

Modern C++ Programming

2. PREPARATION

Federico Busato

2025-01-19

1 Books and References

2 Slide Legend

3 What Editor/ IDE/Compiler Should I Use?

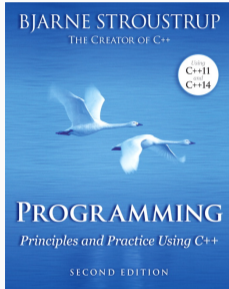
4 How to compile?

5 Hello World

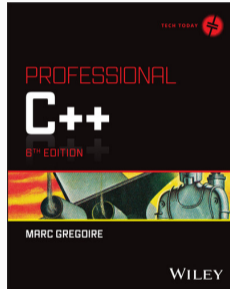
- I/O Stream

Books and References

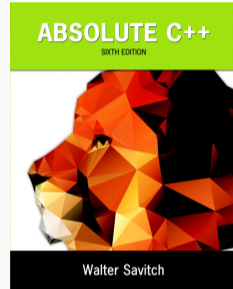
Suggested Books



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

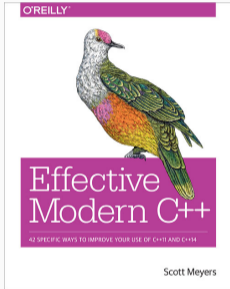


**Professional C++
(6th, C++23)**
S. J. Kleper, N. A. Solter, 2024



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

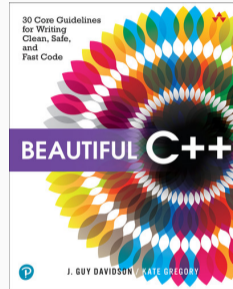
More Advanced Books



Effective Modern C++
S. Meyer, 2014



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



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

(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++ ↗

News:

- isocpp.org  (Standard C++ Foundation)
- [Reddit C++](#) 
- cpp.libhunt.com/newsletter/archive 
- [MeetingCpp Blogroll](#) 

Coding exercises:

- [HackerRank C++](#) 
- leetcode.com/problemset/algorithms 
- open.kattis.com 

Main conferences:

- CppCon [↗](#)
 (slides) [↗](#), (search engine) [↗](#)
- CppNorth [↗](#)
 (slides) [↗](#)
- Meeting C++ [↗](#)
 (slides) [↗](#)
- C++ on Sea [↗](#)
 (slides) [↗](#), (slides 2024) [↗](#)
- [isocpp.com conference list](#) [↗](#)

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

```
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++ (v19)

```
$ wget https://apt.llvm.org/llvm.sh  
$ chmod +x llvm.sh  
$ sudo ./llvm.sh 19  
$ clang++ --version
```

Install the Compiler on Windows

Microsoft Visual Studio

- Direct Installer: Visual Studio Community 2022

Clang on Windows

Two ways:

- Windows Subsystem for Linux (WSL)
 - Run → optionalfeatures
 - Select Windows Subsystem for Linux, Hyper-V, Virtual Machine Platform
 - Run → ms-windows-store: → Search and install Ubuntu 24.04 LTS
- Clang + MSVC Build Tools
 - Download Build Tools per Visual Studio
 - Install Desktop development with C++

Popular C++ IDE (Integrated Development Environment):

- **Microsoft Visual Studio** (MSVC) ([link](#)). Most popular IDE for Windows
- **Clion** ([link](#)). (free for student). Powerful IDE with a lot of options
- **QT-Creator** ([link](#)). Fast (written in C++), simple
- **XCode**. Default on Mac OS
- **Cevelop** (Eclipse) ([link](#))

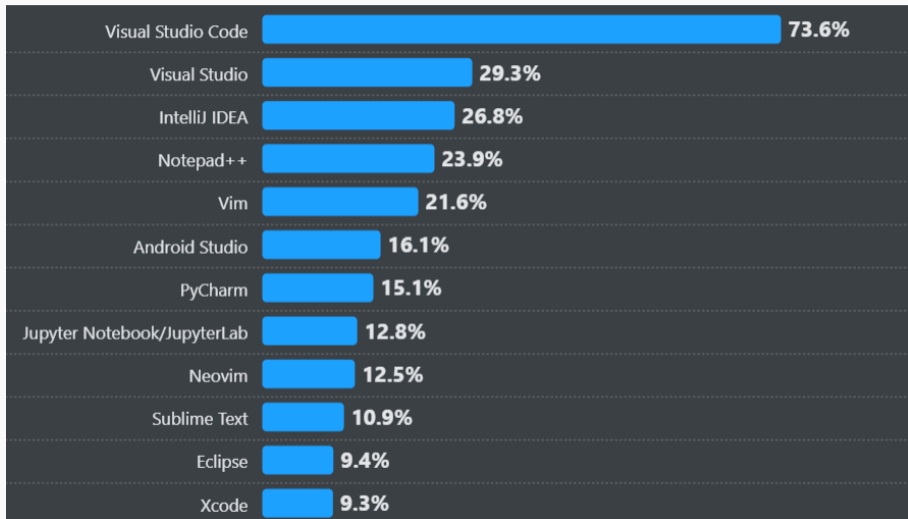
Standalone GUI-based coding editors:

- **Microsoft Visual Studio Code** (VSCode) ([link](#))
- **Sublime** ([link](#))
- **Lapce** ([link](#))
- **Zed** ([link](#))

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

- **Vim**
- **Emacs**
- **NeoVim** ([link](#))
- **Helix** ([link](#))

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



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.)

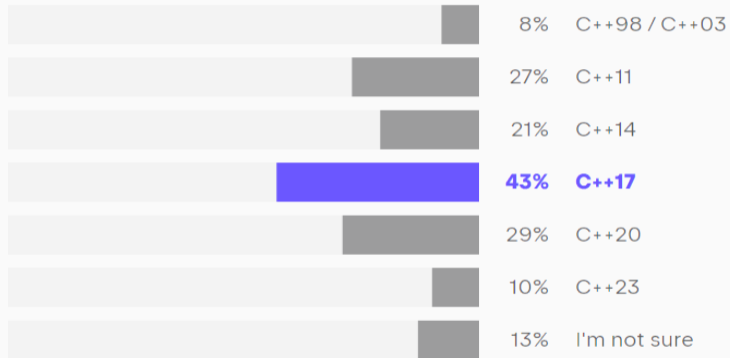
We can potentially compile a pure C program in C++26

*except for very minor deprecated features

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

C++ Standards Adoption



Hello World

C code with `printf` :

```
#include <stdio.h>

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

`printf`

prints on standard output

C++ code with `streams` :

```
#include <iostream>

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

`cout`

represents the standard output stream

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

```
#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

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

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++:

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

- **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.) .

- 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

```
#include <print>

int main() {
    std::print("Hello World! {}, {}, {}\n", 3, 411, "aa");
    // print "Hello World! 3 4 aa"
}
```

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