

Introduction



Chapitre 1

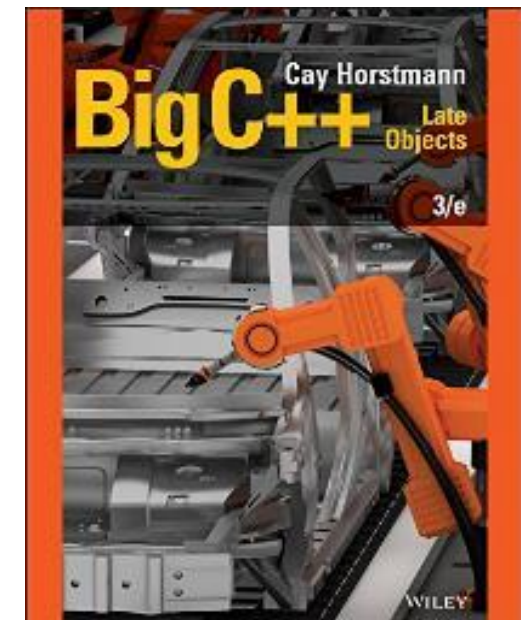
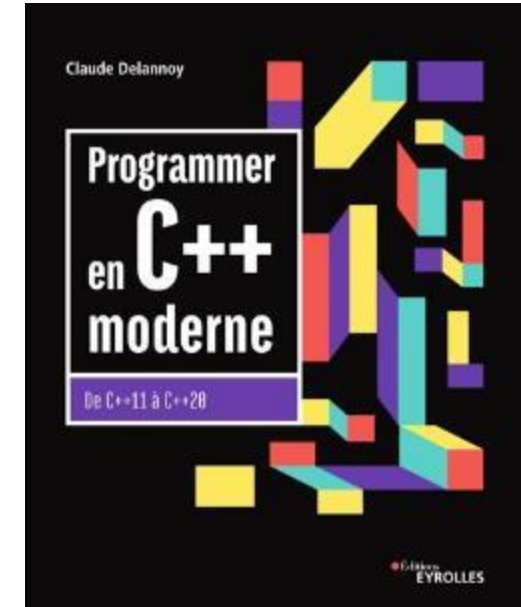
Introduction

HE^{VD} IG Références

Introduction



- *Programmer en C++ moderne*, Claude Delannoy
- *Big C++*, Cay S. Horstmann
- <http://www.cplusplus.com>
- <http://fr.cppreference.com>





1. Bref historique du C / C++



Bref historique du C / C++

- **1954** – John **Backus** (IBM) invente le premier langage de haut niveau : **FORTRAN** (**FOR**mula **TRAN**slator)
- **1958** – John **Backus**, Peter **Naur** et d'autres inventent **ALGOL** (**ALG**orithmic **O**riented **L**anguage), qui introduit la notion de bloc de code
- **1963** – Les universités de Cambridge et de Londres codéveloppent le **CPL** (**C**ombined **P**rogramming **L**angage), qui veut supporter tant la programmation scientifique que commerciale.
- **1966** – Martin **Richards** (Cambridge) crée le langage **BCPL** (**B**asic **C**ombined **P**rogramming **L**anguage), une version simplifiée de CPL pour la programmation système
- **1969** - Ken **Thompson** et Dennis **Ritchie** (Bell Labs) inventent le langage **B** qui simplifie BCPL



