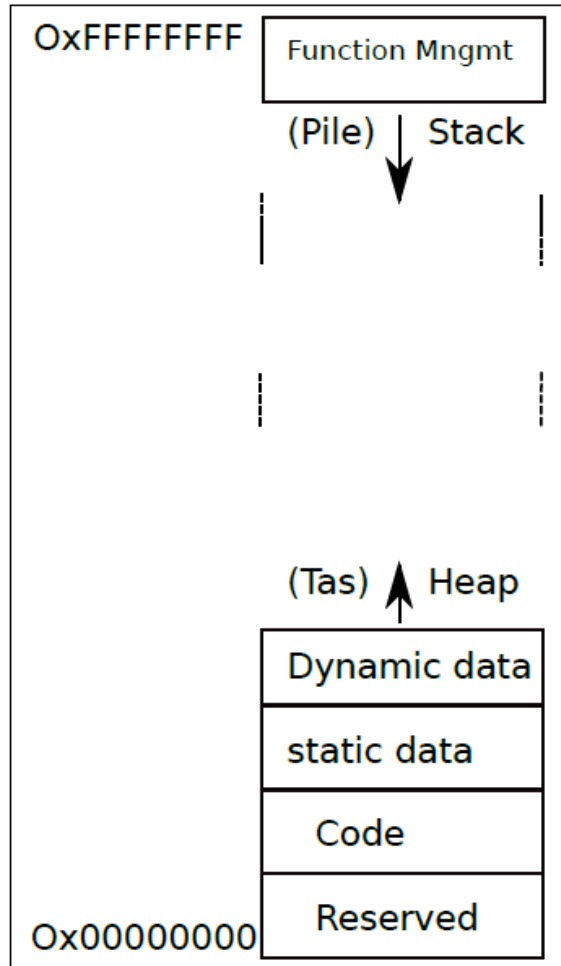
### Pile d'appel = stack

Zone contigue de la mémoire :

- lieu où sont empilés des éléments lors de l'appel d'une fonction
- contient des variables locales
- un prgm peut en posséder en général 1 seule



