Idiomatic Modern C++ for Linux

Week 1: Let’s Go to Class

Today’s agenda

- About the class
- Motivation
- About Docker
- Intro to CMake
- Our very first C++ Program
- Maybe the start of the next lesson if we blaze thru this one in under an hour lol ¯_(ツ)_/¯

About the class

- Learn how to write performant, safe C++ libraries and applications through lecture, discussion, and practical buildable / tweakable code examples
- Learn common C++ build tools and how to organize and build your C++ projects using CMake
- Learn powerful language features and how to leverage them in your projects
- Learn pitfalls of the language and how to avoid them

About the class

- Language standard: C++17
- Compiler – gcc13
- All code examples will be built and run inside the docker container
- Code examples to illustrate concepts

Why care about C++?

- One of the fastest programming languages
- Popular
- Significant developer community
- C++ == Salary++
- Portable
- Enormous ecosystem. Abundance of libraries

Non-exhaustive list of things that you can make with C++

- Robots
- Operating systems
- Video games
- Embedded systems / IoT
- Hardware drivers
- Financial tools
- Medical devices
- Search engines
- Databases
- Navigation systems
- Machine Learning
- Web browsers
- Flight software
- Video editors
- Simulators
- Scientific research

About Docker

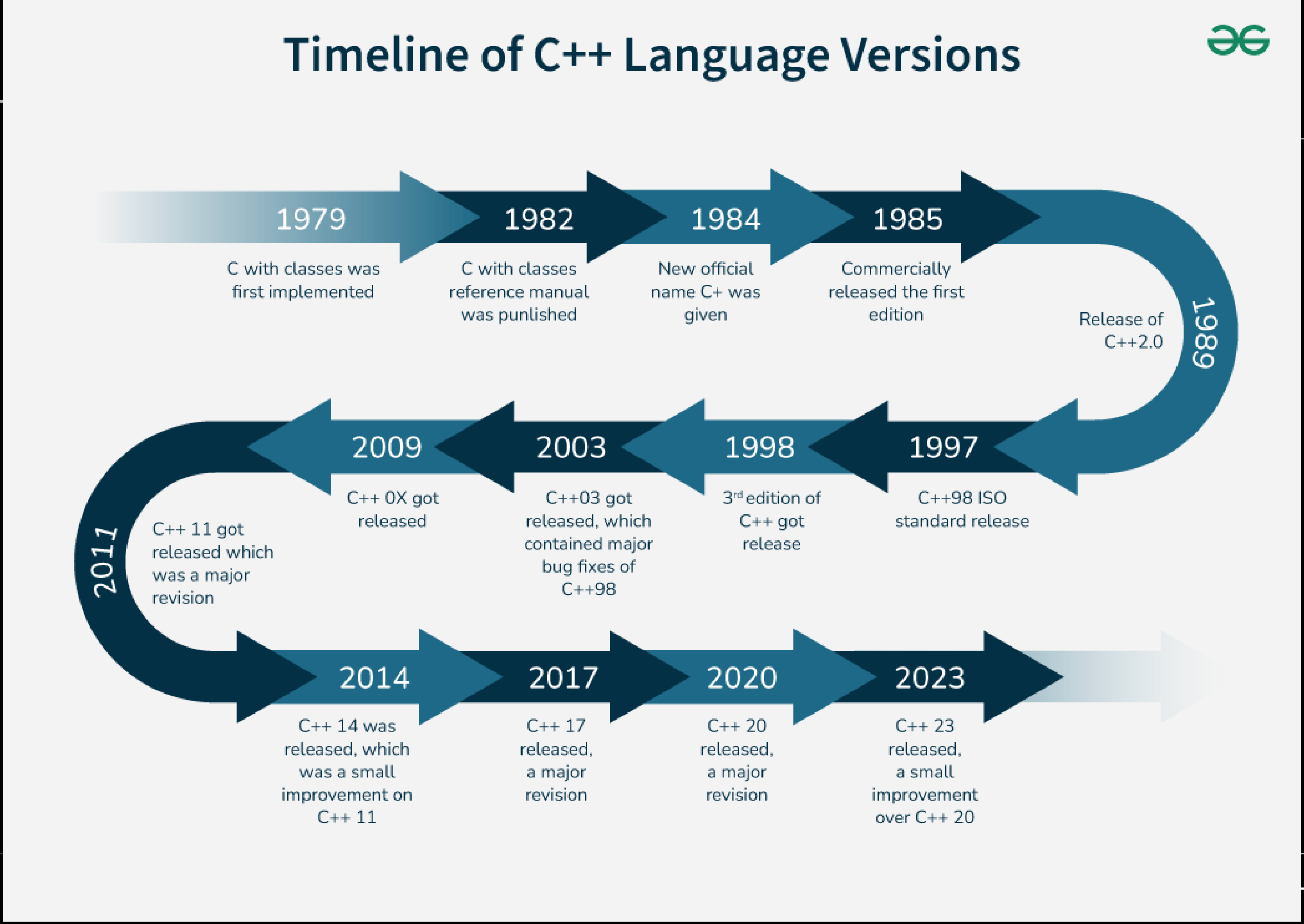
- Docker is an OS–level virtualization platform
- Allows you to run your applications *inside a container*
- Avoid dirtying your host machine with dependencies
- Consistent, disposable build and runtime environments for all
- No messing with system dependency mismatches
- Container build / run scripts are available for student convenience (on linux host machines)
- Unless you wish to add new system dependencies to the container, you should only need to build it once

