Modern C++ Programming

2. Preparation

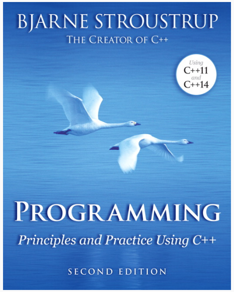
Federico Busato

2026-01-06

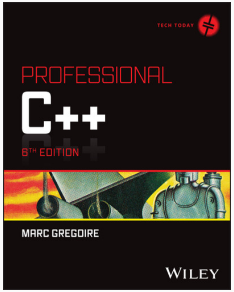
Table of Contents

Books and References

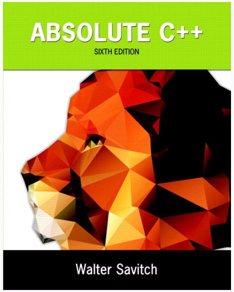
Suggested Books



Programming and Principles using C++ (3nd, C++23)
B. Stroustrup, 2024

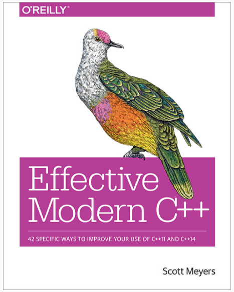


Professional C++ (6th, C++23)
M. Gregoire, 2024

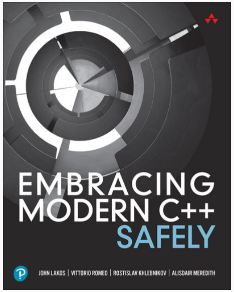


Absolute C++ (6th)
W. Savitch, 2015

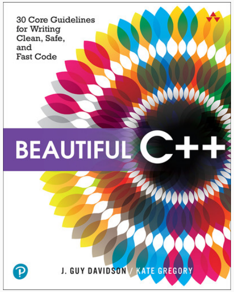
More Advanced Books



Effective Modern C++
S. Meyer, 2014


EMBRACING MODERN C++ SAFELY
JOHN LAKOS | VITTORIO ROMEO | ROSTISLAV KHLEBNIKOV | ALISDAIR MEREDITH

Embracing Modern C++ Safely
J. Lakos, V. Romeo, R. Khlebnikov, A. Meredith, 2021


30 Core Guidelines for Writing Clean, Safe, and Fast Code
BEAUTIFUL C++
J. GUY DAVIDSON / KATE GREGORY

Beautiful C++: 30 Core Guidelines for Writing Clean, Safe, and Fast Code
J. G. Davidson, K. Gregory, 2021

References 1/3

(Un)official C++ reference:

- en.cppreference.com
- C++ Standard Draft

Tutorials:

- Learn C++
- Tutorials Point C++
- en.wikibooks.org/wiki/C++
- yet another insignificant...programming notes

Other resources:

- stackoverflow.com/questions/tagged/c++

References

News:

- `isocpp.org` ↗: Standard C++ Foundation
- `Reddit C++` ↗
- `LibHunt` ↗ and `Awesome C++ Weekly` ↗
- `MeetingCpp Blogroll` ↗
- `Accu Overload Journal` ↗

Coding exercises:

- `HackerRank C++` ↗
- `leetcode.com/problemset/algorithms` ↗
- `open.kattis.com` ↗

References

Main conferences:

- `CppCon`: `slides`, `search engine`
- `CppNow`: `slides`
- `MeetingCpp`: `slides`
- `CppNorth`: `slides`
- `Accu`: `slides`
- `isocpp.com conference list`

Slide Legend

Slide Legend

- ★ **Advanced Concepts**. *In general, they are not fundamental*. They can be related to very specific aspects of the language or provide a deeper exploration of C++ features.
  A beginner reader should skip these sections/slides.

- ⇝ **See next**. C++ concepts are closely linked, and it is almost impossible to find a way to explain them without referring to future topics. These slides should be revisited after reading the suggested topic.

- 🏠 **Homework**. The slide contains questions/exercises for the reader.

Slide Legend

```
this is a code section
```

This is a language `keyword/token` and not a program symbol (variable, functions, etc.). Future references to the token could use a standard code section for better readability.

Parenthesis and Brackets

- `{}` **braces**, informally “curly brackets”
- `[]` **brackets**, informally “square brackets”
- `()` **parenthesis**, informally “round brackets”
- `<>` **angle brackets**

What Editor/ IDE/Compiler Should I Use?