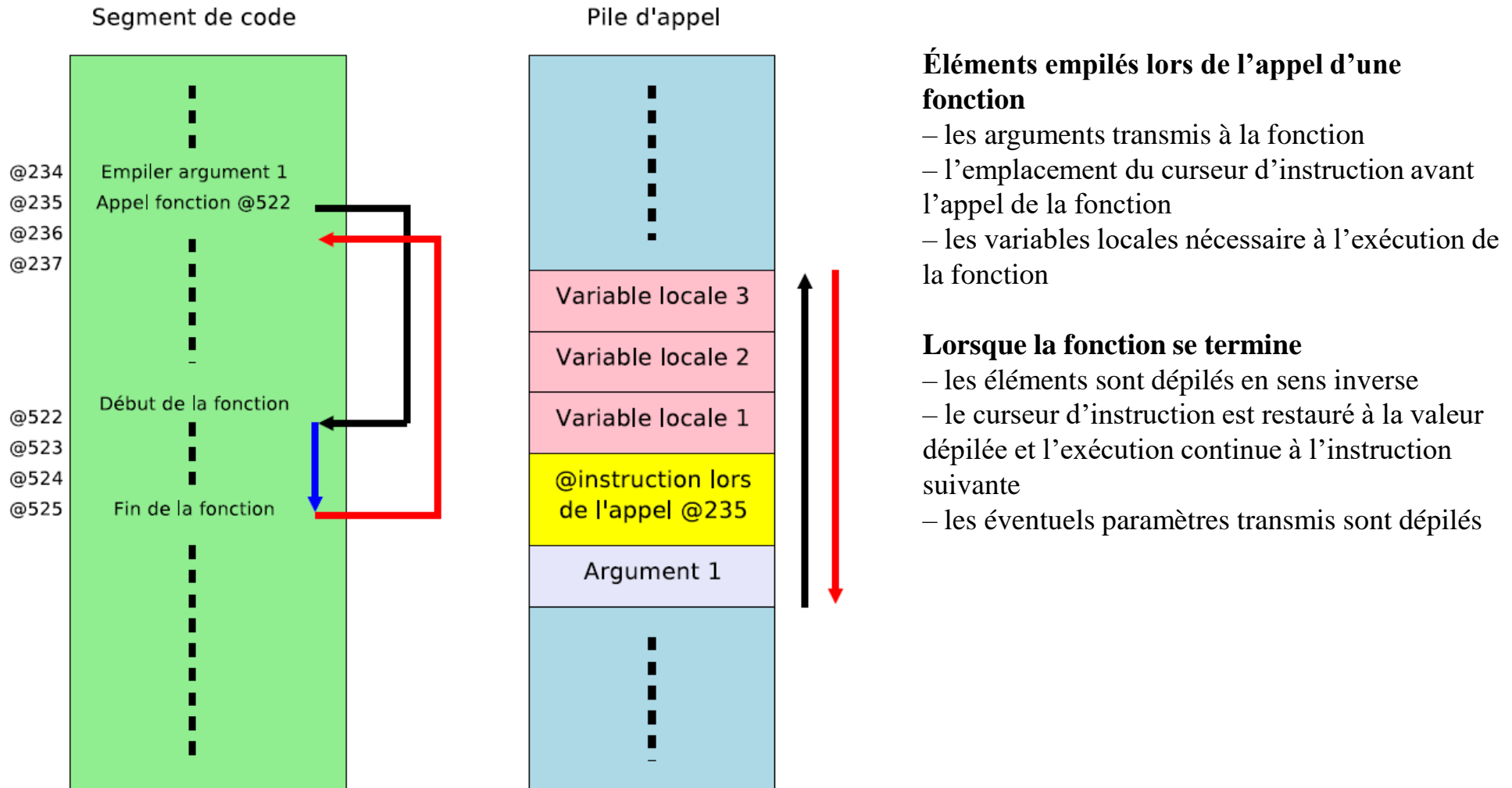
## Gestion de la mémoire



# Introduction



## Gestion de la mémoire

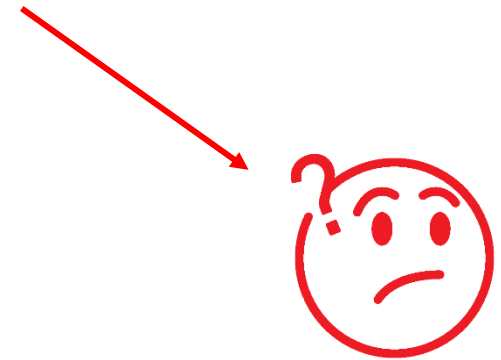


# Rappels des notions de base

---

## CF Cours Programmation C niveau 1

On survole avec quelques focus...





# On se réveille...

---



- Que dire sur les Variables et Types ?
- Modificateurs de type ?

# Rappels des notions de base

## Éléments généraux de syntaxe

### Mots réservés ISO C (<https://learn.microsoft.com/fr-fr/cpp/c-language/c-keywords?view=msvc-170>)

- Initialement 32 mots clés
- Des ajouts selon évolution de la norme ISO

Sensibilité à la casse!



### Mots clés C Standard

Le langage C utilise les mots clés suivants :

<code>auto</code>	<code>extern</code>	<code>short</code>	<code>while</code>
<code>break</code>	<code>float</code>	<code>signed</code>	<code>_Alignas</code> <sup>2, a</sup>
<code>case</code>	<code>for</code>	<code>sizeof</code>	<code>_Alignof</code> <sup>2, a</sup>
<code>char</code>	<code>goto</code>	<code>static</code>	<code>_Atomic</code> <sup>2, b</sup>
<code>const</code>	<code>if</code>	<code>struct</code>	<code>_Bool</code> <sup>1, a</sup>
<code>continue</code>	<code>inline</code> <sup>1, a</sup>	<code>switch</code>	<code>_Complex</code> <sup>1, b</sup>
<code>default</code>	<code>int</code>	<code>typedef</code>	<code>_Generic</code> <sup>2, a</sup>
<code>do</code>	<code>long</code>	<code>union</code>	<code>_Imaginary</code> <sup>1, b</sup>
<code>double</code>	<code>register</code>	<code>unsigned</code>	<code>_Noreturn</code> <sup>2, a</sup>
<code>else</code>	<code>restrict</code> <sup>1, a</sup>	<code>void</code>	<code>_Static_assert</code> <sup>2, a</sup>
<code>enum</code>	<code>return</code>	<code>volatile</code>	<code>_Thread_local</code> <sup>2, b</sup>

<sup>1</sup> Mots clés introduits dans ISO C99.

<sup>2</sup> Mots clés introduits dans ISO C11.

# Rappels des notions de base

## Éléments généraux de syntaxe

### Caractères :

- Table ASCII  
(American Standard Code for Information Interchange)
- Codage des caractères sur 8 bits
- Type de données utilisé : char
- Permet de manipuler un caractère

Exemple : 'a' ou 'A'

Mais aussi son code Ascii, soit une valeur entière

Exemple : 97 ou 65

- Expression arithmétique : 'a' – 'A'

## ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	~
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[END OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1110000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1110001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1110100	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1110101	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111000	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111001	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111100	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111101	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

# Rappels des notions de base

## Éléments généraux de syntaxe

### Caractères spéciaux

- Définis par une séquence d'échappement  
    \lettre
- Tabulation, saut de ligne,...



Séquence	Code ASCII (hexadécimal)	Caractère
\a	\$07	Bip sonore
\b	\$08	Effacement arrière
\f	\$0C	Saut de page
\n	\$0A	Changement de ligne
\r	\$0D	Retour chariot
\t	\$09	Tabulation horizontale
\v	\$0B	Tabulation verticale
\\	\$5C	\
\'	\$2C	'
\"	\$22	"
\?	\$3F	?

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Notion de casse

Le langage C est **sensible à la casse**, ce qui revient à **différencier les minuscules des majuscules**. Aussi, un identificateur, composé des mêmes lettres, mais orthographié avec des combinaisons différentes de minuscules et de majuscules ne fera pas référence au même objet.

MaVariable, mavariable, maVariable sont des identificateurs différents!

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables

#### Terminologie :

**Variable** : zone de stockage / emplacement mémoire d'un objet

**Identificateur** : nom associé à la zone mémoire associée à une variable  $\Leftrightarrow$  nom de la variable

#### Déclaration : spécifier le type d'une variable

**type** identificateur;

**type** identificateur1, identificateur2 ;

**Définition** : réaliser la déclaration d'une variable **et** l'affectation d'une d'une valeur initiale

**type** identificateur = valeur;

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables

Le type d'une variable induit :

La plage des valeurs min et max pouvant être stockées au sein de la variable

Les opérateurs pouvant lui être appliqués

Portée :

Declarations are bound to the scope in which they appear

Déclaration versus Définition :

Declarations specify identifiers, whereas definitions specify objects

An object is defined at the same time it is initialized

Each object must have exactly **one** definition

(Ref : Jens Gustedt, Modern C. Manning 2019, 9781617295812. hal-02383654)

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables

### Identificateurs : syntaxe

- Premiers caractère : **jamais un chiffre**
- Pas d'espace, ni de caractère accentué
- Les mots réservés ne sont pas admis
- Taille max ISO C : 63 caractères

`identifieur`:

`nondigit`

`identifieur nondigit`

`identifieur digit`

`nondigit` : l'un des éléments suivants :

`_ a b c d e f g h i j k l m n o p q r s t u v w x y z`

`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`

`digit` : l'un des éléments suivants :

`0 1 2 3 4 5 6 7 8 9`



# Rappels des notions de base

## Éléments généraux de syntaxe

### Les variables

Typage d'une variable : type numérique

Type	Mots réservés	Taille minimale (octets)
Entier	<code>_Bool</code>	1
	<code>char</code>	1
	<code>short</code>	2
	<code>int</code>	2
Réel	<code>float</code>	4
	<code>double</code>	8
Sans type	<code>void</code>	

La norme définit uniquement le **nombre minimal d'octets** pour stocker en mémoire une variable dans un type donné.