John Backus



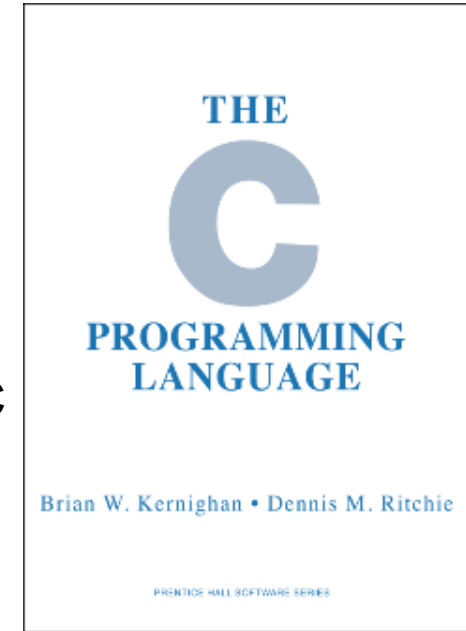
Peter Naur



Martin Richards



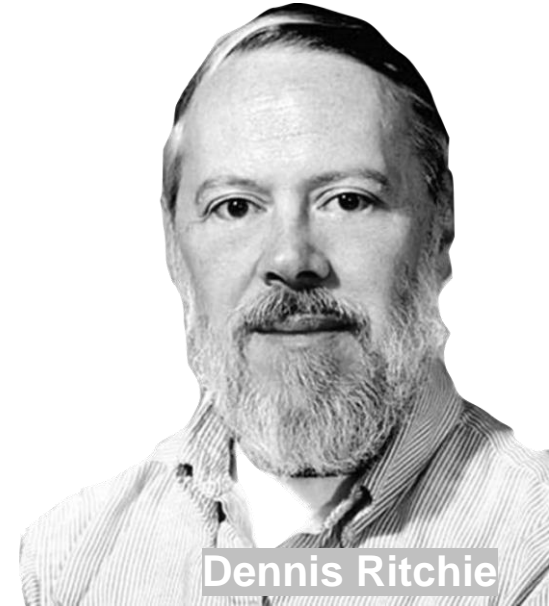
- **1972** - Ken **Thompson** et Dennis **Ritchie** traduisent **UNIX**, écrit en code machine PDP-7, pour tourner sur des PDP-11. Ils le réécrivent en langage de haut niveau et ajoutent les fonctionnalités nécessaires à B, qui devient **C**
- **1978** - Brian **Kernighan** and Dennis **Ritchie** (**K&R**) publient « *The C Programming Language* », qui devient une spécification *de facto* du langage C
- **1989** – L'American National Standards Institute publie le standard ANSI C, adopté par **ISO** en 1990
- **1999** – ISO/IEC 9899:1999



Ken Thompson



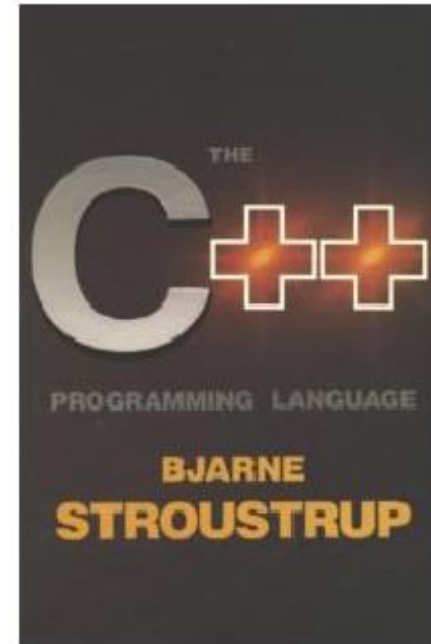
Brian Kernighan



Dennis Ritchie



- **1983** - Bjarne **Stroustrup** (AT&T) ajoute à C des fonctionnalités de Simula, un langage orienté objet conçu pour réaliser des simulations
- **1985** – *The **C++** Programming Language*
- **1998** – ISO/IEC 14882:1998 C++98
- **2003** – ISO/IEC 14882:2003 C++03
- **2011** – ISO/IEC 14882:2011 C++11
- **2014** – ISO/IEC 14882:2014 C++14
- **2017** – ISO/IEC 14882:2017 C++17
- **2020** – ISO/IEC 14882:2020 C++20





C / C++ ... aujourd'hui

- C et C++ coexistent et **évoluent** toujours
- C++ permet la programmation sous de multiples **paradigmes** comme
 - la programmation **procédurale**
 - la programmation **orientée objet**
 - la programmation **générique**
- C++ est l'un des langages de programmation les plus **populaires**, avec une grande variété de plateformes matérielles et de systèmes d'exploitation
- D'autres langages populaires comme **Java** ou **C#** s'en sont largement inspirés

