POLYTECH
DIJON

2025/2026

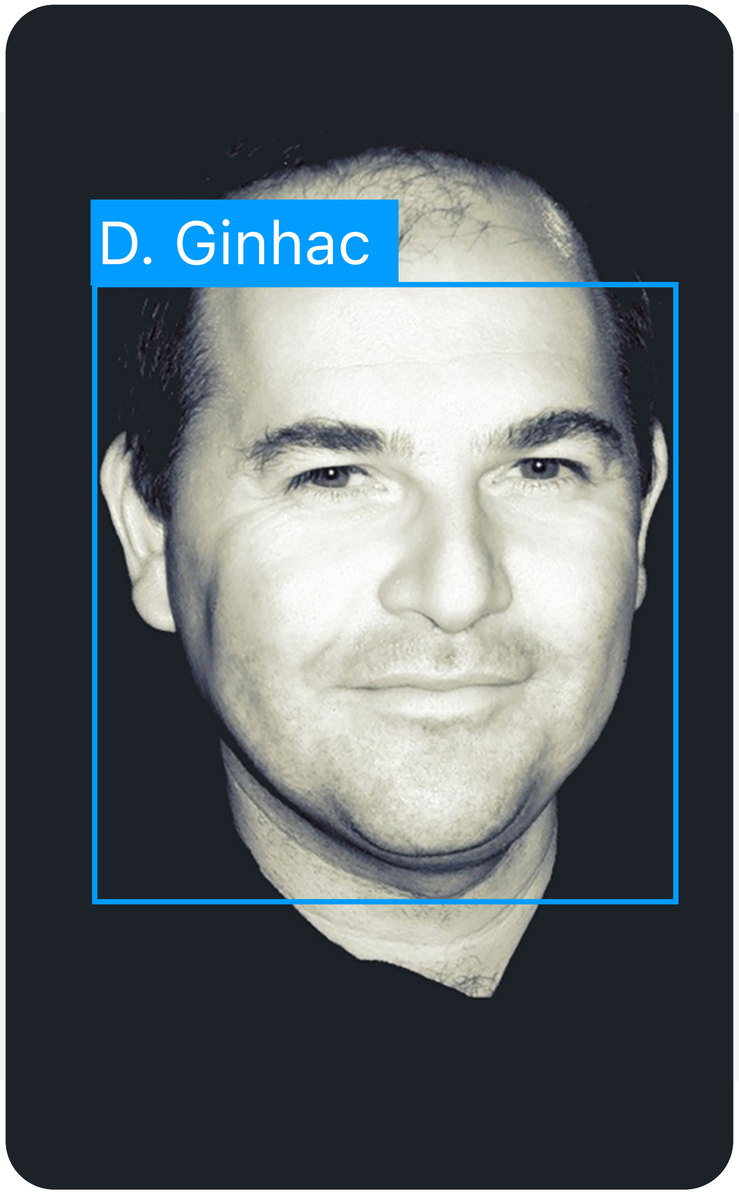
ITC313

Mastering C++:

🎨 The Art of Object-Oriented Programming

Pr. Dominique Ginhac
dginhac@ube.fr

Last updated on September, 30, 2025 by dG



$ ~ whoami

HI, I'M D. GINHAC

👩‍🏫 I'm your instructor for Object-Oriented Programming.

🤖 Researcher in AI architectures for real-time Computer Vision.

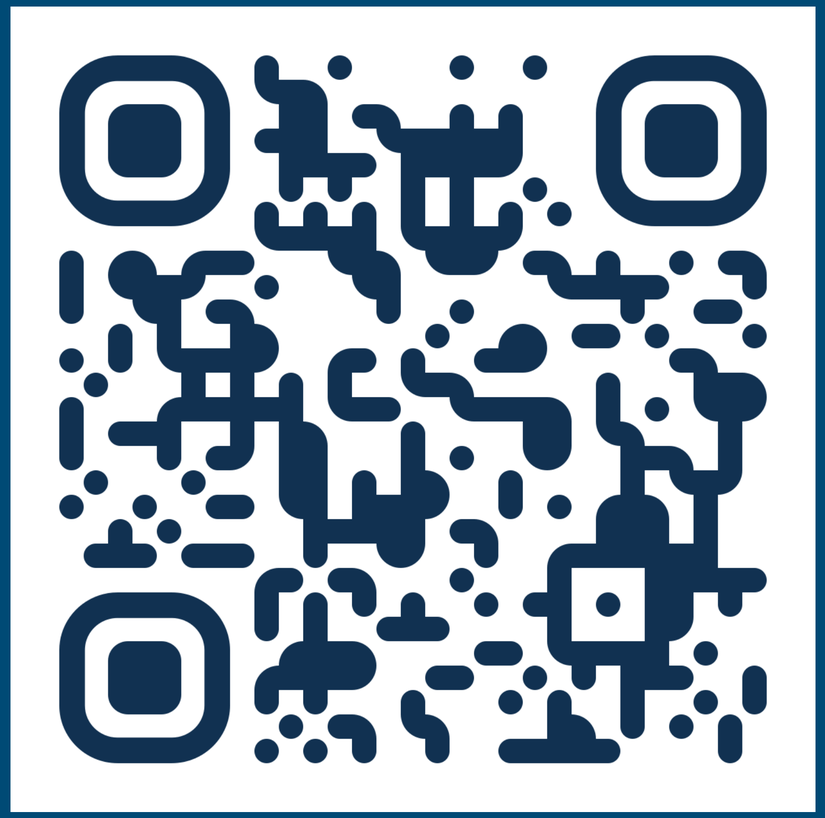
🧹 Passionate about writing elegant code — simple, maintainable, and refined through refactoring.