Valeur obtenue avec l'opérateur `sizeof`  
Exemple :  
`sizeof(int)`

Pour le type `_Bool` :

- Défini à partir du C99
- Nécessite l'inclusion du fichier `stdbool.h`

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables



#### Typage d'une variable : modificateur de type

Attribut permettant de préciser la représentation des données :

- **signed** : représentation signée (type entier **uniquement**)
  - Sauf spécification, modificateur par défaut
- **unsigned** : représentation non signée (type entier **uniquement**)
  - Uniquement des valeurs supérieures ou égales à 0
- **long** : pour étendre l'intervalle des valeurs possibles (type entier **et** double)

# Rappels des notions de base

## Éléments généraux de syntaxe

### Les variables

Type entier : domaine des valeurs minimales

- Définition au sein du fichier d'en tête **limits.h**

```
limits.h
1  /**
2   * This file has no copyright assigned and is placed in the Public Domain.
3   * This file is part of the mingw-w64 runtime package.
4   * No warranty is given; refer to the file DISCLAIMER.PD within this package.
5   */
6  #include <crtdefs.h>
7
8  #ifndef _INC_LIMITS
9  #define _INC_LIMITS
10
11 /*
12  * File system limits
13  *
14  * NOTE: Apparently the actual size of PATH_MAX is 260, but a space is
15  *       required for the NUL. TODO: Test?
16  * NOTE: PATH_MAX is the POSIX equivalent for Microsoft's MAX_PATH; the two
17  *       are semantically identical, with a limit of 259 characters for the
18  *       path name, plus one for a terminating NUL, for a total of 260.
19  */
20 #define PATH_MAX 260
21
22 #define CHAR_BIT 8
23 #define SCHAR_MIN (-128)
24 #define SCHAR_MAX 127
25 #define UCHAR_MAX 0xff
26
27 #ifdef __CHAR_UNSIGNED__
28 #define CHAR_MIN 0
29 #define CHAR_MAX UCHAR_MAX
30 #else
31 #define CHAR_MIN SCHAR_MIN
```

# Rappels des notions de base

## Éléments généraux de syntaxe

### Les variables



### Type entier : domaine des valeurs minimales

Type	Identificateur	Taille minimale (octets)	Domaine minimal
Caractère signé	char	1	-127 à 127 ou -128 à 127
	signed char	1	
Caractère non signé	unsigned char	1	0 à 255
Entier court signé	short	2	-32767 à 32767 ou -32788 à 32767
	signed short	2	
	short int	2	
	signed short int	2	
Entier court non signé	Unsigned short	2	0 à 65535
	unsigned short int	2	
Entier signé	int	2	-32767 à 32767 ou -32788 à 32767
	signed int	2	
Entier non signé	unsigned int	2	0 à 65535
Entier long signé	long int	4	-2147483647 à 2147483647 ou -2147483648 à 2147483647
	signed long int	4	
Entier long non signé	unsigned long	4	0 à 4294967295
	unsigned long int	4	

# Rappels des notions de base

## Eléments généraux de syntaxe

### Les variables Type entier

- Possibilité d'utiliser des types entiers dont la taille est spécifiée et définie quelque soit le contexte.
- Supporté depuis le C99.
- Définition au sein du fichier d'en tête `stdint.h`

Spécifieur	Signing	Bits	Bytes	Minimum Value	Maximum Value
<code>int8_t</code>	Signed	8	1	$-2^7$ which equals $-128$	$2^7 - 1$ which is equal to 127
<code>uint8_t</code>	Unsigned	8	1	0	$2^8 - 1$ which equals 255
<code>int16_t</code>	Signed	16	2	$-2^{15}$ which equals $-32768$	$2^{15} - 1$ which equals 32767
<code>uint16_t</code>	Unsigned	16	2	0	$2^{16} - 1$ which equals 65535
<code>int32_t</code>	Signed	32	4	$-2^{31}$ which equals $-2147483648$	$2^{31} - 1$ which equals 2147483647
<code>uint32_t</code>	Unsigned	32	4	0	$2^{32} - 1$ which equals 4294967295
<code>int64_t</code>	Signed	64	8	$-2^{63}$ which equals $-9223372036854775808$	$2^{63} - 1$ which equals 9223372036854775,807
<code>uint64_t</code>	Unsigned	64	8	0	$2^{64} - 1$ which equals 18446744073709551615

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables

Type entier : `char`

#### - Caractère :

Il n'existe pas de type caractère à proprement parlé ;

Un caractère sera donc représenté par le type `char`, qui est un type entier ;

Il est possible d'effectuer des opérations arithmétiques avec les caractères ;

Dualité de représentation :

'a' représente le caractère a minuscule

'a' représente le code ASCII du caractère a minuscule

#### - Chaînes de caractères :

Il n'y a pas de type explicite pour manipuler les chaînes de caractères ;