About CMake

- free, open source cross-platform development tool for building software
- “meta-build” system: Generates config files for gnu make / ninja / visual studio / xcode
- light scripting language
- Automates testing, packaging and installation
- Vast adoption in industry, academia and open source projects

About C++

- cross-platform, general purpose compiled programming language
- Invented by Bjarne Stroustrup and Bell Labs
- “Modern” C++: C++11 and beyond



Timeline of C++ Language Versions

1979
C with classes was first implemented

1982
C with classes reference manual was punlished

1984
New official name C+ was given

1985
Commercially released the first edition

1989
Release of C++2.0

1997
C++98 ISO standard release

1998
3rd edition of C++ got release

2003
C++03 got released, which contained major bug fixes of C++98

2009
C++ 0X got released

2011
C++ 11 got released which was a major revision

2014
C++ 14 was released, which was a small improvement on C++ 11

2017
C++ 17 released, a major revision

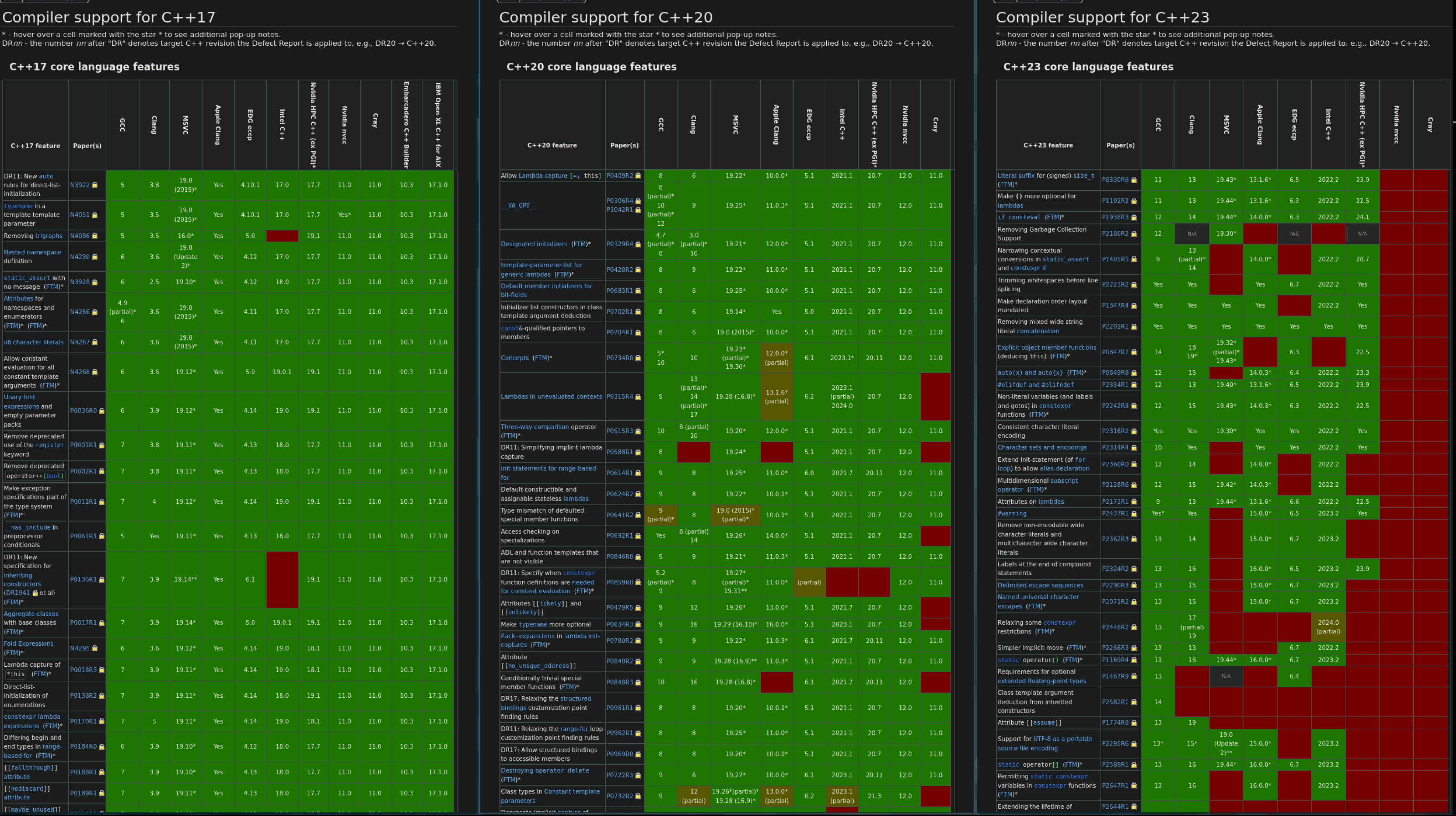
2020
C++ 20 released, a major revision

2023
C++ 23 released, a small improvement over C++ 20

Why C++17 and not C++20 or C++23?