🇫🇷 When not coding, also a national IT expert for the French Ministry of Higher Education and Research.

dginhac@ube.fr

dginhac

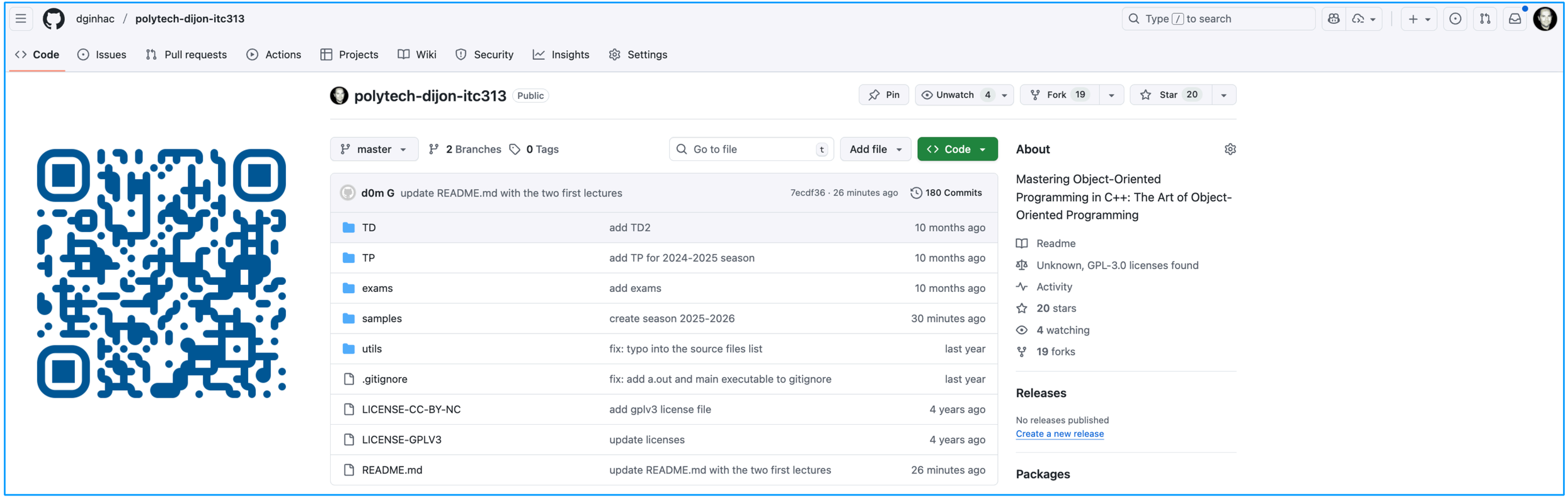
dginhac



🌐 https://ginhac.com/ITC313/00-intro.pdf

Everything Else Available on

https://github.com/dginhac/polytech-dijon-itc313



The ITC313 Course Scheduling

Photo by Xin Wang on Unsplash

Lectures (CM)

Starting Week 39

13 x 1.75 h
D. Ginhac

Tutorials (TD)

Starting Week 42

6 x 1.75 h
D. Ginhac

Labs (TP)

Starting Week 4?

12 x 2 h
Other instructors

Exams

To be defined

Grading = Midterm + Final + Labs

Today

Lecture #00
Course Introduction

Lecture #01
User-defined Data Types
Abstraction/Encapsulation

Lecture #02
Inheritance

Lecture #03
Polymorphism

Lecture #04
STL Containers

Lecture #05
Templates

...

Lecture #00

https://ginhac.com/ITC313/00-intro.pdf

All code samples are available on *in directory "samples/00-intro"*

Course Introduction

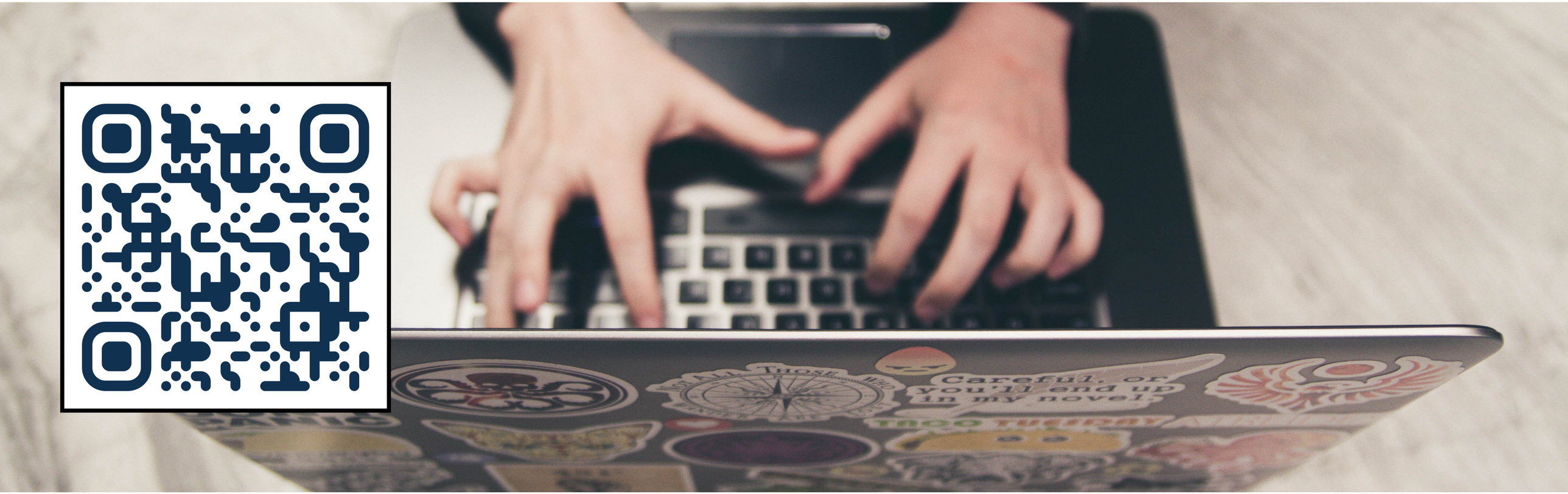


Photo by NeONBRAND on Unsplash

ITC313 Kick-off: Starting Your Journey in C++ and OOP.

A Quick Survey To Get To Know You!

https://app.wooclap.com/ITC313

Find Your Learning Objectives

Beginner
(aka Padawan)

✨ *“Start small, think big!”*

No C++ experience? No problem!

This course will guide you step by step into object-oriented programming, starting from the very basics.

Intermediate
(aka Jedi Knight)

✨ *“Level up your C++ skills!”*

Already coded a bit in C++?

Here, you’ll sharpen your skills, learn best practices, and discover the most common design patterns used by professionals.

Expert
(aka Jedi Master)

✨ *“Master the art of C++!”*

Comfortable with coding?

This course will challenge you with complex problems and show you advanced techniques to optimize your programs to the max.