Convention de représentation des chaînes de caractères : tableau

# Rappels des notions de base

## Éléments généraux de syntaxe

### Les variables

#### Type réel : domaine des valeurs minimales

Type	Identificateur	Taille minimale (octets)	Domaine minimal
Réel	float	4	Précision $\geq 6$ chiffres décimaux au minimum 3.4E-38 à 3.4E+38
	double	8	Précision $\geq 15$ chiffres décimaux au minimum et une étendue de 1.7E-307 à 1.7E+308
	long double	10	Précision $\geq 17$ chiffres décimaux au minimum 3.4E-4932 à 3.4 <sup>E</sup> +4932

Le fichier d'en-tête **float.h** définit les plages de valeurs valides pour les types réels

Remarque : le codage binaire d'un nombre réel n'est pas toujours exact.

```
d:\Documents\ProgrammationLangageC\build-Debug\bin\CodagedesReels.exe
Valeur de x_f : 0.10000000149011611938
Valeur de x_dble : 0.10000000000000000555
Valeur de x_ldble : 0.100000000000000005551115123126

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.031 (MM:SS.MS)
Press any key to continue...
```



# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables

Déclaration : `<modificateur> <type> Id_Var1, Id_Var2;`

`int i, j, k;`

`char c1, c2;`

`unsigned int Nb_elements_tab;`

`float resultat;`

Définition : `<modificateur> <type> Id_Var1 = valeur1, Id_Var2 = valeur2;`

`double pi = 3.14159;`

`unsigned char c = 'E', c2 = '3';`



# On se réveille...

---



- Portée d'une variable ?
- static ?
- const ?
- extern ?

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables

#### Portée d'une variable

- Définie la portion de code où une variable et son contenu sont accessibles ;
  - Notion qui s'applique quel que soit le typage de(s) variable(s) ;
  - La portée d'une variable débute à la fin de sa déclaration.

Portée peut être considérée :

- Au sein d'un bloc { } -> variable **locale**
- Au sein d'un fichier -> variable **globale**
- Au sein de plusieurs fichiers -> **partage** d'un variable globale

# Rappels des notions de base

## Éléments généraux de syntaxe



### Les variables

### Portée d'une variable

Illustration : variable locale et globale

Compilation : erreur

```
PorteeVariable\main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief Illustrer la notion de portée d'une variable
6   * -> variable locale et globale
7   */
8
9  int Ma_variable_globale = 100; // Variable globale
10
11 void affiche_var (void);
12
13 void affiche_var (void) {
14     printf ("2. Ma_variable = %d\n", Ma_variable);
15     printf ("3. Ma_variable_globale = %d\n", Ma_variable_globale);
16 }
17
18 int main(void) {
19     int Ma_variable = 10; // Variable locale
20                          // Portée limitée à la fonction main
21
22     printf ("1. Ma_variable = %d\n", Ma_variable);
23     affiche_var ();
24     printf ("3. Ma_variable_globale = %d\n", Ma_variable_globale);
25     return EXIT_SUCCESS;
26 }
```

d:/Documents/ProgrammationLangageC/PorteeVariable/main.c:14:38: error: 'Ma\_variable' undeclared (first use in this function)

```
14 | printf ("2. Ma_variable = %d\n", Ma_variable);
```

```
|           ^~~~~~
```

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables

#### Portée d'une variable

Variable globale partagée entre plusieurs fichiers source

Déclaration ou définition de la variable au sein d'un fichier : `int Nb = 0;`

Cette déclaration / définition s'effectue **une seule fois**

Mécanisme de réservation mémoire

Au sein des autres fichiers : **`extern int Nb;`**

Permet d'indiquer au compilateur que la variable Nb est définie dans un autre fichier

# Rappels des notions de base

---

## Éléments généraux de syntaxe



### Les variables

#### Modificateurs appliqués sur une variable

Lors de sa déclaration, toute variable peut se voir appliquer des modificateurs qui permettent de modifier :

Le mode d'accès à la variable -> mot réservé **const**

Le mode de stockage en mémoire de la variable -> mots réservés **static**, **extern**

**<modificateur>** **<type>** **identificateur = valeur;**

# Rappels des notions de base

## Éléments généraux de syntaxe

### Les variables

#### Modificateurs appliqués sur une variable : `const`

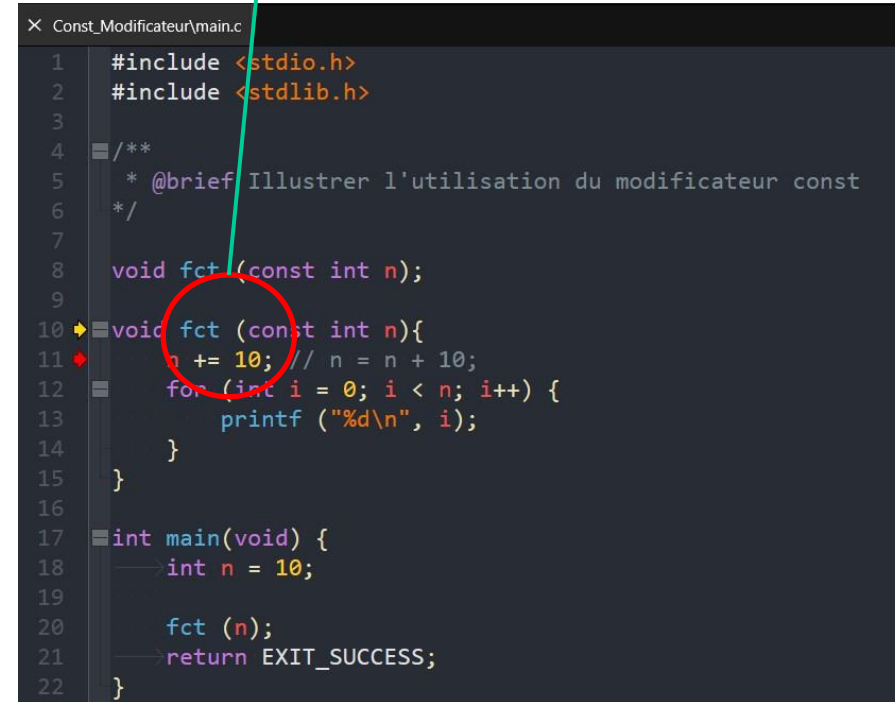
Définir une variable dont le contenu n'est plus modifiable à l'issue de l'affectation.