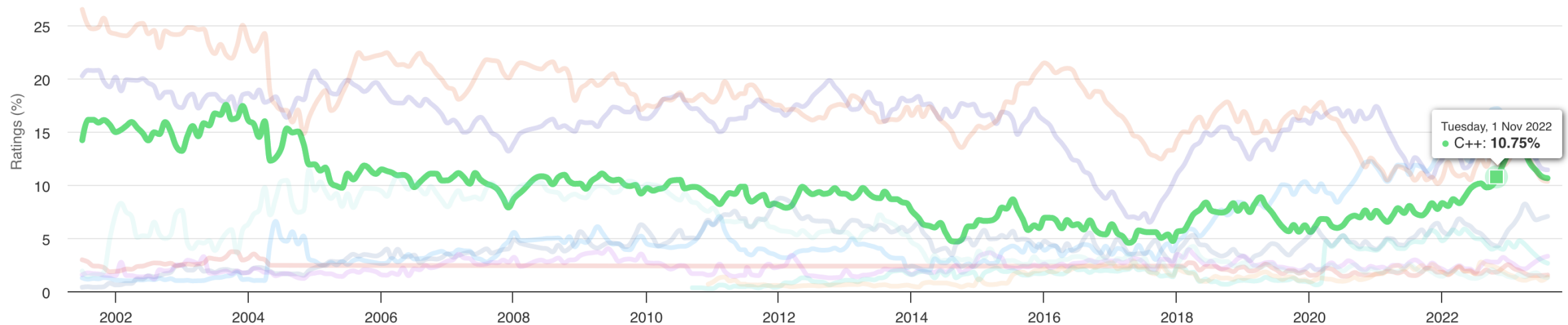
HE^{VD} IG C / C++ ... aujourd'hui

Introduction



- Source : <https://www.tiobe.com/tiobe-index/>

Aug 2023	Aug 2022	Change	Programming Language	Ratings	Change
1	1		 Python	13.33%	-2.30%
2	2		 C	11.41%	-3.35%
3	4	▲	 C++	10.63%	+0.49%
4	3	▼	 Java	10.33%	-2.14%
5	5		 C#	7.04%	+1.64%
6	8	▲	 JavaScript	3.29%	+0.89%





- **Première année**

- **Semestre 1**

- PRG1 – C++

- **Semestre 2**

- ASD – C++

- PRG2 – C

- **Deuxième année**

- **Semestre 1**

- POO – Java

- **Semestre 2**

- POA – C++



2. Premier programme C++

Hello
World!



Regardons de près un premier programme tout simple en C++

```
// entête ...
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    // intro
    cout << "Bienvenu(e)s au cours PRG1" << endl;

    // traitement ...

    // fin de programme
    cout << "fin de programme";
    return EXIT_SUCCESS;
}
```



Tout programme doit avoir une fonction principale nommée `main`

```
// entête ...
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    // intro
    cout << "Bienvenu(e)s au cours PRG1" << endl;

    // traitement ...

    // fin de programme
    cout << "fin de programme";
    return EXIT_SUCCESS;
}
```

HE^{VD} IG 1^{er} programme en C++

Introduction



Cette fonction ne prend ici pas de paramètre « () »

Note: `int main(void)`
serait faut en C++,
ok en C

```
// entête ...
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    // intro
    cout << "Bienvenu(e)s au cours PRG1" << endl;

    // traitement ...

    // fin de programme
    cout << "fin de programme";
    return EXIT_SUCCESS;
}
```


HE^{VD} IG 1^{er} programme en C++



Les instructions qui composent votre programme.
L'instruction **cout** affiche à l'écran du texte et **endl** passe à la ligne.

```
// entête ...  
#include <iostream>  
#include <cstdlib>  
using namespace std;  
  
int main() {  
    // intro  
    cout << "Bienvenu(e)s au cours PRG1" << endl;  
  
    // traitement ...  
  
    // fin de programme  
    cout << "fin de programme";  
    return EXIT_SUCCESS;  
}
```



La valeur de retour vaut typiquement 0 ou -1. La librairie `<cstdlib>` propose deux constantes `EXIT_SUCCESS` et `EXIT_FAILURE`

```
// entête ...
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    // intro
    cout << "Bienvenu(e)s au cours PRG1" << endl;

    // traitement ...

    // fin de programme
    cout << "fin de programme";
    return EXIT_SUCCESS;
}
```



Fichier(s) **d'en-tête(s)** pour utiliser des services complémentaires. Ici des entrées/sorties pour « dialoguer » avec l'utilisateur

```
// entête ...
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    // intro
    cout << "Bienvenu(e)s au cours PRG1" << endl;

    // traitement ...

    // fin de programme
    cout << "fin de programme";
    return EXIT_SUCCESS;
}
```

HE^{VD} IG 1^{er} programme en C++



Indique au compilateur d'utiliser un **espace de noms std** (standard). Pas obligatoire, mais évite d'écrire `std::cout` et `std::endl`

```
// entête ...
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    // intro
    cout << "Bienvenu(e)s au cours PRG1" << endl;

    // traitement ...

    // fin de programme
    cout << "fin de programme";
    return EXIT_SUCCESS;
}
```



Un entête permettra de **documenter très globalement** un fichier (nom du fichier, auteur, objectifs, particularités, ...)

```
// entête ...
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    // intro
    cout << "Bienvenu(e)s au cours PRG1" << endl;

    // traitement ...

    // fin de programme
    cout << "fin de programme";
    return EXIT_SUCCESS;
}
```