🚀 “ By the end of this course …

… You’ll walk away with OOP skills you didn’t have before!”

How To Achieve These Learning Objectives?

👂 Listen carefully.

Lecture slides, code examples, tutorials, lab works → all on GitHub.

Attend and participate actively in lectures, tutorials, and labs.

🚀 Be proactive.

Slides are sometimes intentionally sparse → take notes!

Use your smartphone 📱 for blackboard snapshots.

Use your laptop 💻 to download, modify, and test code examples.

❓ Ask when unclear.

Learning C++ is not always easy.

Be active: ask questions whenever something is unclear.

I'll explain again — no problem, that's how you learn.

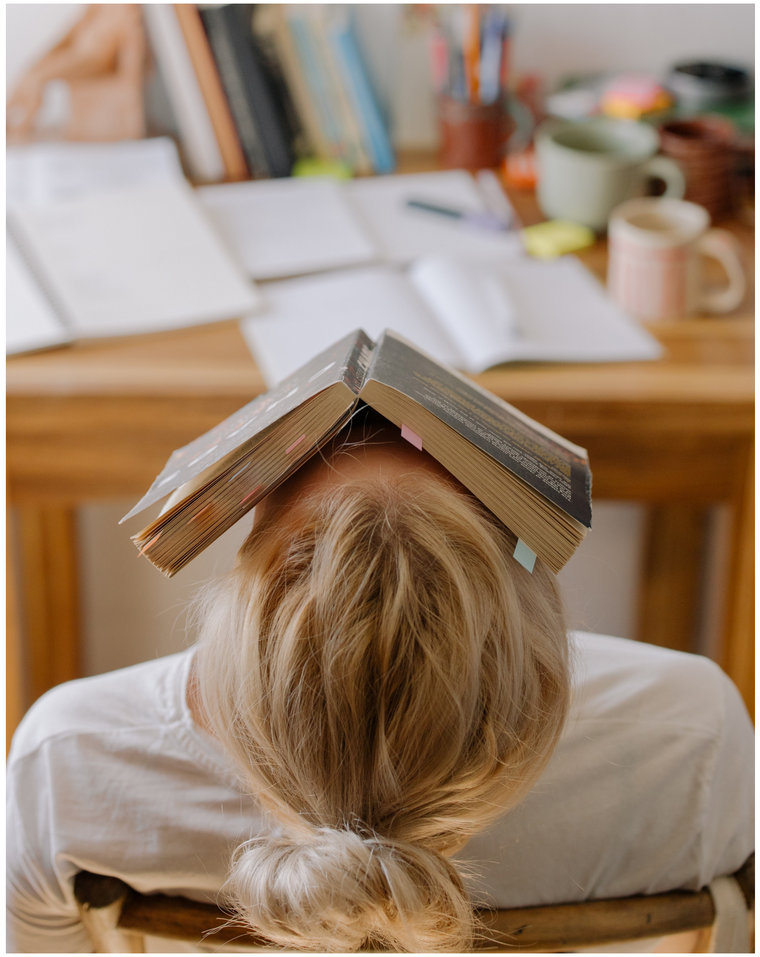


Photo by cottonbro from Pexels

✨ "Your success depends on listening, practicing, and asking questions."

AGENDA

00 - Course Introduction

1. Why learning C++?
2. The art of programming
3. hello, world

AGENDA

00 - Course Introduction

1.Why learning C++?

2. The art of programming

3. hello, world

The C Language Was Born in 1972...

Closely tied to **the dev of** UNIX OS.

The C programming language has been created to move the UNIX kernel code **from assembly to a higher level language** while guaranteeing **high performance**.

Photo by National Inventors Hall of Fame

Dennis RITCHIE
/ Inventor of C

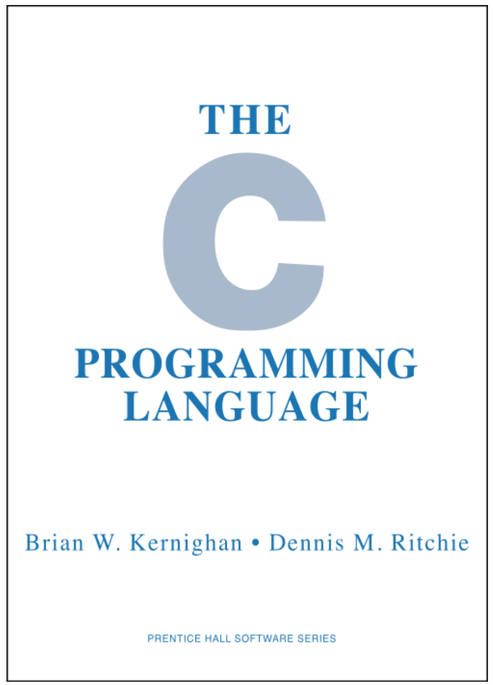


Photo by National Inventors Hall of Fame

Ken THOMSON
/ Designer of UNIX OS

... And C++ in 1983.

Originally defined as an extension of the C language called “**C++ = C with Classes**”.

Photo by Bjarne Stroustrup

Bjarne STROUSTRUP
/ Inventor of C++

📏 Standardization

Updated standard every 3 years.
C++23 is the current release (23/12), C++26 is in dev.