Permet d'introduire un mécanisme de protection du contenu d'une variable ou d'un paramètre de fonction.

```
const double pi = 3.14159;
```

```
char * strcat( char * destination, const char * source );
```

Avertissement et erreur au sein de l'IDE Codelite



```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief Illustrer l'utilisation du modificateur const
6   */
7
8  void fct( const int n);
9
10 void fct (const int n){
11     n += 10; // n = n + 10;
12     for (int i = 0; i < n; i++) {
13         printf ("%d\n", i);
14     }
15 }
16
17 int main(void) {
18     int n = 10;
19
20     fct (n);
21     return EXIT_SUCCESS;
22 }
  
```

#### Erreur lors de la compilation :

```

/ProgrammationLangageC/Const_Modificateur/main.
c:11:7: error: assignment of read-only parameter 'n'
  
```

```

11 |   n += 10; // n = n + 10;
    |     ^~
  
```

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les variables

Modificateurs appliqués sur une variable : `static`

Deux usages :

#### Cas d'une variable locale

- Sa durée de vie s'étend à la durée d'exécution du programme.

Exemple : conserver le contenu d'une variable entre 2 appels d'une fonction.

#### Cas d'une variable globale

- La portée de la variable est limitée au fichier dans lequel elle est définie.

# Rappels des notions de base



## Eléments généraux de syntaxe

### Les variables

### Modificateurs appliqués sur une variable : **static**

```

Static_Modificateur\main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief Illustrer l'utilisation du modificateur static
6   * Without static
7   */
8
9  void fnct (void);
10
11 void fnct (void) {
12     int compteur = 0 ;
13     compteur = compteur + 1 ;
14     printf ("Valeur du compteur : %d\n", compteur) ;
15 }
16
17 int main (void) {
18     for (int i = 0; i < 10; i++) fnct ();
19     return EXIT_SUCCESS;
20 }
21
22 d:\Documents\ProgrammationLangageC\build-Debug\bin\Static_Modificateur.exe
23
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1
Valeur du compteur : 1

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.047 (MM:SS.MS)
Press any key to continue...

```

```

Static_Modificateur\main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief Illustrer l'utilisation du modificateur static
6   * With static
7   */
8
9  void fnct (void);
10
11 void fnct (void) {
12     static int compteur = 0 ;
13     compteur = compteur + 1 ;
14     printf ("Valeur du compteur : %d\n", compteur) ;
15 }
16
17 int main (void) {
18     for (int i = 0; i < 10; i++) fnct ();
19     return EXIT_SUCCESS;
20 }
21
22 d:\Documents\ProgrammationLangageC\build-Debug\bin\Static_Modificateur.exe
23
Valeur du compteur : 1
Valeur du compteur : 2
Valeur du compteur : 3
Valeur du compteur : 4
Valeur du compteur : 5
Valeur du compteur : 6
Valeur du compteur : 7
Valeur du compteur : 8
Valeur du compteur : 9
Valeur du compteur : 10

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.062 (MM:SS.MS)
Press any key to continue...

```



# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les constantes

#### Les constantes littérales

- Valeur numériques directement introduites dans le code ;
- Exemples :
  - 100 est une constante **entière** ;
  - 100. ou 100.0 sont des constantes **réelles** ;
- Mise en œuvre de mécanismes de conversion de type.

#### Les constantes symboliques

Représentée par un nom

Syntaxe :

```
#define NOM valeur
```

- Par convention, NOM est en majuscules
- Par convention, la définition de constantes symbolique s'effectue en début de fichier source
- Exemple :
  - #define NBLIGNE 10

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les constantes

La valeur effective d'une constante réelle peut être différente de sa valeur littérale (codage de 0.2 par exemple)

Constantes numériques en hexadécimal : syntaxe

**0xnnnn** : nnnn est exprimé en hexadécimal et n appartient à {0,1,...,9,A-F,a-f}

Exemple : 0xFFAB

Constantes numériques en binaire : syntaxe

**0bnn...n** : n appartient à {0,1}

Exemple : 0b01111111 (correspond au codage binaire de 127)

Une constante numérique, sous forme décimale, hexadécimale ou binaire, est convertie dans le type de données le plus proche qui permet de la représenter correctement.

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les constantes

Il est possible de forcer le typage d'une constante numérique en utilisant un *suffixe* (**U**, **L** ou **F**).

Exemple :

```
#define MACONSTANTE 32 L
```

Forcer la représentation dans le type **long int**

Type	Format numérique	Suffixe	Exemple
<b>int</b>	entier	aucun	31415
<b>unsigned int</b>	entier	U	31415U
<b>long int</b>	entier	L	31415L
<b>unsigned long int</b>	entier	UL	31415UL
<b>float</b>	réel	F	3.1415F
<b>double</b>	réel	aucun	3.1415
<b>long double</b>	réel	L	3.1415L

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les opérateurs

Définitions préalables

**lvalue** : correspond à une **variable**.

**rvalue** : correspond à une **expression**, une constante ou la valeur renvoyée par une fonction.

Opérateur **unaire** : n'acceptent qu'un seul opérande.

Opérateur **binaire** : acceptent 2 opérandes.