- C++17 is the default standard used in our docker container

  (Check via “***c++ –dM –E –x c++ /dev/null | grep –F __cplusplus***”)
- Many compilers still do not support a lot of features from newer C++ versions (demonstrated on the next slide)
- In particular, nvcc (nvidia cross compiler, for gpu programming) does not support C++23 yet
- We’ll still discuss relevant examples and changes in newer versions of the language when applicable



Compiler support for C++17

* - hover over a cell marked with the star * to see additional pop-up notes.
DR*nn* - the number *nn* after "DR" denotes target C++ revision the Defect Report is applied to, e.g., DR20 → C++20.

C++17 core language features

C++17 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	Nvidia HPC C++ (ex PGI)*	Nvidia nvcc	Cray	Embarcadero C++ Builder	IBM Open XL C++ for AIX
DR11: New auto rules for direct-list-initialization	N3922	5	3.8	19.0 (2015)*	Yes	4.10.1	17.0	17.7	11.0	11.0	10.3	17.1.0
typename in a template template parameter	N4051	5	3.5	19.0 (2015)*	Yes	4.10.1	17.0	17.7	Yes*	11.0	10.3	17.1.0
Removing trigraphs	N4086	5	3.5	16.0*	Yes	5.0	🟥	19.1	11.0	11.0	10.3	17.1.0
Nested namespace definition	N4230	6	3.6	19.0 (Update 3)*	Yes	4.12	17.0	17.7	11.0	11.0	10.3	17.1.0
`static_assert` with no message (FTM)*	N3928	6	2.5	19.10*	Yes	4.12	18.0	17.7	11.0	11.0	10.3	17.1.0
Attributes for namespaces and enumerators (FTM)* (FTM)*	N4266	4.9 (partial)* 6	3.6	19.0 (2015)*	Yes	4.11	17.0	17.7	11.0	11.0	10.3	17.1.0
u8 character literals	N4267	6	3.6	19.0 (2015)*	Yes	4.11	17.0	17.7	11.0	11.0	10.3	17.1.0
Allow constant evaluation for all constant template arguments (FTM)*	N4268	6	3.6	19.12*	Yes	5.0	19.0.1	19.1	11.0	11.0	10.3	17.1.0
Unary fold expressions and empty parameter packs	P0036R0	6	3.9	19.12*	Yes	4.14	19.0	19.1	11.0	11.0	10.3	17.1.0
Remove deprecated use of the `register` keyword	P0001R1	7	3.8	19.11*	Yes	4.13	18.0	17.7	11.0	11.0	10.3	17.1.0
Remove deprecated `operator++(bool)`	P0002R1	7	3.8	19.11*	Yes	4.13	18.0	17.7	11.0	11.0	10.3	17.1.0
Make exception specifications part of the type system (FTM)*	P0012R1	7	4	19.12*	Yes	4.14	19.0	19.1	11.0	11.0	10.3	17.1.0
`__has_include` in preprocessor conditionals	P0061R1	5	Yes	19.11*	Yes	4.13	18.0	17.7	11.0	11.0	10.3	17.1.0
DR11: New specification for inheriting constructors (DR1941 et al) (FTM)*	P0136R1	7	3.9	19.14**	Yes	6.1	🟥	19.1	11.0	11.0	10.3	17.1.0
Aggregate classes with base classes (FTM)*	P0017R1	7	3.9	19.14*	Yes	5.0	19.0.1	19.1	11.0	11.0	10.3	17.1.0
Fold Expressions (FTM)*	N4295	6	3.6	19.12*	Yes	4.14	19.0	18.1	11.0	11.0	10.3	17.1.0
Lambda capture of `*this` (FTM)*	P0018R3	7	3.9	19.11*	Yes	4.14	19.0	18.1	11.0	11.0	10.3	17.1.0
Direct-list-initialization of enumerations	P0138R2	7	3.9	19.11*	Yes	4.14	18.0	19.1	11.0	11.0	10.3	17.1.0
`constexpr` lambda expressions (FTM)*	P0170R1	7	5	19.11*	Yes	4.14	19.0	18.1	11.0	11.0	10.3	17.1.0
Differing begin and end types in range-based for (FTM)*	P0184R0	6	3.9	19.10*	Yes	4.12	18.0	17.7	11.0	11.0	10.3	17.1.0
`[[fallthrough]]` attribute	P0188R1	7	3.9	19.10*	Yes	4.13	18.0	17.7	11.0	11.0	10.3	17.1.0
`[[nodiscard]]` attribute	P0189R1	7	3.9	19.11*	Yes	4.13	18.0	17.7	11.0	11.0	10.3	17.1.0
`[[maybe_unused]]`												

Compiler support for C++20

* - hover over a cell marked with the star * to see additional pop-up notes.
DR*nn* - the number *nn* after "DR" denotes target C++ revision the Defect Report is applied to, e.g., DR20 → C++20.