A noter

- Les instructions d'une fonction forment un bloc et sont placées entre accolades { ... }
- Chaque instruction se termine par un ;
- L'opérateur << transmet le contenu (int, string, ...) au flux std::cout

```
// entête ...
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    // intro
    cout << "Bienvenu(e)s au cours PRG1" << endl;

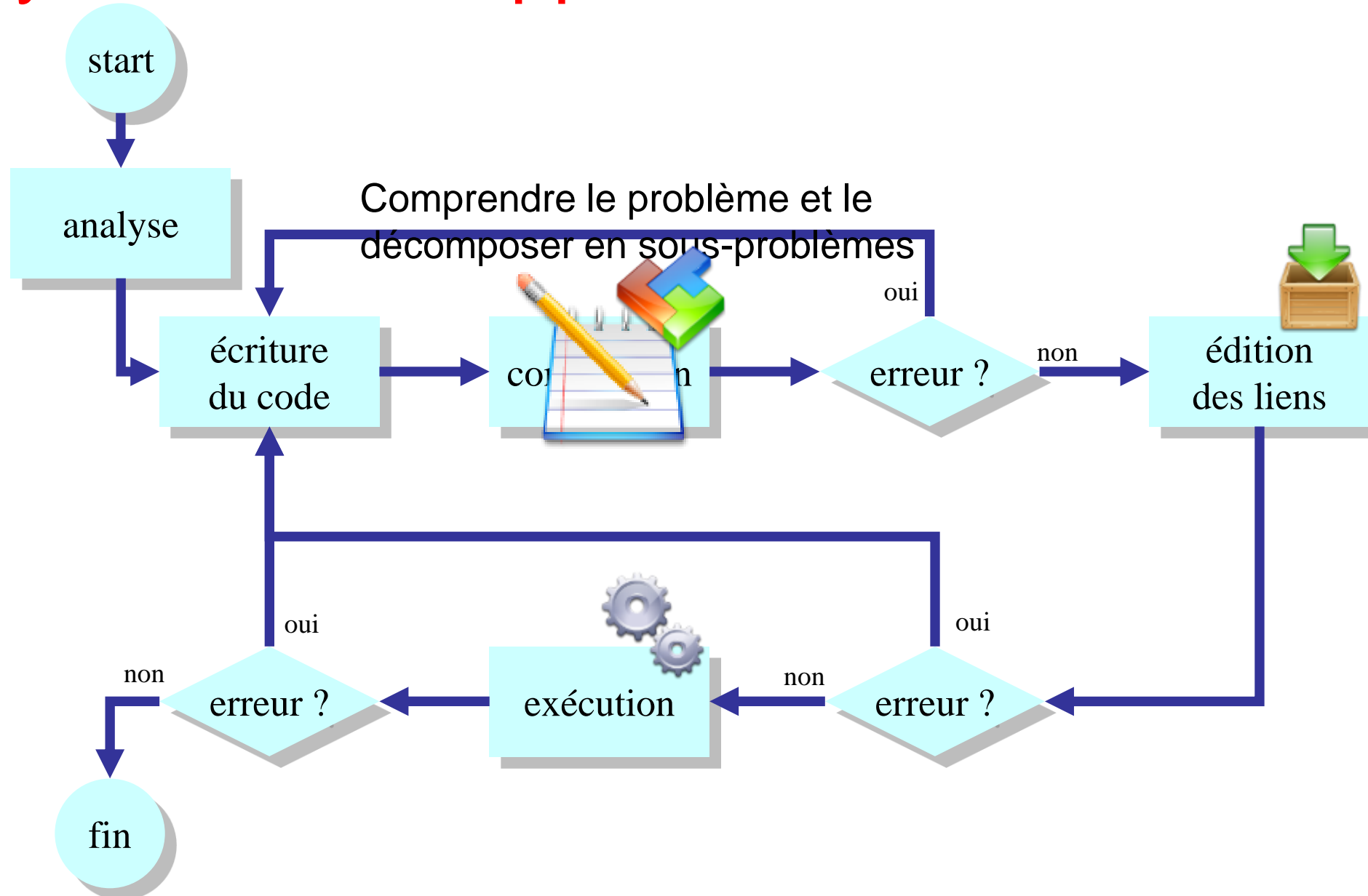
    // traitement ...

    // fin de programme
    cout << "fin de programme";
    return EXIT_SUCCESS;
}
```



3. Compilation

Cycle de développement





Compilation en ligne de commande

Pour écrire du code, un simple **éditeur de texte** suffit et la compilation peut se faire en ligne de commande.