Un même symbole peut représenter des opérateurs différents

-> dépendance au contexte

# Rappels des notions de base

## Eléments généraux de syntaxe

### Les opérateurs

Catégorie d'opérateurs	Description	Opérateur
Affectation	Affecter à une variable (lvalue) la valeur d'une expression (rvalue)	=
Arithmétique	Calculs arithmétiques classiques	+, -, /, *, %
Incrémentation Décrémentation	Forme condensé d'opérateurs arithmétiques	++, --
Manipulation de bits	Combinaison logique bit à bit	~, &, ^, ~
Décalage	Décalage de bits	<<, >>
Relationnels	Comparaison de valeurs numériques	==, <, <=, >, >=, !=
Logiques	Lier des expressions logiques	, &&, !
Cast	Conversion explicite de type	(type)
Autres	Opérateur conditionnel, séquentiel et sizeof	?, , sizeof

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les opérateurs

**Affectation** lvalue = rvalue;

Mécanismes de conversion de type : la rvalue est toujours convertie dans le type de la lvalue (conversion implicite)

- Conversion sans perte d'information

- o Lorsque le type de la rvalue est contenu ou égal à celui de la lvalue ;
- o Lorsque rvalue et lvalue sont toutes les 2 dans une représentation signée ou non signée.

Hiérarchie des types est donnée ci-dessous :

char < short ≤ int ≤ long < float < double ≤ long double

-----> Conversion sans perte

- Conversion avec perte d'information

- Survient lorsque le type de la lvalue est hiérarchiquement inférieur à celui de la rvalue.

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les opérateurs

#### Opérateurs arithmétiques

Opérateur	Symbole	Description	Syntaxe
Addition	+	Somme de la valeur de 2 opérandes	$x + y$
Soustraction	-	Soustraction de la valeur de 2 <sup>nd</sup> e opérande à celle de la première	$x - y$
Multiplication	*	Multiplication de la valeur de 2 opérandes	$x * y$
Division	/	Division de la valeur de 1 <sup>ère</sup> opérande par celle de la seconde	$x / y$
Modulo	%	Reste de la division entière	$x \% y$
Inversion de signe	-	Inversion de signe	$- x$

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les opérateurs

#### Opérateurs arithmétiques

- Opérateur de division /
  - Le résultat dépend du type des opérandes ;
  - Si les deux opérandes sont du type entier, alors le résultat est entier (division euclidienne).
  - Illustration :

```
float x = 1 / 3 ;           /* x = 0.0 */  
float x = 1. / 3 ;        /* x = 0.33333333 */  
float x = 1 /3. ;        /* x = 0.33333333 */
```

- Opérateur modulo %
  - Les 2 opérandes doivent être dans un **type entier** uniquement



# Rappels des notions de base

## • Éléments généraux de syntaxe

### Les opérateurs

#### Incrémentation Décrémentement

#### Contexte :

- Basée sur les opérateurs `++` et `--`
- Opérateurs unaires
- Opérandes entières ou réelles
- Incrémenter ou décrémenter de 1 le contenu d'une variable

#### Syntaxe :

- Pré incrémentation : `++ x`     `/* x←x + 1 */`
- Pré décrémentement : `-- x`     `/* x←x - 1 */`
- Post incrémentation : `x ++`     `/* x←x + 1 */`
- Post décrémentement : `x --`     `/* x←x - 1 */`

```

IncDec_opérateur\main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief Illustrer l'utilisation des opérateurs ++ et --
6   */
7
8  int main(void) {
9
10     int x = 3, y = 7;
11     double z = 10.2;
12
13     //Post incrémentation
14     printf ("1. x = %d", x++); printf ("\tx = %d\n", x);
15     //Pré incrémentation
16     printf ("2. y = %d", ++y); printf ("\ty = %d\n", y);
17     // ++ sur variable réelle
18     printf ("1. x = %lf", ++z);
19     return EXIT_SUCCESS;
20 }
21
22
23 d:\Documents\ProgrammationLangageC\build-Debug\bin\IncDe...
23 1. x = 3      x = 4
24 2. y = 8      y = 8
25 1. x = 11.200000
==== Program exited with exit code: 0 ====
Time elapsed: 000:00.047 (MM:SS.MS)
Press any key to continue...

```

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les opérateurs

#### Décalage << et >>

Les 2 opérandes doivent être dans un type entier uniquement.

Le résultat du décalage est exact si :

1. Il n'y a pas de débordement de type
2. L'opérande sur laquelle le décalage est appliquée est positive

Typage de données à privilégier : **unsigned**

Effectuer n décalages de bits vers la droite : division par  $2^n$

$x \gg n$

Effectuer n décalages de bits vers la gauche : multiplication par  $2^n$

$x \ll n$

# Rappels des notions de base

---

## Éléments généraux de syntaxe

### Les opérateurs

Relationnels et Logiques : préalable

### Notion de booléen

Le langage C impose de considérer que :

- Faux : valeur nulle
- Vrai : toute valeur  $\neq$  de 0

En considérant le type bool (`#include <stdbool.h>`) :

- Faux : false
- Vrai : true

# Rappels des notions de base

## Eléments généraux de syntaxe



### Les opérateurs Relationnels

Opérateur	Symbole	Description	Syntaxe
Egalité	<code>==</code>	Test d'égalité	<code>x == y</code>
Inférieur	<code>&lt;</code>	Test d'infériorité	<code>x &lt; y</code>
Supérieur	<code>&gt;</code>	Test de supériorité	<code>x &gt; y</code>
Inférieur ou égal	<code>&lt;=</code>	Test d'égalité et de supériorité	<code>x &lt;= y</code>
Supérieur ou égal	<code>&gt;=</code>	Test d'égalité et d'infériorité	<code>x &gt;= y</code>
Différent	<code>!=</code>	Test de différence	<code>x != y</code>