🔥 C++11 was a revolution

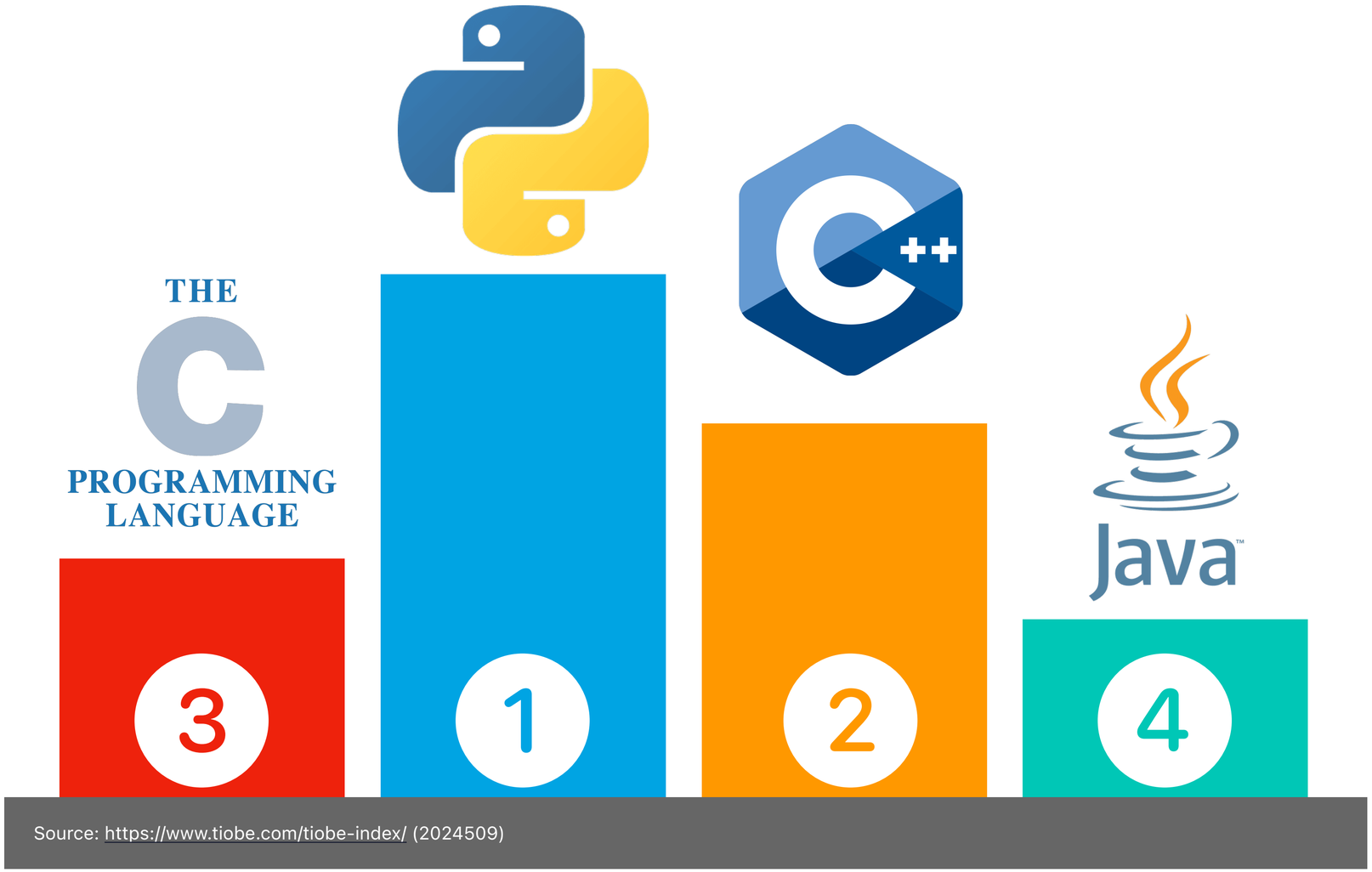
C++ before 2011 was C with classes.
C++ since 2011 is a new OOP language.

💡 Modern C++

Modern C++ stands for C++ that is based on C++11 and beyond.
Most C++ compilers support C++11 to C++20 (23?) features.

C/C++ Are Dinosaurs From the 20th Century — Who Would Still Use Them Today?

Photo by Computer History - DEC - PDP-11- Ken Thompson and Dennis Ritchie



✨ “C, C++, Python, Java, all born in the 70s, 80s, and 90s — still powering today’s software.”







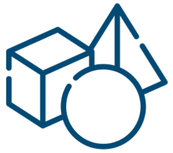

Welcome to the Jungle!

So many languages — why does C++ still lead the way?





💪 The Power of OOP



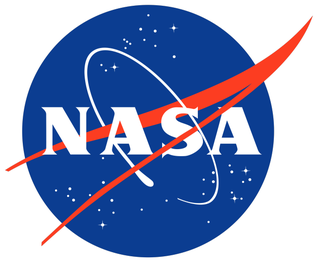
C++ supports Object-Oriented Programming — the paradigm behind everything from graphical user interfaces to modern game engines.

💬 One Language, Many Worlds

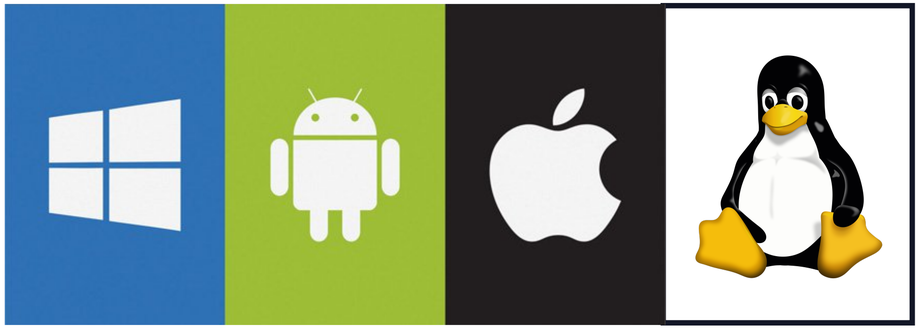
From operating systems to video games to scientific simulations, C++ is everywhere — even on Mars rovers!

🏎️ Speed That Matters

C++ is built for performance — from the smartphones in your pocket to the world's fastest supercomputers.



C++ Powers the Fortune 500 Companies



C++ Runs Behind the Software You Use Every Day

TensorFlow

Companies Worldwide Are Looking for C++ Talent — Will You Be One of Them?

On LinkedIn France alone, more than 20,000 job offers mention C++ (2025/09)

WE ARE HIRING!

YES

C++ Is Not Outdated. It Still Matters in 2025 — and Mastering It Makes You a Better Developer.

AGENDA

00 - Course Introduction

1. Why learning C++?
2. The art of programming
3. hello, world

AGENDA

00 - Course Introduction

1. Why learning C++?
2. The art of programming
3. hello, world

Programming is the art of telling another human what one wants the computer to do.