C++20 core language features

C++20 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	Nvidia HPC C++ (ex PGI)*	Nvidia nvcc	Cray
Allow Lambda capture `[=, this]`	P0409R2	8	6	19.22*	10.0.0*	5.1	2021.1	20.7	12.0	11.0
`__VA_OPT__`	P0306R4 P1042R1	8 (partial)* 10 (partial)* 12	9	19.25*	11.0.3*	5.1	2021.1	20.7	12.0	11.0
Designated initializers (FTM)*	P0329R4	4.7 (partial)* 8	3.0 (partial)* 10	19.21*	12.0.0*	5.1	2021.1	20.7	12.0	11.0
template-parameter-list for generic lambdas (FTM)*	P0428R2	8	9	19.22*	11.0.0*	5.1	2021.1	20.7	12.0	11.0
Default member initializers for bit-fields	P0683R1	8	6	19.25*	10.0.0*	5.1	2021.1	20.7	12.0	11.0
Initializer list constructors in class template argument deduction	P0702R1	8	6	19.14*	Yes	5.0	2021.1	20.7	12.0	11.0
`const`&-qualified pointers to members	P0704R1	8	6	19.0 (2015)*	10.0.0*	5.1	2021.1	20.7	12.0	11.0
Concepts (FTM)*	P0734R0	5* 10	10	19.23* (partial)* 19.30*	12.0.0* (partial)	6.1	2023.1*	20.11	12.0	11.0
Lambdas in unevaluated contexts	P0315R4	9	13 (partial)* 14 (partial)* 17	19.28 (16.8)*	13.1.6* (partial)	6.2	2023.1 (partial) 2024.0	20.7	12.0	🟥
Three-way comparison operator (FTM)*	P0515R3	10	8 (partial) 10	19.20*	12.0.0*	5.1	2021.1	20.7	12.0	11.0
DR11: Simplifying implicit lambda capture	P0588R1	8	🟥	19.24*	🟥	5.1	2021.1	20.7	12.0	🟥
init-statements for range-based for	P0614R1	9	8	19.25*	11.0.0*	6.0	2021.7	20.11	12.0	11.0
Default constructible and assignable stateless lambdas	P0624R2	9	8	19.22*	10.0.1*	5.1	2021.1	20.7	12.0	11.0
Type mismatch of defaulted special member functions	P0641R2	9 (partial)*	8	19.0 (2015)* (partial)*	10.0.1*	5.1	2021.1	20.7	12.0	11.0
Access checking on specializations	P0692R1	Yes	8 (partial) 14	19.26*	14.0.0*	5.1	2021.1	20.7	12.0	🟥
ADL and function templates that are not visible	P0846R0	9	9	19.21*	11.0.3*	5.1	2021.1	20.7	12.0	11.0
DR11: Specify when `constexpr` function definitions are needed for constant evaluation (FTM)*	P0859R0	5.2 (partial)* 9	8	19.27* (partial)* 19.31**	11.0.0*	(partial)	🟥	🟥	12.0	11.0
Attributes `[[likely]]` and `[[unlikely]]`	P0479R5	9	12	19.26*	13.0.0*	5.1	2021.7	20.7	12.0	🟥
Make `typename` more optional	P0634R3	9	16	19.29 (16.10)*	16.0.0*	5.1	2023.1	20.7	12.0	🟥
`Pack-expansions` in lambda init-captures (FTM)*	P0780R2	9	9	19.22*	11.0.3*	6.1	2021.7	20.11	12.0	11.0
Attribute `[[no_unique_address]]`	P0840R2	9	9	19.28 (16.9)**	11.0.3*	5.1	2021.1	20.7	12.0	11.0
Conditionally trivial special member functions (FTM)*	P0848R3	10	16	19.28 (16.8)*	🟥	6.1	2021.7	20.11	12.0	🟥
DR17: Relaxing the structured bindings customization point finding rules	P0961R1	8	8	19.20*	10.0.1*	5.1	2021.1	20.7	12.0	11.0
DR11: Relaxing the range-for loop customization point finding rules	P0962R1	8	8	19.25*	11.0.0*	5.1	2021.1	20.7	12.0	11.0
DR17: Allow structured bindings to accessible members	P0969R0	8	8	19.20*	10.0.1*	5.1	2021.1	20.7	12.0	11.0
Destroying `operator delete` (FTM)*	P0722R3	9	6	19.27*	10.0.0*	6.1	2023.1	20.11	12.0	11.0
Class types in Constant template parameters	P0732R2	9	12 (partial)	19.26*(partial)* 19.28 (16.9)*	13.0.0* (partial)	6.2	2023.1 (partial)	21.3	12.0	🟥

Compiler support for C++23

* - hover over a cell marked with the star * to see additional pop-up notes.
DR*nn* - the number *nn* after "DR" denotes target C++ revision the Defect Report is applied to, e.g., DR20 → C++20.