La **mise en page** du code est déjà importante.

- Séparation visuelle (lignes vides)
- L'indentation
Préférer les espaces aux tabulations qui peuvent être interprétés différemment selon les programmes.

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main() {
    cout << "Hello World" << endl;
    return EXIT_SUCCESS;
}
```

Compilation en ligne de commande

Une fois le code écrit, il s'agit maintenant de le compiler.

```
g++ -c 01_HelloWorld.cpp
```

Cette étape créer un fichier **objet** « 01_HelloWorld.o » qui n'est pas encore directement exploitable. A défaut, les erreurs de compilation seront affichées.

Pour créer un fichier **exécutable** (.exe pour Windows), il faut encore passer l'étape de **l'édition des liens** qui va grouper les différents fichiers « .o » de votre projet en un seul fichier exécutable.

```
g++ 01_HelloWorld.o -o 01_HelloWorld.exe
```


Compilation en ligne de commande

La compilation et l'édition des liens peuvent être groupées sur une seule et même ligne de commande.

Des options de compilations sont souvent ajoutées

```
g++ -std=c++20 -Wall -Wextra -Wconversion -Wsign-conversion -pedantic 01_HelloWorld.cpp -o 01_HelloWorld.out
```

- `-std=c++20` standard C++
- `-Wxx` les différents warning
- `-pedantic` warnings concernant C++ ISO
- `-o` le fichier de sortie (exécutable)



Il existe de nombreux IDE pour C++

- Code::Blocks
- CodeLite
- Dev-C++
- Eclipse
- NetBeans
- Visual Studio
- Xcode

Pour PRG1, ASD, ... nous utiliserons

- **CLion**



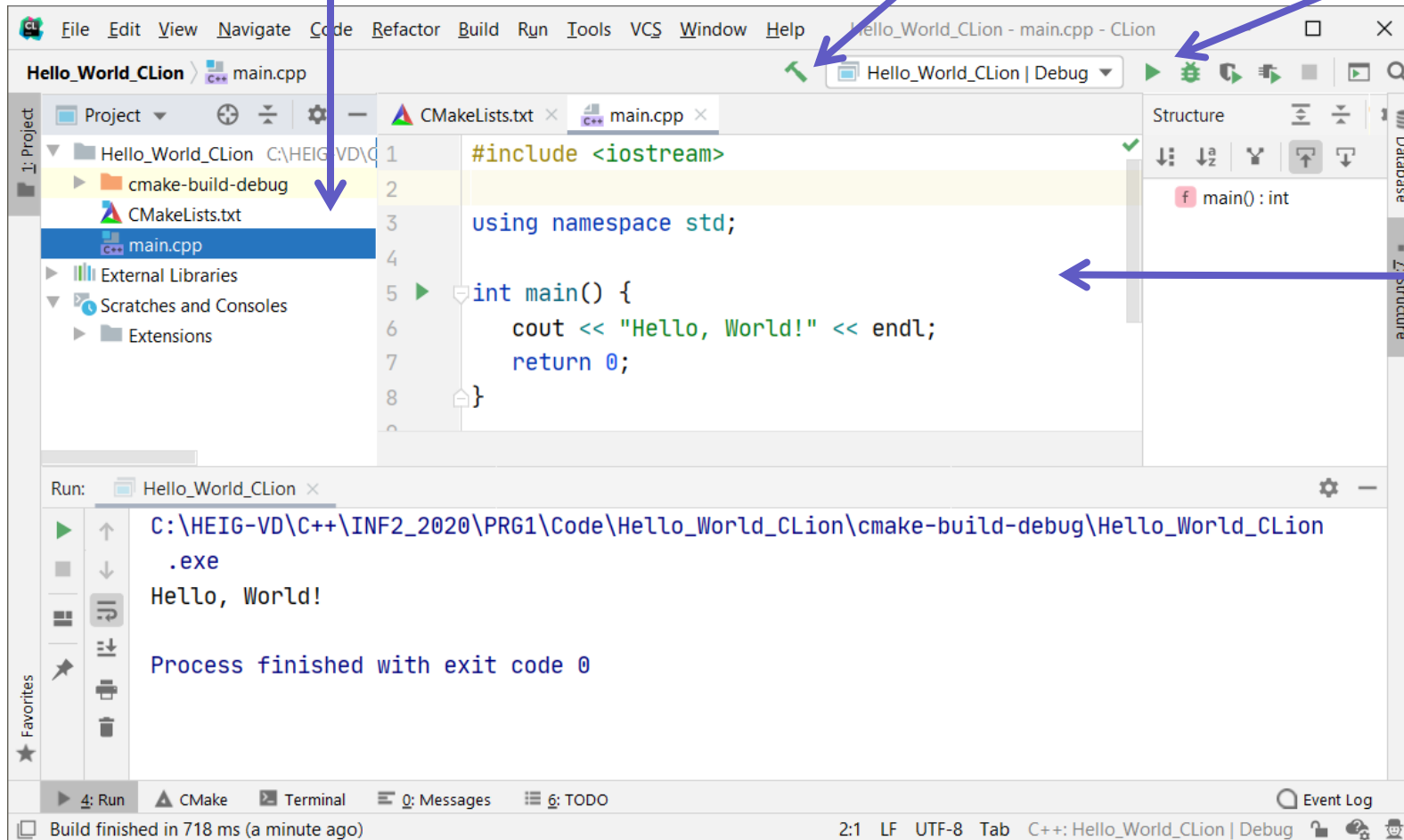


Les composantes de tout IDE

Un outil pour naviguer
dans vos projets / fichiers

Un outil pour construire
(compiler et lier) votre
programme

Un outil pour exécuter ou
debugger votre programme



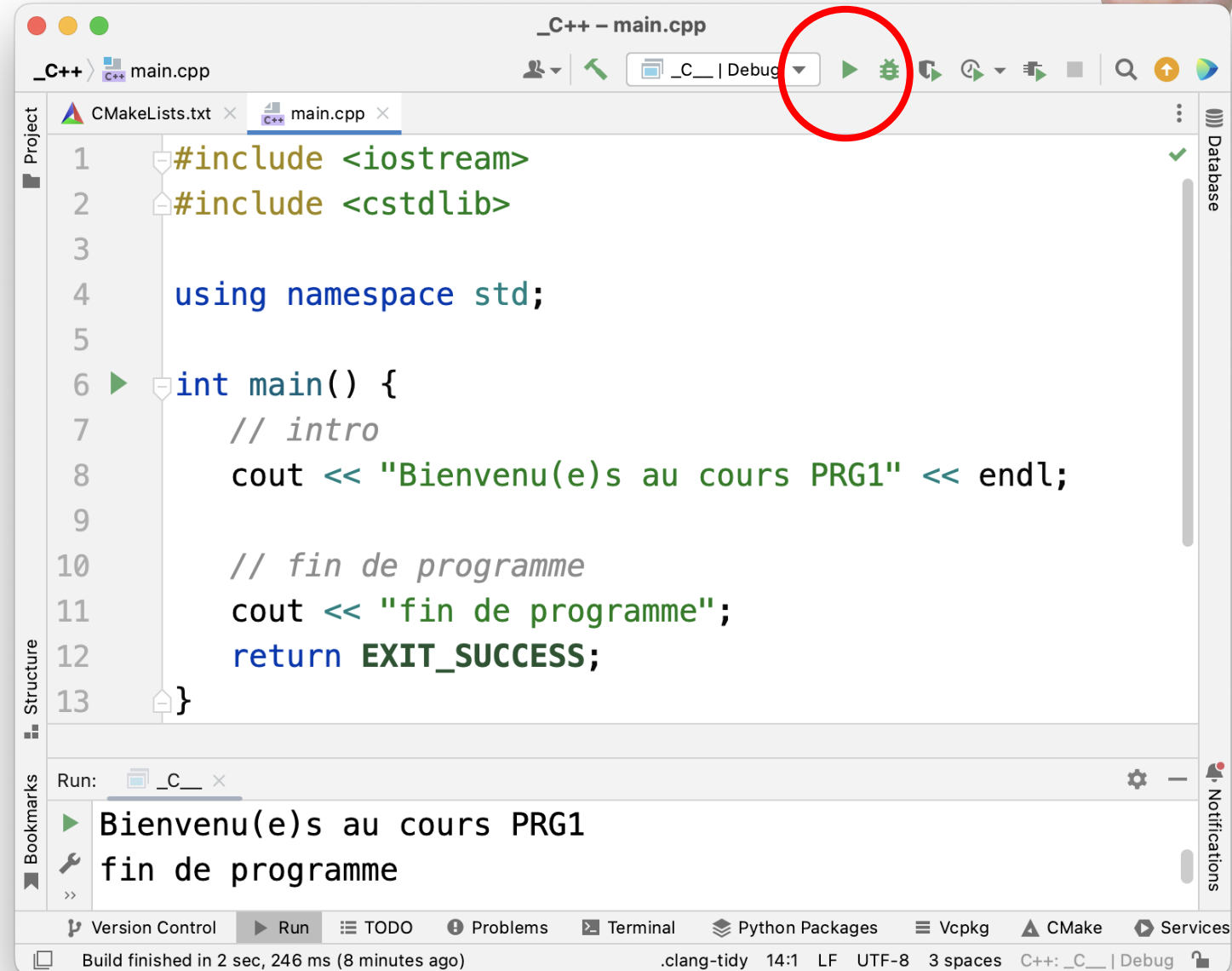
Un éditeur de texte où
écrire votre code source

HE^{VD} IG Compilation avec CLion

Introduction

Compiler avec un IDE est souvent très facile.

L'ensemble du processus (compilation et édition des liens) étant automatisé.





Clion utilise un fichier « **CMakeLists.txt** » qui permet de configurer le processus de compilation.

Lors de la création du projet, un fichier par défaut est mis à disposition