Donald KNUTH

/ The art of computer programming

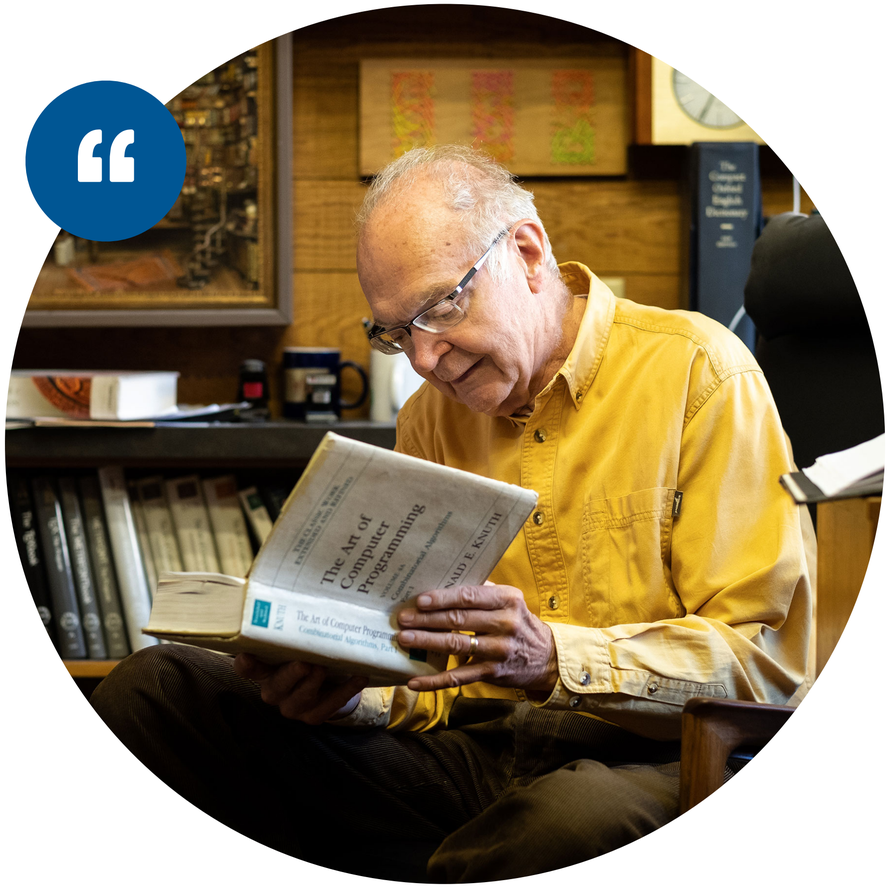


Photo by Vivian Cromwell for Quanta Magazine

Art of Correctness

First goal: **make it work**.

📝 **It is just a matter of syntax to write a correct sequence of instructions.**

“Does the code you wrote (or someone else) compile and run correctly?”

⚠️ Reminder: This is only the first step, not the final one.

01

Art of Design

Next goal: **make it efficient**.

⚙️ **Even bad code can work!**

“How fast? How much memory? How scalable?”

02

👉 Performance matters — but efficiency without clarity is fragile.

03

Art of Style

Final goal: **make it beautiful**.

🎨 **Elegant or Clean code is easy to understand and easy to change.**

"How understandable is your source code?

Can someone else (or you, in 6 months) read and reuse it?"

🕰️ Without clarity and style, today's working code becomes tomorrow's legacy."

Make It Work, Make It Fast, Make It Clean.

Iterative process

Correctness

🚀 Make your code run correctly.

Focus on small steps → split the problem and solve one piece at a time.
Write minimal classes → include only what's necessary.
Use language reference → rely on the C++ standard library and official docs (🌐 https://en.cppreference.com)
Keep cheatsheets handy → quick reminders save time (🌐 https://github.com/mortennobel/cpp-cheatsheet)

Design

⚡Think code quality.

Test everything → each function, each module.
Isolate components → structure your code to prevent regressions.
Don't repeat yourself → avoid duplication, reuse code.
Refactor often → improve efficiency step by step.

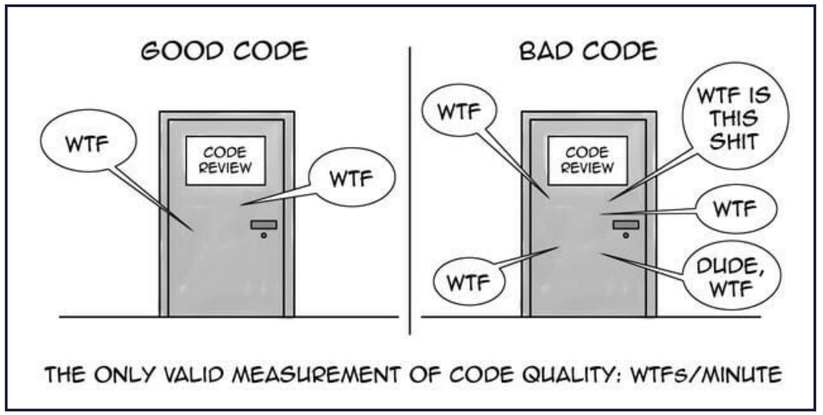


Photo by Elyess Eleuch on dev.to

Style

🎨 Increase readability.

Name things well → variables, classes, functions should express intent.
Follow naming conventions → consistent style improves collaboration (🌐 http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines.html)
Comments ≠ code → code shows what is done, comments explain why.
Minimal comments → compilers ignore them… and many programmers too.

AGENDA

00 - Course Introduction

1. Why learning C++?
2. The art of programming
3. hello, world

AGENDA

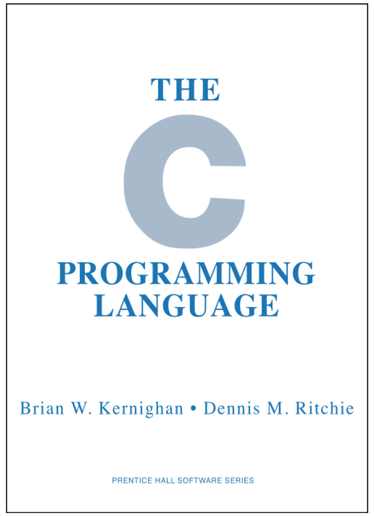
00 - Course Introduction

1. Why learning C++?
2. The art of programming

3.hello, world

hello, world

... takes more than just code



In C, the program to print "hello, world" is

```
main( )
{
        printf("hello, world\n");
}
```