C++23 core language features

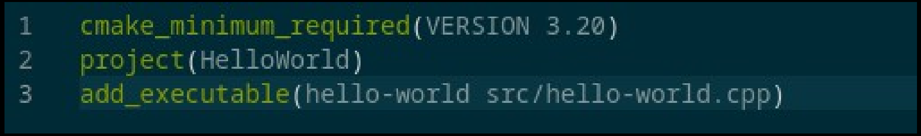
C++23 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	Nvidia HPC C++ (ex PGI)*	Nvidia nvcc	Cray
Literal suffix for (signed) `size_t` (FTM)*	P0330R8	11	13	19.43*	13.1.6*	6.5	2022.2	23.9	🟥	🟥
Make `()` more optional for lambdas	P1102R2	11	13	19.44*	13.1.6*	6.3	2022.2	22.5	🟥	🟥
`if consteval` (FTM)*	P1938R3	12	14	19.44*	14.0.0*	6.3	2022.2	24.1	🟥	🟥
Removing Garbage Collection Support	P2186R2	12	N/A	19.30*	🟥	N/A	🟥	N/A	🟥	🟥
Narrowing contextual conversions in `static_assert` and constexpr if	P1401R5	9	13 (partial)* 14	🟥	14.0.0*	🟥	2022.2	20.7	🟥	🟥
Trimming whitespaces before line splicing	P2223R2	Yes	Yes	🟥	Yes	6.7	2022.2	Yes	🟥	🟥
Make declaration order layout mandated	P1847R4	Yes	Yes	Yes	Yes	🟥	2022.2	Yes	🟥	🟥
Removing mixed wide string literal concatenation	P2201R1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	🟥	🟥
Explicit object member functions (deducing `this`) (FTM)*	P0847R7	14	18 19*	19.32* (partial)* 19.43*	🟥	6.3	🟥	22.5	🟥	🟥
`auto(x) and auto{x}` (FTM)*	P0849R8	12	15	🟥	14.0.3*	6.4	2022.2	23.3	🟥	🟥
`#elifdef and #elifndef`	P2334R1	12	13	19.40*	13.1.6*	6.5	2022.2	23.9	🟥	🟥
Non-literal variables (and labels and gotos) in `constexpr` functions (FTM)*	P2242R3	12	15	19.43*	14.0.3*	6.3	2022.2	22.5	🟥	🟥
Consistent character literal encoding	P2316R2	Yes	Yes	19.30*	Yes	Yes	2022.2	Yes	🟥	🟥
Character sets and encodings	P2314R4	10	Yes	🟥	Yes	Yes	2022.2	Yes	🟥	🟥
Extend init-statement (of for loop) to allow alias-declaration	P2360R0	12	14	🟥	14.0.0*	🟥	2022.2	🟥	🟥	🟥
Multidimensional subscript operator (FTM)*	P2128R6	12	15	19.42*	14.0.3*	🟥	2022.2	🟥	🟥	🟥
Attributes on lambdas	P2173R1	9	13	19.44*	13.1.6*	6.6	2022.2	22.5	🟥	🟥
`#warning`	P2437R1	Yes*	Yes	🟥	15.0.0*	6.5	2023.2	Yes	🟥	🟥
Remove non-encodable wide character literals and multicharacter wide character literals	P2362R3	13	14	🟥	15.0.0*	6.7	2023.2	🟥	🟥	🟥
Labels at the end of compound statements	P2324R2	13	16	🟥	16.0.0*	6.5	2023.2	23.9	🟥	🟥
Delimited escape sequences	P2290R3	13	15	🟥	15.0.0*	6.7	2023.2	🟥	🟥	🟥
Named universal character escapes (FTM)*	P2071R2	13	15	🟥	15.0.0*	6.7	2023.2	🟥	🟥	🟥
Relaxing some `constexpr` restrictions (FTM)*	P2448R2	13	17 (partial) 19	🟥	🟥	🟥	2024.0 (partial)	🟥	🟥	🟥
Simpler implicit move (FTM)*	P2266R3	13	13	🟥	🟥	6.7	2022.2	🟥	🟥	🟥
`static operator()` (FTM)*	P1169R4	13	16	19.44*	16.0.0*	6.7	2023.2	🟥	🟥	🟥
Requirements for optional extended floating-point types	P1467R9	13	🟥	N/A	🟥	6.4	🟥	🟥	🟥	🟥
Class template argument deduction from inherited constructors	P2582R1	14	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Attribute `[[assume]]`	P1774R8	13	19	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Support for UTF-8 as a portable source file encoding	P2295R6	13*	15*	19.0 (Update 2)**	15.0.0*	🟥	2023.2	🟥	🟥	🟥
`static operator[]` (FTM)*	P2589R1	13	16	19.44*	16.0.0*	6.7	2023.2	🟥	🟥	🟥
Permitting `static constexpr` variables in `constexpr` functions (FTM)*	P2647R1	13	16	🟥	16.0.0*	🟥	2023.2	🟥	🟥	🟥
Extending the lifetime of	P2644R1			🟥	🟥	🟥	🟥	🟥	🟥	🟥

Let’s get started!