What Compiler Should I Use?

Most popular compilers:

- Microsoft Visual Code (**MSVC**) is the compiler offered by Microsoft
- The GNU Compiler Collection (**GCC**) contains the most popular C++ Linux compiler
- **Clang** is a C++ compiler based on LLVM Infrastructure available for Linux/Windows/Apple (default) platforms

Suggested compiler on Linux for beginner: **Clang**

- Comparable performance with GCC/MSVC and low memory usage
- Expressive diagnostics (examples and propose corrections)
- Strict C++ compliance. GCC/MSVC compatibility (inverse direction is not ensured)
- Includes very useful tools: memory sanitizer, static code analyzer, automatic formatting, linter, etc.

Install the Compiler on Linux

Install the last `gcc/g++ (v14)`

```
$ sudo add-apt-repository ppa:ubuntu-toolchain-r/test
$ sudo apt update
$ sudo apt install gcc-14 g++-14
$ gcc-14 --version
```

Install the last `clang/clang++ (v21)`

```
$ wget https://apt.llvm.org/llvm.sh
$ chmod +x llvm.sh
$ sudo ./llvm.sh 21
$ clang++ --version
```

Install the Compiler on Windows

Microsoft Visual Studio

- Direct Installer: `Visual Studio Community 2026`

Clang on Windows

- Windows Subsystem for Linux (WSL)
  - `Windows Powershell` → `wsl -install -d Ubuntu-24.04`
  - `Enable-WindowsOptionalFeature -Online -FeatureName VirtualMachinePlatform -All -NoRestart`
  - Enable virtualization support in UEFI/BIOS: `Intel Virtualization Technology (VT-x)` (Intel) or `SVM Mode` (AMD)
- Clang + MSVC Build Tools
  - Download Build Tools per Visual Studio
  - Install `Desktop development with C++`

What Editor/IDE/Compiler Should I Use? 1/3

Popular C++ IDE (Integrated Development Environment):

- **`Microsoft Visual Studio`** Most popular IDE and compiler (MSVC) for Windows .
- **`Clion`** (free for non-commercial use). Powerful IDE with a lot of options.
- **`QT-Creator`**. Fast C++ IDE.
- **`XCode`**. Default IDE on Mac OS
- **`Cevelop`** C++ IDE based on Eclipse.

What Editor/IDE/Compiler Should I Use? 1/3

Standalone GUI-based coding editors:

- **`Microsoft Visual Studio Code`** (VSCode)
- **`Cursor`** is an AI-powered code editor designed to enhance the software development process. Based on Microsoft Visual Studio Code.
- **`Void Editor`** is an open-source Cursor alternative.
- **`Windsurf`** Windsurf Editor is an AI-powered editor designed to deeply integrate artificial intelligence into the coding workflow.
- **`Sublime`** is a sophisticated, high-performance text editor.
- **`Lapce`** is a modern, open-source code editor written in Rust, designed for speed.
- **`Zed`** is a high-performance code editor built from scratch in Rust, focusing on speed, collaboration, and AI integration

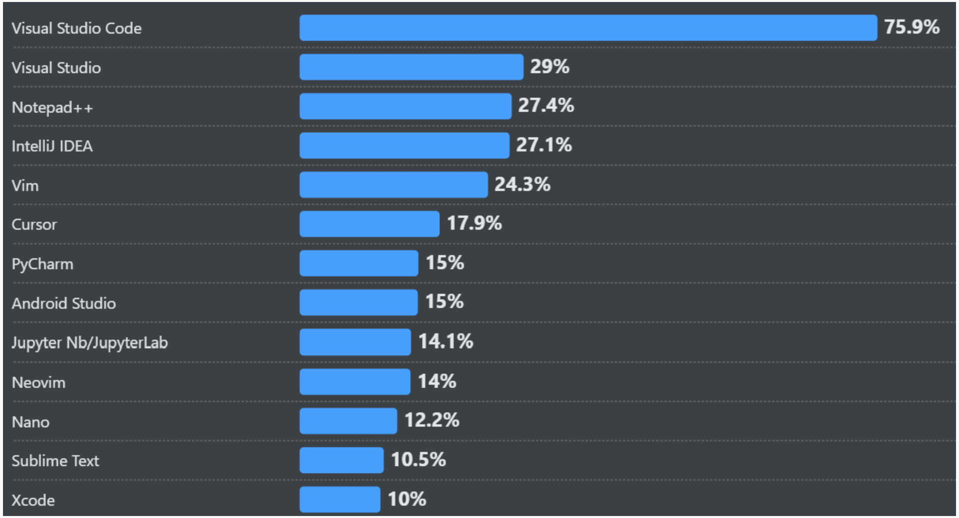
What Editor/IDE/Compiler Should I Use? 2/3