```
cmake_minimum_required(VERSION 3.25)
project(_C__)

set(CMAKE_CXX_STANDARD 23)

add_executable(_C__ main.cpp)
```

Pour plus de détails <https://cmake.org> et
<https://cmake.org/cmake/help/latest/manual/cmake-language.7.html>



ERROR

4. Erreurs de compilation, d'édition de liens et d'exécution

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!" << endl
    return 0;
}
```

Oups !

Erreur de compilation (*syntax error*)

- Sans le point-virgule, nous avons en fait écrit

```
cout << "Hello, World!" << endl return 0;
```

... ce que le compilateur ne comprend pas

- Une erreur de compilation survient dès lors qu'une **violation de la syntaxe du langage** est constatée
- Le compilateur ne sait plus, à partir de l'endroit où se trouve l'erreur, traduire votre code source en un code machine



Erreur de compilation (*syntax error*)

- Supposons que vous écriviez malencontreusement

```
cot << "Hello, World!" << endl;
```



Cela entraîne également une erreur de compilation

- Le **compilateur** se plaindra en vous avertissant qu'il ne sait pas ce que `cot` signifie. Le **message** exact dépend du compilateur, mais il ressemblera à :

```
error: use of undeclared identifier 'cot'; did you mean  
'cout'?
```

Erreur de compilation (*syntax error*)

- Le compilateur ne s'arrête pas de compiler après une erreur mais continue tant qu'il peut
 - Il est probable qu'il affiche de nombreuses erreurs, qui sont souvent des conséquences de la première erreur rencontrée
 - Dès lors, on ne corrigera que les erreurs qui nous parlent (qui correspondent à des messages d'erreurs compréhensibles), et surtout **la première erreur**
- ➔ Puis on relance la compilation, jusqu'à ne plus avoir d'erreur



Erreur d'édition des liens (*link error*)

- Chaque programme C++ doit avoir une et une seule fonction principale nommée `main`. La plupart des programmes C++ contiennent d'autres fonctions en plus de `main` (nous reviendrons sur les fonctions plus tard)
- Attention, C++ est **sensible à la casse** (MAJUSCULE \neq minuscule). Le programme suivant compile mais l'éditeur de lien échoue en ne trouvant pas `main`

```
int Main() {  
    return 0;  
}
```

Undefined symbols for architecture arm64:

 "_main", referenced from:

 implicit entry/start for main executable

ld: symbol(s) not found for architecture arm64

clang: error: linker command failed with exit code 1 (use -v to see invocation)

Erreur d'exécution – exceptions

- Certaines erreurs d'exécution sont si graves qu'elles génèrent une **exception**, un signal envoyé par le processeur qui - s'il n'est pas géré - **arrête le programme** et génère un message d'erreur
- Par exemple, si votre programme contient :

```
string s = "Hello";  
cout << s.at(10);
```

vosre programme peut se terminer en levant une exception `out_of_range`

```
libc++abi: terminating due to uncaught exception of  
type std::out_of_range: basic_string
```




Erreur d'exécution – segmentation fault

- Parfois, c'est l'OS qui décide de stopper l'exécution d'un programme jugé dangereux pour l'intégrité du système.
- Par exemple, un programme qui contient les lignes suivantes provoque une segmentation fault

```
int *p;  
p = nullptr;  
*p = 42;
```

```
Process finished with exit code 139  
(interrupted by signal 11: SIGSEGV)
```




Erreur d'exécution (*runtime error*)

- Considérons ce code

```
cout << "Hollo, World!" << endl;
```



- Une **erreur d'exécution** - ou **erreur logique** - est une erreur dans un programme qui compile (syntaxe correcte) mais qui n'exécute pas l'action attendue.