Just how to run this program depends on the system you are using. As an specific example, on Unix you must create the source program on a file whose name ends in ".c", such as *hello.c*, then compile it with the *cc* command

cc hello.c

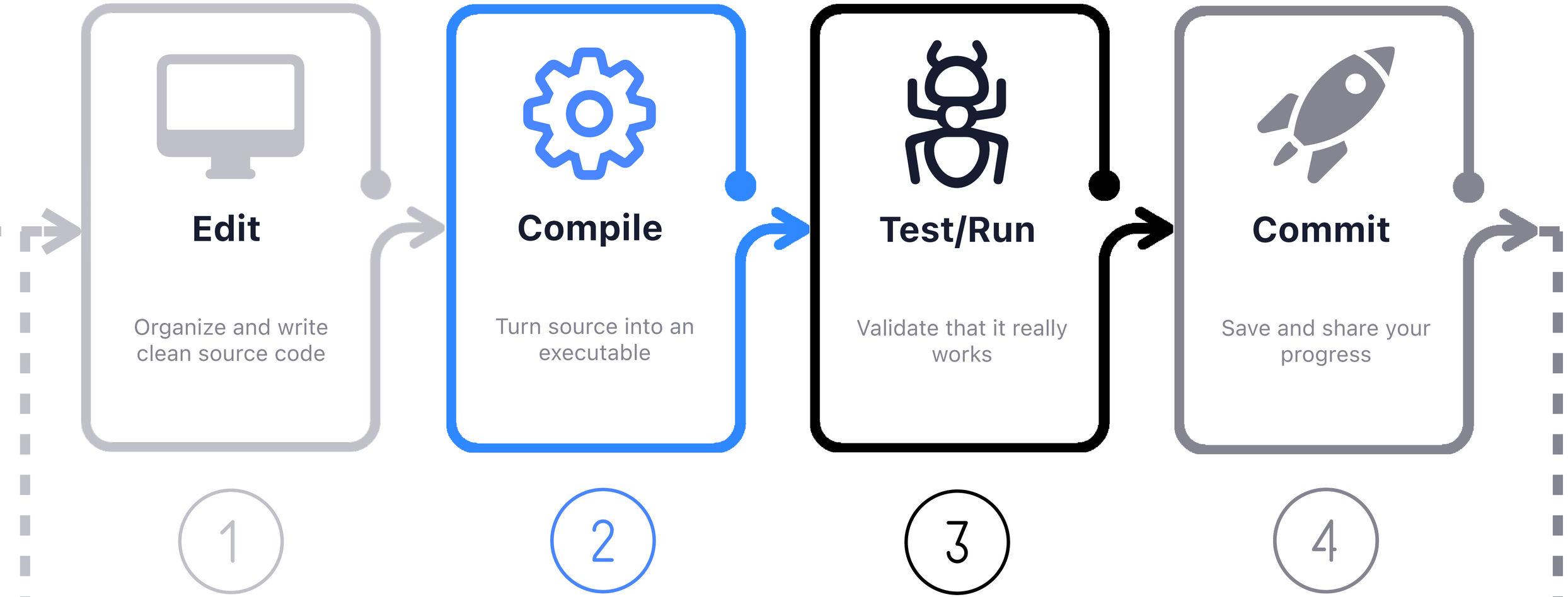
If you haven't botched anything, such as omitting a character or misspelling something, the compilation will proceed silently, and make an executable file called *a.out*. Running that by the command

a.out

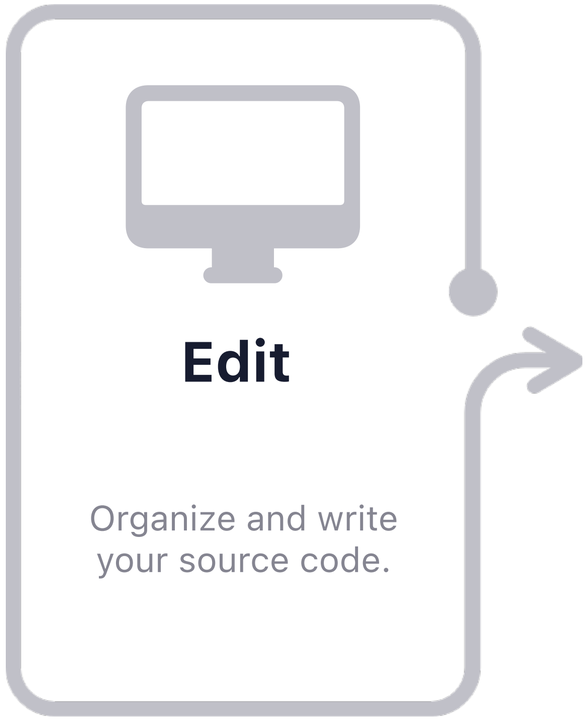
will produce

hello, world

Hello, World Needs More Than Code — It Needs a Workflow.



🧠 Think First — Then Turn Ideas Into Code ✍️.



Edit

Organize and write your source code.

