


Modern C++ — Introduction

Nils Wentzell
Oct 20, 2022

C++ Evolution

- Developed by Bjarne Stroustrup
- Release in 1985
-  standardized in 1998 — C++98
- Amended every three years:
C++11, C++14, C++17, C++20, C++23,...
- Extensions of core language (e.g. `auto`)
& standard library (e.g. `std::optional`)
- Backward compatible



C++ Evolution

Many directions

- **Simpler** syntax
- More Python-like
- Generic programming
- Functional programming
- Compile-time programming
- Avoiding raw memory pointers
- Concurrency

Today's Menu



- `auto`
- Ranges
- Return Value Optimization

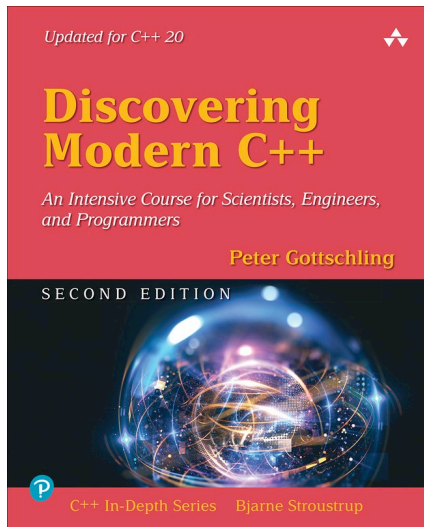


- Lambda functions
- Standard Template Library (STL)

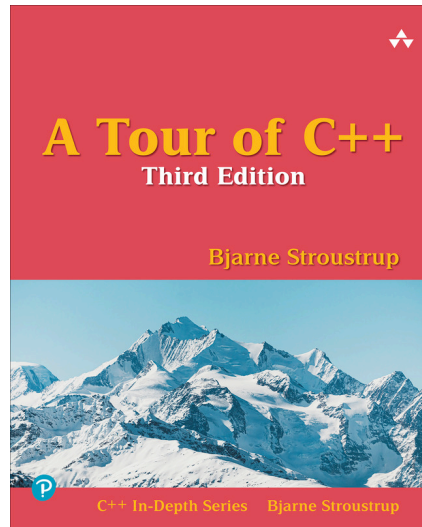


- Generic programming
- Concepts

Book Recommendations



Beginner Course for Scientists



Modern C++ Overview

cppreference.com

- Excellent Reference Documentation of Language Features & Standard Library
- Executable Code-Samples
- Lists C++ Version Requirements

cppreference.com

Create account Search

Page Discussion View Edit History

C++ Utilities library **std::optional**

std::optional

Defined in header `<optional>`

```
template< class T > (since C++17)
class optional;
```

The class template `std::optional` manages an *optional* contained value, i.e. a value that may or may not be present.

Example

Run this code

```
#include <string>
#include <functional>
#include <iostream>
#include <optional>

// optional can be used as the return type of a factory that may fail
std::optional<std::string> create(bool b) {
    if (b)
        return "Godzilla";
    return {};
}
```

cppreference.com

Create account Search

Page Discussion View View source History

CppCon 2022
It's the annual, week-long gathering for the entire C++ community. [Register now!](#)

C++ reference

C++98, C++03, C++11, C++14, C++17, C++20, C++23 | Compiler support C++11, C++14, C++17, C++20, C++23

| | | |
|--|---|---|
| Freestanding implementations Language Basic concepts Keywords Preprocessor Expressions Declaration Initialization Functions Statements Classes Overloading Templates Exceptions Standard library (headers) Named requirements Feature test macros (C++20) Language support library Source code information (C++20) Type support Program utilities Coroutine support (C++20) Three-way comparison (C++20) numeric limits – type_info initializer_list (C++11) Concepts library (C++20) Metaprogramming library (C++11) Type traits – ratio integer_sequence (C++14) | Diagnostics library basic_stacktrace (C++23) Memory management library unique_ptr (C++11) shared_ptr (C++11) General utilities library Function objects – hash (C++11) Utility functions pair – tuple (C++11) optional (C++17) – expected (C++23) variant (C++17) – any (C++17) String conversions (C++17) Formatting (C++20) Bit manipulation (C++20) Strings library basic_string basic_string_view (C++17) Null-terminated strings: byte – multibyte – wide Containers library array (C++11) – vector – deque list – forward_list (C++11) map – multimap set – multiset unordered_map (C++11) unordered_multimap (C++11) unordered_set (C++11) unordered_multiset (C++11) stack – queue – priority_queue span (C++20) – mdspan (C++23) | Iterators library Ranges library (C++20) Algorithms library Constrained algorithms (C++20) Numerics library Common math functions Mathematical special functions (C++17) Mathematical constants (C++20) Numeric algorithms Pseudo-random number generation Floating-point environment (C++11) complex – valarray Date and time library Localizations library Input/output library Stream-based I/O Synchronized output (C++20) I/O manipulators Filesystem library (C++17) Regular expressions library (C++11) basic_regex – algorithms Concurrency support library (C++11) atomic – atomic_flag atomic_ref (C++20) thread – jthread (C++20) mutex condition_variable future – promise |
|--|---|---|