- Ne pas confondre l'opérateur d'affectation `=` avec l'opérateur de test d'égalité `==`
- Test d'égalité de 2 **réels** :  $(x = y)$  ?
  - Ne jamais utiliser les opérateurs `==` ou `!=`
  - Préférer en remplacement  $\text{fabs}(x - y) < \varepsilon$  ou  $\text{fabs}(x - y) > \varepsilon$

# Rappels des notions de base

## Éléments généraux de syntaxe



### Les opérateurs Logiques

Opérateur	Symbole	Description	Syntaxe
et	<b>&amp;&amp;</b>	Les expressions sont elles vraies	Exp1 <b>&amp;&amp;</b> Exp2
ou	<b>  </b>	Une des expressions est elle vraie	Exp1 <b>  </b> Exp2
non	<b>!</b>	L'expression est elle fausse	<b>!</b> Exp

Évaluation des expressions avec les opérateurs **&&** et **||**

Exp1 **&&** Exp2 : Expr2 n'est évaluée que si Expr1 est vraie

Exp1 **||** Exp2 : Expr2 n'est évaluée que si Expr1 est fausse

# Rappels des notions de base

## Éléments généraux de syntaxe

### Les opérateurs

#### Affectations étendues

$lvalue \text{ opérateur} = rvalue$  ;  $\Leftrightarrow lvalue = lvalue \text{ opérateur} rvalue$ ;

Affectation et ...	Symbole	Equivalence	Exemple
Addition	<b>+=</b>	$x = x + y$	$x += y$
Soustraction	<b>-=</b>	$x = x - y$	$x -= y$
Multiplication	<b>*=</b>	$x = x * y$	$x *= y$
Division	<b>/=</b>	$x = x / y$	$x /= y$
Modulo	<b>%</b>	$x = x \% y$	$x \% = y$
Décalage à droite	<b>&gt;&gt;=</b>	$x = x >> y$	$x >> = y$
Décalage à gauche	<b>&lt;&lt;=</b>	$x = x << y$	$x << = y$
Complémentation	<b>^=</b>	$x = x \wedge y$	$x \wedge = y$
Et bit à bit	<b>&amp;=</b>	$x = x \& y$	$x \& = y$
Ou bit à bit	<b> =</b>	$x = x   y$	$x   = y$

# Rappels des notions de base

---

## Éléments généraux de syntaxe



### Les opérateurs

#### Conversion explicite : **cast**

**Rappel :** au sein des expressions arithmétiques, mécanismes de conversion de type implicite.

**Cast :** permet de forcer explicitement la représentation du contenu d'une variable dans un type particulier

Syntaxe :

(**type**) Identificateur

Exemple :

```
int j = 32;
```

```
d = (double) j    → conversion explicite de j en représentation réelle, soit 32.0
```

# Rappels des notions de base

## Éléments généraux de syntaxe

### Les opérateurs

#### sizeof

1. sizeof (*type*) renvoie la taille en **octets** d'un type de données
2. sizeof (*Ident*) renvoie la taille en **octets** d'une variable définie par l'identificateur *Ident*
3. Permet de réaliser du code portable

```
Sizeof\main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief Opérateur sizeof
6   */
7
8  int main(void) {
9      int tab_int [10] = {0};
10
11     printf("Type char : %llu octet(s)\n", sizeof (char));
12     printf("Type int : %llu octet(s)\n", sizeof (int));
13     printf("Type long int : %llu octet(s)\n", sizeof (long int));
14     printf("Type double : %llu octet(s)\n", sizeof (double));
15     printf("Tableau de 10 int : %llu octet(s)\n", sizeof (tab_int));
16
17     return EXIT_SUCCESS;
18 }
19
```

```
d:\Documents\ProgrammationLangageC\build-Debug\bin\Sizeof.exe
Type char : 1 octet(s)
Type int : 4 octet(s)
Type long int : 4 octet(s)
Type double : 8 octet(s)
Tableau de 10 int : 40 octet(s)

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.063 (MM:SS.MS)
Press any key to continue...
```



# Rappels des notions de base

## Éléments généraux de syntaxe

### Le contrôle de flux

- if...else
- Switch
- boucle for,
- while,
- do... while, ...

### CF Programmation C I

```
StructureIteratives\main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief Illustration structures itératives
6   * boucle for
7   * boucle while
8   * boucle do...while
9   */
10
11 #define NB_ITERATIONS 3
12
13 int main(void) {
14
15     for (int i = 0; i < NB_ITERATIONS; i++) {
16         printf("Id_boucle_for : %d\n", i);
17     }
18
19     int i = 0;
20     while (i < NB_ITERATIONS) {
21         printf("Id_boucle_while : %d\n", i);
22         i++;
23     }
24
25     i = 0;
26     do {
27         printf("Id_boucle_do_while : %d\n", i);
28         i++;
29     } while (i < NB_ITERATIONS);
30
31     return EXIT_SUCCESS;
32 }
```

```
d:\Documents\ProgrammationLangageC\build-Debug\bin\StructureIteratives.exe
Id_boucle_for : 0
Id_boucle_for : 1
Id_boucle_for : 2
Id_boucle_while : 0
Id_boucle_while : 1
Id_boucle_while : 2
Id_boucle_do_while : 0
Id_boucle_do_while : 1
Id_boucle_do_while : 2

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.047 (MM:SS.MS)
Press any key to continue...
```

# Rappels des notions de base

---

## Eléments généraux de syntaxe



### **E/S de base**

**printf** : affichage au sein de la console

**scanf** : acquisition de paramètres depuis le clavier

Nécessité d'inclure le fichier d'en tête `stdio.h` `#include <stdio.h>`

**printf** : affichage formaté du contenu d'une ou plusieurs variables

Prototype : `int printf(const char* format, ...);`

format : chaîne de caractères

... la liste des variables dont on souhaite afficher le contenu

**scanf** : effectuer des saisies de données formatées depuis le clavier

Prototype :