- Clone the git repo: https://github.com/skyegalaxy/learning-cpp
- Build the docker container
- We’ll try building the example code and viewing the output

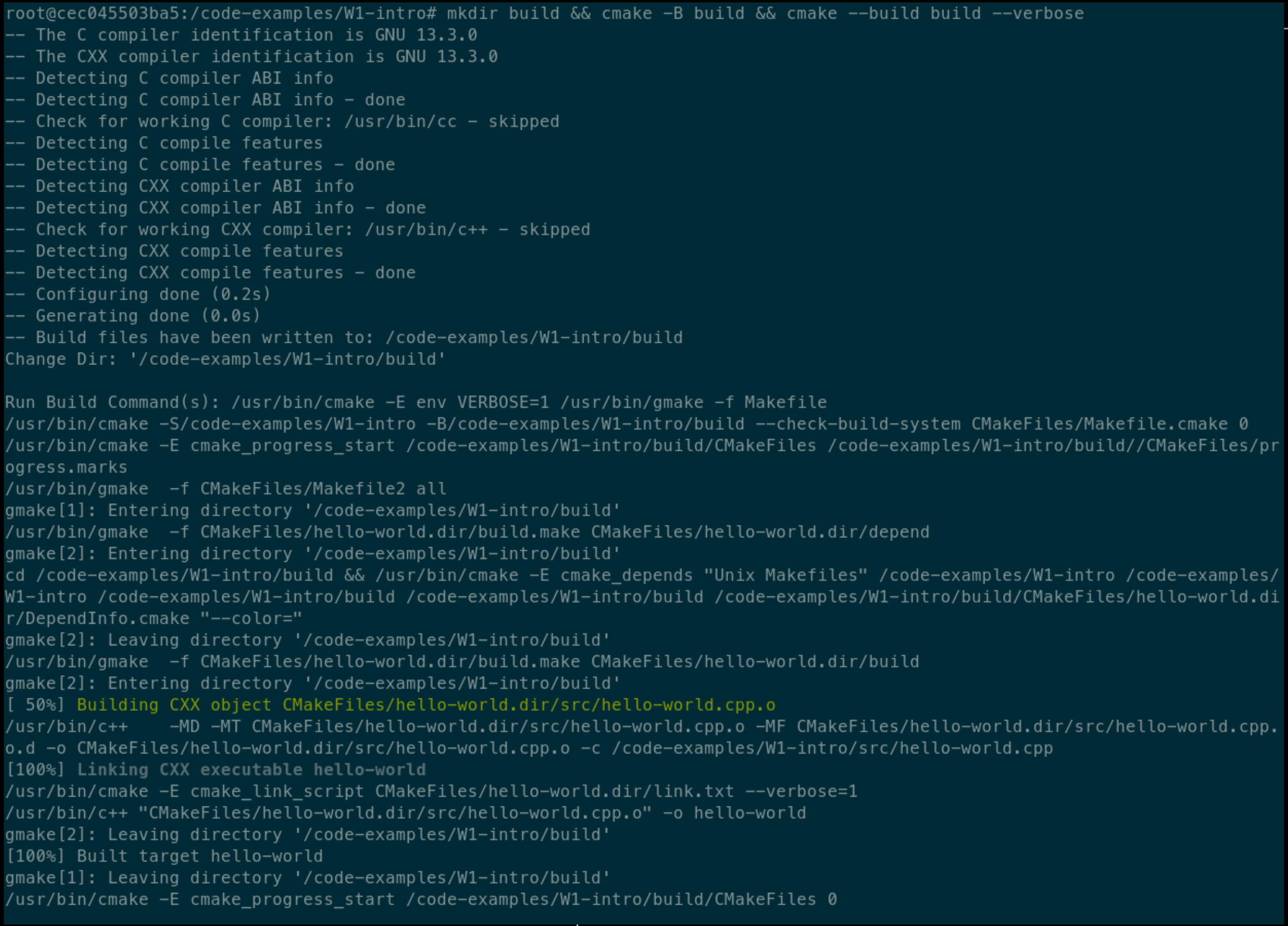
Examining our CMakeLists.txt

- This is obviously a very basic example
- As projects grow in complexity, we’ll add new tools onto our CMake toolbelt and discuss them together



```
cmake_minimum_required(VERSION 3.20)
project(HelloWorld)
add_executable(hello-world src/hello-world.cpp)
```