Standalone text-based coding editors (powerful, but needs expertise):

- **`Vim`**
- **`Emacs`**
- **`NeoVim`**
- **`Helix`**

Not suggested: `Notepad, Gedit, and other similar editors` (lack of support for programming)

What Editor/IDE/Compiler Should I Use? 3/3



StackOverflow Developer Survey 2025

How to compile?

How to Compile?

Compile C++11, C++14, C++17, C++20, C++23, C++26 programs:

```
g++  -std=c++11 <program.cpp> -o program
g++  -std=c++14 <program.cpp> -o program
g++  -std=c++<version> <program.cpp> -o program
```

Any C++ standard is backward compatible*

C++ is also backward compatible with C in most case, except if it contains C++ keywords (`new`, `template`, `class`, `typename`, `etc.`)
We can potentially compile a pure C program in C++26

*except for very minor deprecated features

C++ Standard

Compiler	C++11		C++14		C++17		C++20	
	Core	Library	Core	Library	Core	Library	Core	Library
`g++`	`4.8.1`	`5.1`	`5.1`	`5.1`	`7.1`	`9.0`	`11`	`14`
`clang++`	`3.3`	`3.3`	`3.4`	`3.5`	`5.0`	`11.0`	`19+`	`19+`
`MSVC`	`19.0`	`19.0`	`19.10`	`19.0`	`19.15`	`19.15`	`19.29+`	`19.29`

C++23, C++26 are working in progress

Hello World

Hello World

C code with `printf`:

```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
}
```

`printf`
prints on standard output

C++ code with `streams`:

```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!\n";
}
```

`cout`
represents the standard output stream

Hello World

The previous example can be written with the global `std` namespace:

```cpp
#include <iostream>

using namespace std;
int main() {
    cout << "Hello World!\n";
}
```

Note: For sake of space and for improving the readability, we intentionally omit the `std` namespace in most slides

I/O Stream (std::cout) 1/3

`std::cout` is an example of *output* stream. Data is redirected to a destination, in this case the destination is the standard output

C:

```c
#include <stdio.h>
int main() {
    int    a    = 4;
    double b    = 3.0;
    char   c[] = "hello";
    printf("%d %f %s\n", a, b, c);
}
```

C++:

```cpp
#include <iostream>
int main() {
    int    a    = 4;
    double b    = 3.0;
    char   c[] = "hello";
    std::cout << a << " " << b << " " << c << "\n";
}
```

I/O Stream (Why should we prefer I/O stream?) 2/3

- **Type-safe**: The type of object provided to the I/O stream is known statically by the compiler. In contrast, `printf` uses `%` fields to figure out the types dynamically
- **Less error prone**: With I/O Stream, there are no redundant `%` tokens that have to be consistent with the actual objects passed to I/O stream. Removing redundancy removes a class of errors
- **Extensible**: The C++ I/O Stream mechanism allows new user-defined types to be passed to I/O stream without breaking existing code
- **Comparable performance**: If used correctly may be faster than C I/O (`printf`, `scanf`, etc.)

I/O Stream (Common C errors) 3/3

- Forget the number of parameters:

```
printf("long phrase %d long phrase %d", 3);
```

- Use the wrong format:

```
int a = 3;
...many lines of code...
printf(" %f", a);
```

- The `%c` conversion specifier does not automatically skip any leading white space:

```
scanf("%d", &var1);
scanf(" %c", &var2);
```

`std::print`

C++23 introduces an improved version of `printf` function `std::print` based on *formatter strings* that provides all benefits of C++ stream and is less verbose

```cpp
#include <print>

int main() {
    std::print("Hello World! {}, {}, {}\n", 3, 4ll, "aa");
    // print "Hello World! 3 4 aa"
}
```

This will be the default way to print when the C++23 standard will be widely adopted