Pas vraiment une
erreur, alors ?

- Si, c'est une **erreur**, car le programmeur est responsable de l'**inspection** et du **test** de son programme pour éliminer les erreurs d'exécution



Le processus de développement

Pour chaque problème,
le programmeur passe par ces étapes

Comprendre le problème



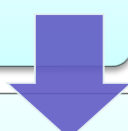
Développer et décrire un
algorithme



Tester l'algorithme avec
des entrées simples



Mettre en œuvre
l'algorithme en C++



Compiler et tester le
programme



Décrire un algorithme en pseudo-code

- Le **pseudo-code** est
 - une description informelle
 - pas un langage que l'ordinateur comprend
 - mais qu'on peut aisément traduire en un langage de haut niveau comme le C++
- La méthode décrite en pseudo-code doit
 - **être non ambiguë**, i.e. indiquer précisément
 - que faire à chaque étape
 - quelle est l'étape suivante
 - **être exécutable**, i.e. chaque étape peut être mise en œuvre
 - **se terminer**, i.e. son exécution amène à une étape finale

Décrire un algorithme en pseudo-code

- Considérons le problème suivant :
 - Vous hésitez entre acheter deux voitures.
 - L'une d'elles est plus chère à l'achat, mais consomme moins d'essence.
 - Vous connaissez le prix d'achat (en CHF) et la consommation (en litres aux 100 km) de chaque voiture.
 - Vous espérez utiliser votre voiture pendant 10 ans.
 - On suppose que l'essence coûte 1.45 CHF par litre et que l'on parcourt 10'000 km par an.
 - On achète la voiture cash et on ignore la complexité du financement.
- Quelle voiture est la moins chère globalement ?



Etape 1 – déterminer les entrées / sorties

- Dans notre exemple, les **entrées** sont
 - `prixAchat1`, le prix (en CHF) de la première voiture
 - `consommation1`, la consommation (en litres aux 100 km) de la première voiture
 - `prixAchat2`, le prix (en CHF) de la seconde voiture
 - `consommation2`, la consommation (en litres aux 100 km) de la seconde voiture
- Quant à la **sortie**, nous désirons simplement savoir
 - quelle est la voiture la plus économique



Etape 2 – décomposer en sous-problèmes

- Pour chaque voiture (N vaudra 1 ou 2) :
 1. Le coût total de la voiture sera
 $\text{prixAchatN (entrée)} + \text{coutUtilisationN}$
 2. En supposant une utilisation constante et un prix de l'essence stable, le prix d'utilisation sera
 $\text{nombreAnnees (10)} \times \text{coutAnnuelEssenceN}$
 3. Le coût annuel de l'essence sera
 $\text{prixParLitre (1.45)} \times \text{quantiteAnnuelleEssenceN}$
 4. Enfin, la quantité annuelle d'essence consommée sera
 $\text{nombreDeKmRoules (10000)} \times \text{consommationN (entrée)} / 100$



Etape 3 – décrire les sous-probl. en pseudo-code

- On doit organiser les étapes pour que chaque valeur intermédiaire nécessaire à une étape soit calculée avant d'être utilisée

```
pour chaque voiture N (N = 1 ou 2)
    calculer quantiteAnnuelleEssenceN = nombreDeKmRoules (10000) x consommationN / 100
    calculer coutAnnuelEssenceN = prixParLitre (1.45) x quantiteAnnuelleEssenceN
    calculer coutUtilisationN = nombreAnnees (10) x coutAnnuelEssenceN
    calculer coutTotalN = prixAchatN + coutUtilisationN

si coutTotal1 < coutTotal2
    choisir la voiture 1
sinon
    choisir la voiture 2
```



Etape 4 – tester le pseudo-code

- Testons le pseudo-code avec les valeurs suivantes :
 - Voiture 1 : 25'000 CHF, 5.6 litres aux 100 km
 - Voiture 2 : 22'000 CHF, 7.8 litres aux 100 km
- Pour la voiture 1 :
 - $\text{quantiteAnnuelleEssence1} = 10000 * 5.6 / 100 = 560$
 - $\text{coutAnnuelEssence1} = 1.45 * 560 = 812$
 - $\text{coutUtilisation1} = 10 * 812 = 8120$
 - $\text{coutTotal1} = 25000 + 8120 = 33120$
- Pour la voiture 2, on trouve : $\text{coutTotal2} = 33310$
- Conclusion : la voiture 1 est la plus économique