Using CMake to build the examples



```
root@cec045503ba5:/code-examples/W1-intro# mkdir build && cmake -B build && cmake --build build --verbose
-- The C compiler identification is GNU 13.3.0
-- The CXX compiler identification is GNU 13.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.2s)
-- Generating done (0.0s)
-- Build files have been written to: /code-examples/W1-intro/build
Change Dir: '/code-examples/W1-intro/build'

Run Build Command(s): /usr/bin/cmake -E env VERBOSE=1 /usr/bin/gmake -f Makefile
/usr/bin/cmake -S/code-examples/W1-intro -B/code-examples/W1-intro/build --check-build-system CMakeFiles/Makefile.cmake 0
/usr/bin/cmake -E cmake_progress_start /code-examples/W1-intro/build/CMakeFiles /code-examples/W1-intro/build//CMakeFiles/pr
ogress.marks
/usr/bin/gmake  -f CMakeFiles/Makefile2 all
gmake[1]: Entering directory '/code-examples/W1-intro/build'
/usr/bin/gmake  -f CMakeFiles/hello-world.dir/build.make CMakeFiles/hello-world.dir/depend
gmake[2]: Entering directory '/code-examples/W1-intro/build'
cd /code-examples/W1-intro/build && /usr/bin/cmake -E cmake_depends "Unix Makefiles" /code-examples/W1-intro /code-examples/
W1-intro /code-examples/W1-intro/build /code-examples/W1-intro/build /code-examples/W1-intro/build/CMakeFiles/hello-world.di
r/DependInfo.cmake "--color="
gmake[2]: Leaving directory '/code-examples/W1-intro/build'
/usr/bin/gmake  -f CMakeFiles/hello-world.dir/build.make CMakeFiles/hello-world.dir/build
gmake[2]: Entering directory '/code-examples/W1-intro/build'
[ 50%] Building CXX object CMakeFiles/hello-world.dir/src/hello-world.cpp.o
/usr/bin/c++    -MD -MT CMakeFiles/hello-world.dir/src/hello-world.cpp.o -MF CMakeFiles/hello-world.dir/src/hello-world.cpp.
o.d -o CMakeFiles/hello-world.dir/src/hello-world.cpp.o -c /code-examples/W1-intro/src/hello-world.cpp
[100%] Linking CXX executable hello-world
/usr/bin/cmake -E cmake_link_script CMakeFiles/hello-world.dir/link.txt --verbose=1
/usr/bin/c++ "CMakeFiles/hello-world.dir/src/hello-world.cpp.o" -o hello-world
gmake[2]: Leaving directory '/code-examples/W1-intro/build'
[100%] Built target hello-world
gmake[1]: Leaving directory '/code-examples/W1-intro/build'
/usr/bin/cmake -E cmake_progress_start /code-examples/W1-intro/build/CMakeFiles 0
```

Using CMake to build the examples

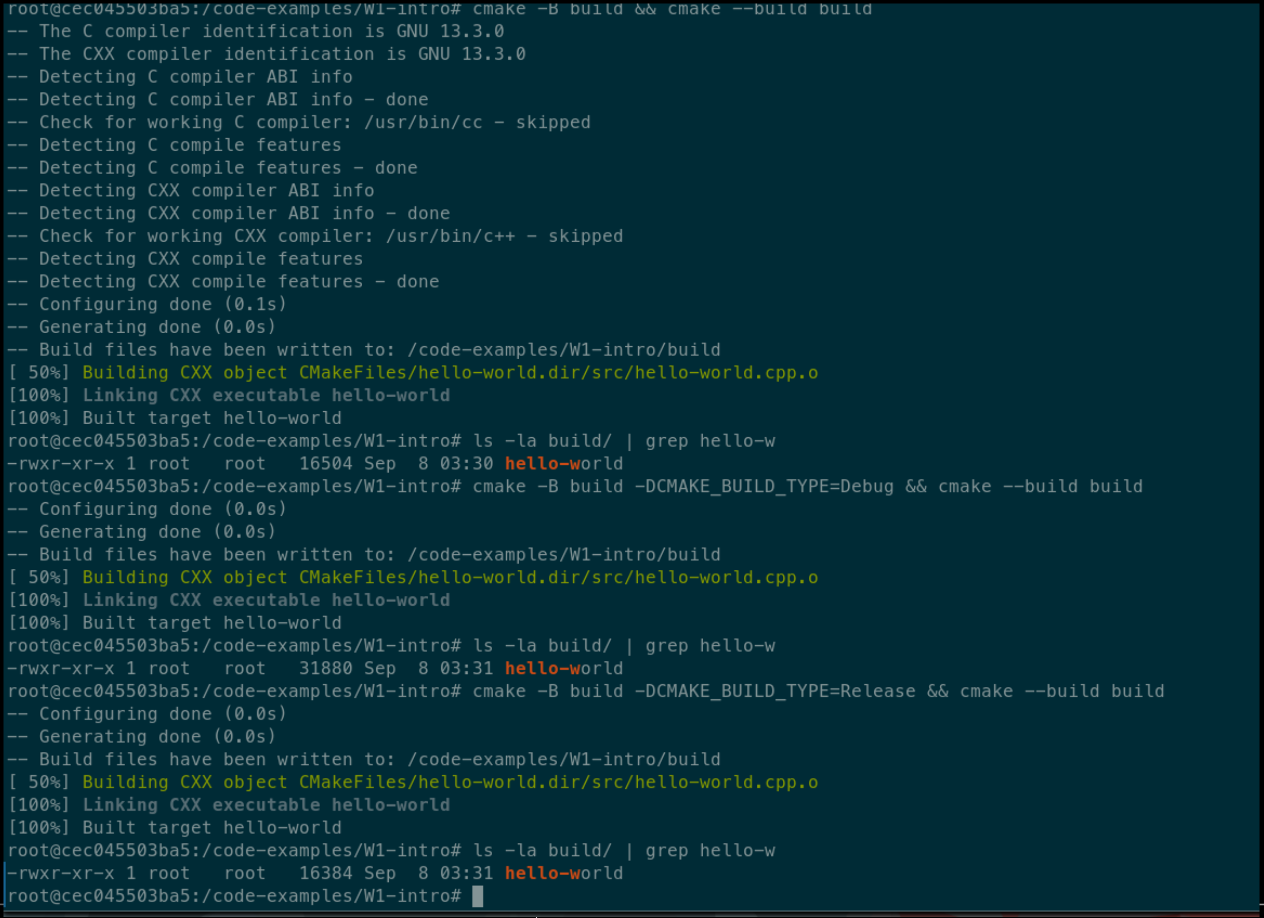
cmake –B build: run cmake and configure build, store results in the **build** folder