Compiling Modern C++

file: main.cpp

```
#include <iostream>
#include <vector>

int main() {
    auto words = std::vector{"Hello", " World"};
    for(auto& w: words) std::cout << w;
}
```

g++ -std=c++20 main.cpp

```
→ ls
a.out* main.cpp
→ ./a.out
Hello World
```

- Support by all major compilers
 - GCC
 - Microsoft MSVC
 - Clang & Apple Clang
 - IntelLLVM (OneAPI)
- For Details see

en.cppreference.com/w/cpp/compiler_support



C++20 core language features

| C++20 feature | Paper(s) | GCC | Clang | MSVC | Apple Clang | EDG eccp | Intel C++ |
|--------------------------------|--------------------|-------------------------------------|-------|--------|-------------|----------|-----------|
| Allow lambda-capture [=, this] | P0409R2 | 8 | 6 | 19.22* | 10.0.0* | 5.1 | 2021.1 |
| __VA_OPT__ | P0306R4 P1042R1 | 8 (partial)* 10 (partial)* 12 | 9 | 19.25* | 11.0.3* | 5.1 | 2021.1 |

Compiler Explorer — godbolt.org

Code Compiler & Version Flags Program output

The screenshot displays the Compiler Explorer interface with the following components:

- Code:** C++ source code in a file named "C++ source #1". The code includes `<iostream>` and `<vector>`, and contains a `main` function that creates a vector with "Hello" and "World", and prints them. The code is highlighted with an orange box.
- Compiler & Version:** The compiler is set to "x86-64 gcc 12.2" (C++, Editor #1, Compiler #1). This selection is highlighted with an orange box.
- Flags:** The compiler flags are set to "-O3 -std=c++20". This selection is highlighted with an orange box.
- Program output:** The output window shows the results of the compilation and execution. It indicates that the ASM generation compiler returned 0, the Execution build compiler returned 0, and the Program returned 0. The output "Hello World" is displayed. This section is highlighted with an orange box.

```
#include <iostream>
#include <vector>

int main() {
    auto words = std::vector{"Hello", "World"};
    for(auto& w: words) std::cout << w;
}
```

```
.LC0:
2      .string "Hello"
3      .LC1:
4      .string "World"
5  main:
6      push    r12
7      mov     edi, 16
8      push    rbp
9      xor     ebp, ebp
10     push    rbx
11     call    operator new(unsigned long)
12     mov     r12, rax
13     mov     eax, OFFSET FLAT:.LC0
14     movq    xmm0, rax
15     movhps  xmm0, QWORD PTR .LC2[rip]
16     movups  XMMWORD PTR [r12], xmm0
17     jmp     .L3
18  .L2:
19     mov     rdi, rbx
20     call    strlen
21     mov     rsi, rbx
22     mov     edi, OFFSET FLAT:std::cout
23     mov     rdx, rax
24     call    std::basic_ostream<char, std::ch
25     cmp     rbp, 8
26     je      .L13
```

Output of x86-64 gcc 12.2 (Compiler #1)

ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
Hello World

Compiler Explorer — godbolt.org

The screenshot displays the Godbolt Compiler Explorer interface. On the left, the C++ source code is shown in a text editor. The middle pane displays the generated assembly code for x86-64 gcc 12.2. On the right, a share menu is open, highlighting the 'Short Link' option. An arrow points from the 'Short Link' option to a callout box containing the shareable URL.

C++ source code:

```
1 #include <iostream>
2 #include <vector>
3
4 int main() {
5     auto words = std::vector{"Hello", " World"};
6     for(auto& w: words) std::cout << w;
7 }
```

Assembly output (x86-64 gcc 12.2):

```
1 .LC0:
2     .string "Hello"
3 .LC1:
4     .string " World"
5 main:
6     push    r12
7     mov     edi, 16
8     push    rbp
9     xor     ebp, ebp
10    push    rbx
11    call     operator new(unsigned long)
12    mov     r12, rax
13    mov     eax, OFFSET FLAT:.LC0
14    movq    xmm0, rax
15    movhps  xmm0, QWORD PTR .LC2[rip]
16    movups  XMMWORD PTR [r12], xmm0
17    jmp     .L3
18 .L2:
19    mov     rdi, rbx
20    call     strlen
21    mov     rsi, rbx
22    mov     edi, OFFSET FLAT:std::cout
23    mov     rdx, rax
24    call     std::basic_ostream<char, std::ch
25    cmp     rbp, 8
26    je      .L13
```

Share menu options:

- Short Link
- Full Link
- Embed in iframe

Shareable URL:

godbolt.org/z/WMxf4os1h

Modern C++ Eco-System

Developing

- `git` — Version Control
- `cmake` — Build Tool
- `clang-format` — Code Formatting
- `googletest` — Testing Framework
- `clang-tidy` — Linter
- `clangd` — Language Server
- `vscode` — Plug-In based Editor
- `clion` — Full-fledged IDE

Debugging

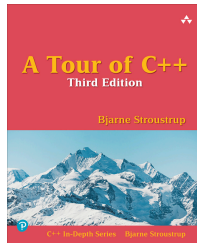
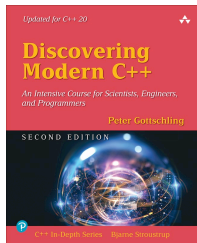
- `lldb` — Debugger
- `LLVM Sanitizers` — Dynamic Analyzer

Performance

- `gperftools` — CPU & Memory Profiling
- `google/benchmark` — Benchmarking

Useful Resources

Books



Podcasts

- cpp.chat
- cppcast.com

Videos

- youtube.com/user/CppCon/videos
- youtube.com/c/lefticus1/videos

Web-Tools

- godbolt.org
- quick-bench.com
- cppinsights.io

News

- isocpp.org
- reddit.com/r/cpp

Blogs

- fluentcpp.com
- cppstories.com
- abseil.io/tips

Cheat Sheets

- hackingcpp.com/cpp/cheat_sheets.html

More Modern C++ Resources: github.com/fffaraz/awesome-cpp