1

🌐 *www.vim.org*

🌐 *www.sublimetext.com*

🌐 *www.codeblocks.org*

🌐 *zed.dev*

🌐 *code.visualstudio.com*

🌐 *www.jetbrains.com*

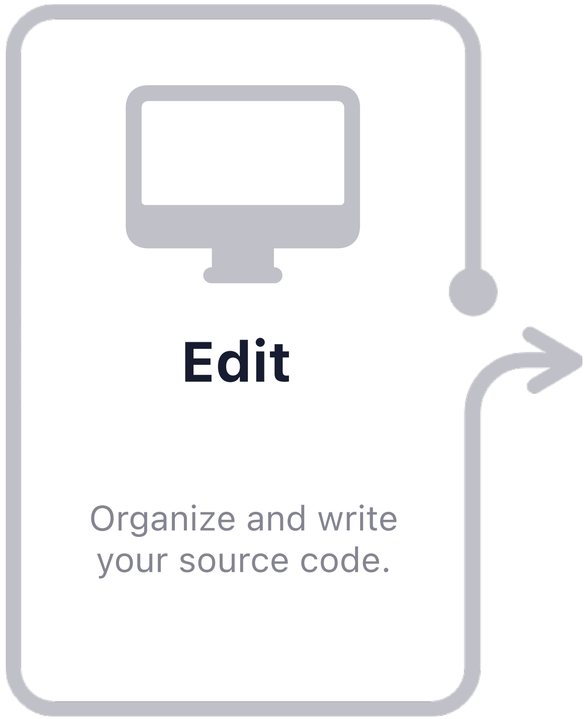
🌐 *cursor.com*

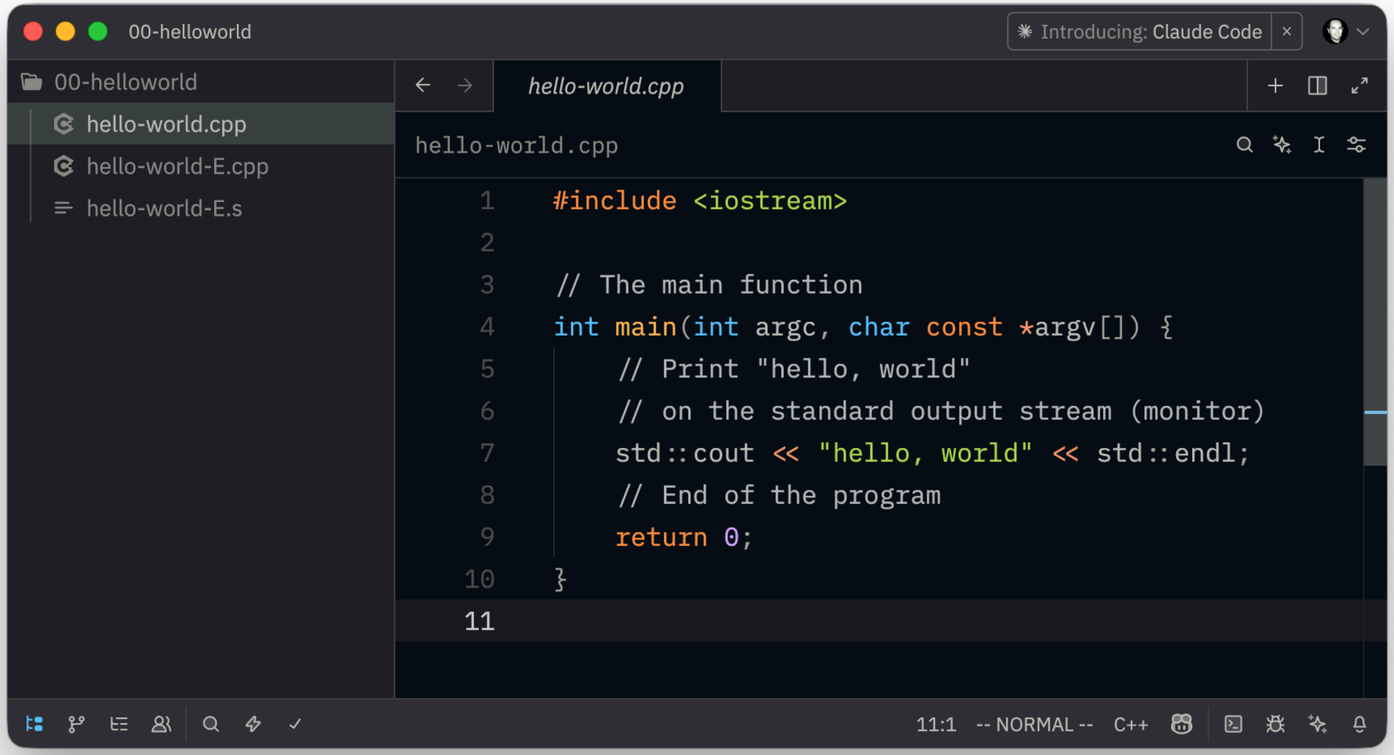
🌐 *replit.com*

🛠️ "From plain editors to AI-powered IDEs, there’s a tool for every coder."

Writing Your First C++ Code: “hello, world”

00-helloworld/hello-world.cpp





Written with

Without Compilation, C++ Is Just Text and Not a Working Program.

Compile

Turn source into an executable

2

GNU Compiler Collection

gcc.gnu.org

Minimalist GNU for Windows

www.mingw.org

LLVM Compiler Infrastructure

clang.llvm.org

From C++ Source to Executable — Behind the Compilation Process

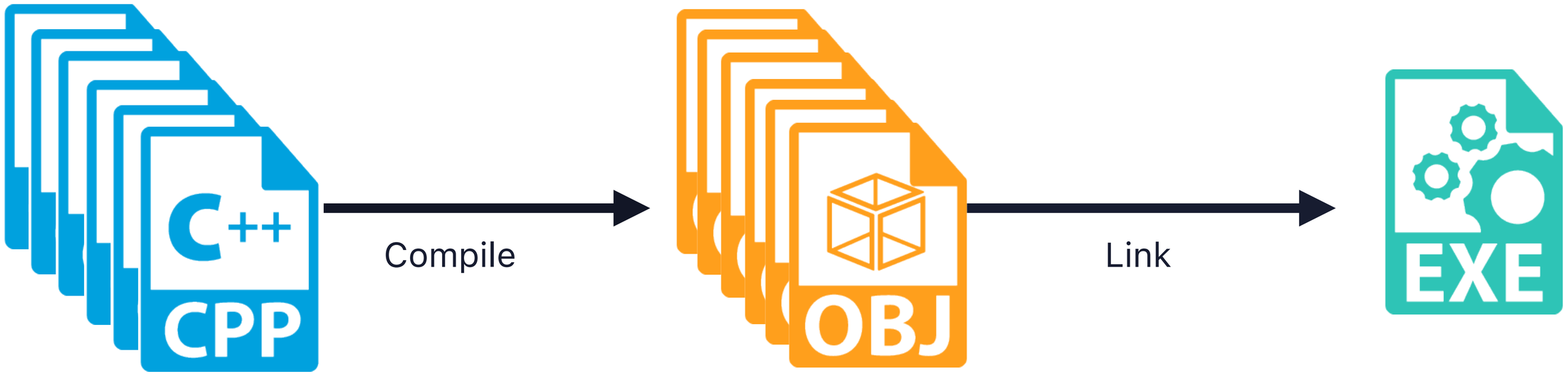
⚙️ “Compilation is not a single action, but a series of steps that translate human-readable C++ code into machine instructions.”