cmake ––build build: Actually build the binaries, invoking GCC (or other compiler like clang)

Building in ***Debug*** mode adds debug symbols. Extremely useful in gdb
cmake –B build–debug –D CMAKE_BUILD_TYPE=Release && cmake –build build–debug

Building in ***Release*** mode strips all debug symbols and performs more extensive compiler optimizations

Using CMake to build the examples



```
root@cec045503ba5:/code-examples/W1-intro# cmake -B build && cmake --build build
-- The C compiler identification is GNU 13.3.0
-- The CXX compiler identification is GNU 13.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.1s)
-- Generating done (0.0s)
-- Build files have been written to: /code-examples/W1-intro/build
[ 50%] Building CXX object CMakeFiles/hello-world.dir/src/hello-world.cpp.o
[100%] Linking CXX executable hello-world
[100%] Built target hello-world
root@cec045503ba5:/code-examples/W1-intro# ls -la build/ | grep hello-w
-rwxr-xr-x 1 root   root   16504 Sep  8 03:30 hello-world
root@cec045503ba5:/code-examples/W1-intro# cmake -B build -DCMAKE_BUILD_TYPE=Debug && cmake --build build
-- Configuring done (0.0s)
-- Generating done (0.0s)
-- Build files have been written to: /code-examples/W1-intro/build
[ 50%] Building CXX object CMakeFiles/hello-world.dir/src/hello-world.cpp.o
[100%] Linking CXX executable hello-world
[100%] Built target hello-world
root@cec045503ba5:/code-examples/W1-intro# ls -la build/ | grep hello-w
-rwxr-xr-x 1 root   root   31880 Sep  8 03:31 hello-world
root@cec045503ba5:/code-examples/W1-intro# cmake -B build -DCMAKE_BUILD_TYPE=Release && cmake --build build
-- Configuring done (0.0s)
-- Generating done (0.0s)
-- Build files have been written to: /code-examples/W1-intro/build
[ 50%] Building CXX object CMakeFiles/hello-world.dir/src/hello-world.cpp.o
[100%] Linking CXX executable hello-world
[100%] Built target hello-world
root@cec045503ba5:/code-examples/W1-intro# ls -la build/ | grep hello-w
-rwxr-xr-x 1 root   root   16384 Sep  8 03:31 hello-world
root@cec045503ba5:/code-examples/W1-intro# 
```

Hello, World!

#include <iostream>: bring in the headers for streaming to console

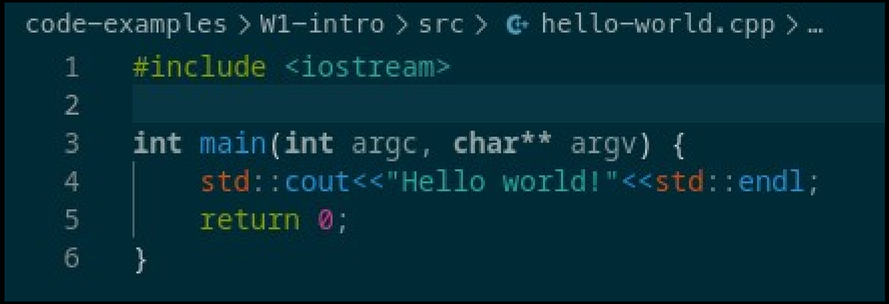
Every C++ binary uses a function named **main** as its entrypoint. This main takes an arg count and list of args

std: standard namespace

cout: Global stream buffer object to stdout

<< operator: output to the buffer

std::endl: insert newline to output sequence



```cpp
#include <iostream>

int main(int argc, char** argv) {
    std::cout<<"Hello world!"<<std::endl;
    return 0;
}
```

Additional Resources

- https://cmake.org/cmake/help/latest/guide/tutorial/Packaging%20Debug%20and%20Release.html