`int scanf( const char *format, ... );`

# Rappels des notions de base

## Eléments généraux de syntaxe

E/S de base

printf : affichage au sein de la console

Définition du format d'affichage

`%[flags][width][.precision][size]type`

**type** : spécificateur du type de la variable dont on souhaite afficher le contenu

**width** : nombre de caractères générés

**.precision** : nombre de décimales pour les réels

**flags** : directives d'indicateurs (gestion signe + - ,...)

**size** : modificateur de longueur d'argument pour le spécificateur de conversion *type*

type

Caractère de conversion	Type de valeur reçue	Signification
<b>d</b>	<i>signed int</i> ou <i>long int</i>	affichage en base 10 d'une valeur entière
<b>o, u, x, X</b>	<i>unsigned int</i> ou <i>unsigned long int</i>	o : affichage en base 8 u : affichage en base 10 x : affichage en base 16 {0,...9, a, b, c, d, e, f} X : affichage en base 16 {0,...9, A, B, C, D, E, F}
<b>f</b>	<i>float</i> ou <i>double</i> ou <i>long double</i>	affichage en base 10 d'une valeur réelle
<b>e, E</b>	<i>float</i> ou <i>double</i> ou <i>long double</i>	affichage en notation exponentielle d'une valeur réelle
<b>c</b>	<i>int</i>	affichage en fait d'un caractère (de la valeur son code ASCII)
<b>s</b>	<i>char *</i> ou <i>signed char *</i> ou <i>unsigned char *</i>	affichage d'une chaîne de caractères
<b>p</b>	<i>void *</i>	affichage dépendant de l'implémentation (pointeur)

size

Modificateurs	Caractère de conversion	Signification
<b>l</b>	d, i	expression est du type <i>long int</i>
	u, o, x, X	expression est du type <i>unsigned long int</i>
	f	expression est du type <i>double</i>
<b>L</b>	e, E, f, g, G	expression est du type <i>long double</i>

# Rappels des notions de base

## Éléments généraux de syntaxe

Passage des paramètres par adresse -> utilisation de l'opérateur &

### E/S de base

scanf : acquisition de paramètres depuis le clavier

format : chaîne de caractères qui précise le type des données qui seront saisies

Exemple :

```
int d = 0;
```

```
float f = 0.0;
```

```
double lf = 0.0;
```

```
char S[80];
```

```
scanf (" %d %f %lf %s", &n, &f, &lf, s);
```

%d	Une donnée entière de type int
%ld	Une donnée entière de type long.
%f	Une donnée décimale de type float.
%lf	Une donnée décimale de type double.
%c	Une donnée de type caractère.
%s	Une donnée de type chaîne de caractères. Attention, tout séparateur (blanc, tabulation, retour à la ligne) interrompra la lecture.
%[characters]	Une donnée de type chaîne de caractères, constituée que de caractères parmi ceux spécifiés.
%[^characters]	Une donnée de type chaîne de caractères, constituée de tous caractères sauf les caractères spécifiés.

# On se réveille...



- Les subtilités des entrées/sorties standards en C
- printf, scanf, fgetc, getchar, putchar, fflush,...

Exercice : Faire le programme correspondant

```
d:\Documents\ProgrammationLangageC\build-Debug\bin\PrintfScanf.exe
Veillez saisir votre nom : FF
Veillez saisir une fraction (sous la forme [num/den]) : [12/24]
num : 12 den 24
FF a saisi 0.500000
```

# Rappels des notions de base

```
PrintfScanf\main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief illustration utilisation de printf et scanf
6   */
7
8  int main(void)
9  {
10     int num;
11     int den;
12     double result;
13     char buffer[80];
14
15     printf( "Veuillez saisir votre nom : " );
16     fflush( stdout );
17     scanf( "%[^\n]", buffer );
18     fgetc( stdin ); /* to delete '\n' character */
19
20     printf( "Veuillez saisir une fraction (sous la forme [num/den]) : " );
21     fflush( stdout );
22     scanf( "[%d/%d]", &num, &den );
23     printf( "num : %d den %d\n", num, den );
24     result = num / (double) den;
25
26     printf( "%s a saisi %lf\n", buffer, result );
27     return EXIT_SUCCESS;
28 }
```

```
d:\Documents\ProgrammationLangageC\build-Debug\bin\PrintfScanf.exe
Veuillez saisir votre nom : FF
Veuillez saisir une fraction (sous la forme [num/den]) : [12/24]
num : 12 den 24
FF a saisi 0.500000

==== Program exited with exit code: 0 ====
Time elapsed: 000:20.515 (MM:SS.MS)
Press any key to continue...
```

## ANNEXE : MEMO MAKEFILE GCC

---

- Gcc
  - gcc main.c -o myprg
  - gcc -c main.c + gcc main.o -o myprg
  - gcc -g main.c -o myprog + gdb ;/myprg
  - gcc -c math\_ops.c + ar rcs libmath\_ops.a math\_ops.o + gcc main.c -L. -lmath\_ops -o program
  - gcc -fPIC -c math\_ops.c + gcc -shared -o libmath\_ops.so math\_ops.o + gcc main.c -L. -lmath\_ops -o program + LD\_LIBRARY\_PATH=. ./program
  
- Make
  - % représente un modèle qui peut correspondre à n'importe quelle chaîne de caractères dans le nom de fichier. Utilisation :  
 %o : %c  
 gcc -c \$< -o \$@
  - \$@ : Nom de la cible
  - \$^ : Liste des dépendances
  - \$< : La première dépendance d'une règle.
  - SRCS = \$(wildcard \*.c) renvoie tous les fichiers .c dans le répertoire courant et les stocke dans la variable SRCS.
  - OBJS = \$( patsubst %.c, %.o, \$(SRCS) ) : substitution de pattern