From C++ Source to Executable — Behind the Compilation Process

COMPILATION = **4 steps**

- 📄 **Editing** → .cpp file - 10 lines (246 B)
- 📄 **Preprocessing** (-E) → Expands #include, macros, and directives → cpp file - 66023 lines (3.1 MB)
- 📝 **Compilation** (-S) → Turns preprocessed C++ into assembly code → .s file - 1405 lines (52 KB)
- ⚙️ **Assembly** (-c) → Converts assembly into object code → .o file - 10 KB
- 🔗 **Linking** → Combines all objects and libraries into the final executable → executable file - 38 KB

Splitting Code Into Multiple Files, Building Better Programs



AUTOMATE
THE COMPILATION
with MAKEFILES

Make is a **build automation tool** that compiles programs and libraries from source code using instructions written in Makefiles.

In a **Makefile**, you describe what you want to build and how to build it.

Then, with a single **make** command in the **terminal**, the compilation runs automatically.

Upcoming demo

Proving Your Code Works

🧪 "Only deep testing proves your code really behaves as expected."

The testing process involves detecting bugs, assessing their impact, identifying their causes, and fixing them.

Using a **debugger** to **monitor** your program.

An interactive debugger analyzes how a program flows and helps identifying incorrect running code that can cause unexpected behavior or crash.

Key concepts are breakpoints, watchpoints, stepping, viewing data, ...

The most used C++ debuggers are 🌐 gdb from GNU and 🌐 lldb from LLVM.

Writing and running **unit tests** to **prevent bugs**.

A unit test is an automated test that verifies a small piece of code in an isolated manner. It helps finding problems early in the development cycle and makes the final debug step easier.

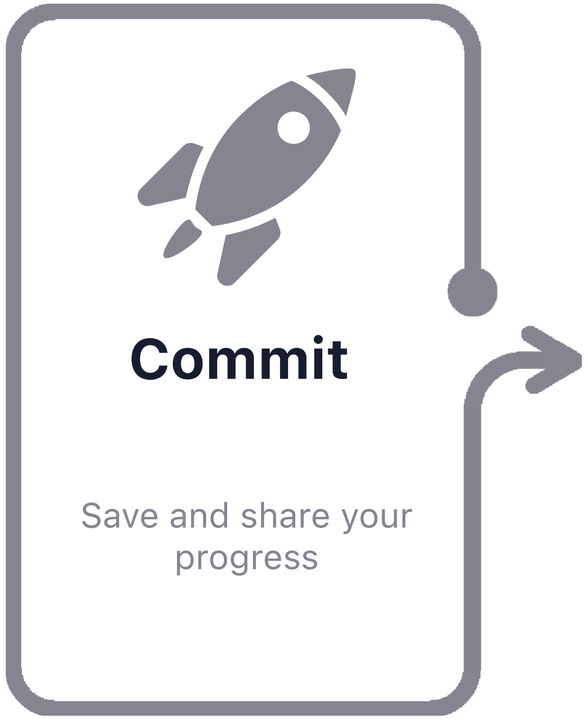
Key concepts are assertions (boolean expressions).

The most used C++ testing frameworks are 🌐 Google Test, 🌐 Boost, 🌐 Catch2.

Upcoming demo

The Power of Versioning

📂 "Commits are the backbone of safe and shareable code."

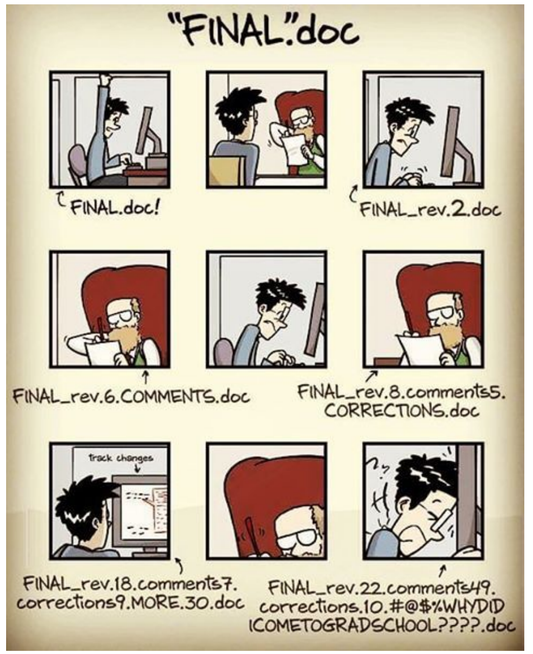


Commit

Save and share your progress

4

Software versioning is the practice of archiving successive versions of a codebase. It allows developers to track changes, understand how the project evolves, and collaborate more effectively.



What is git?

🌐 Git is the most commonly used version control system.

Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to.

See 🌐 githowto.com for a guided tour that walks through the fundamentals of Git.

Why using git is so crucial?

Git is the key to a successful and modern development workflow.

Git helps you to maintain the backup of your source code.

Git helps you to collaboratively work with other developers.

Upcoming demo

AI Tools for Coding

Claude Code

Gemini Code Assist

Codestral

ChatGPT Codex

AskCodi

Lovable

GitHub Copilot

Tabnine

Cursor

V0

Windsurf

Qodo

✨ “From Copilot to Cursor, AI is everywhere in programming.”

Will AI Replace Developers?

❓ “A question everyone is asking… but is it the right one?”

- **AI can automate routine tasks** — code generation, bug fixing, documentation.
- **AI learns from massive datasets** — generating code similar to what already exists.
- **But AI lacks creativity and critical thinking** — humans are essential for designing complex software.

👨‍💻 “Humans innovate, 🤖 AI assists, — making AI your perfect Pair Programming Partner.”

C++ Intro — Key Takeaways

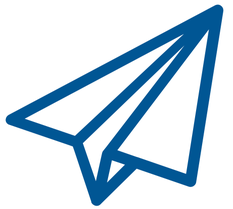
🌍 **Relevance** → C++ still powers the world.

🎨 **Art** → Code must be correct, efficient, clean.

⚙️ **Workflow** → Edit → Compile → Test → Commit.

🤖 **AI Tools** → Helpful assistants, but true quality comes from mastering OOP principles.

A FIRST TAKE HOME MESSAGE



#1

“Programming is a skill best acquired by practice and example rather than from books.” – Alan Turing

✨ “Coding comes from problem-solving, design, and practice — not syntax.”

Questions

Contacts

Pr. Dominique Ginhac

dginhac@ube.fr

Come visit us at

https://github.com/dginhac/polytech-dijon-itc313