How to start a modern C++ project?

Mikhail Svetkin

Meeting C++, 2023

Agenda

- Motivation
- Let’s make a C++ project
- Build tools
- Project layout
- Tooling
- Dependency management
- Continue integrations
- IDEs

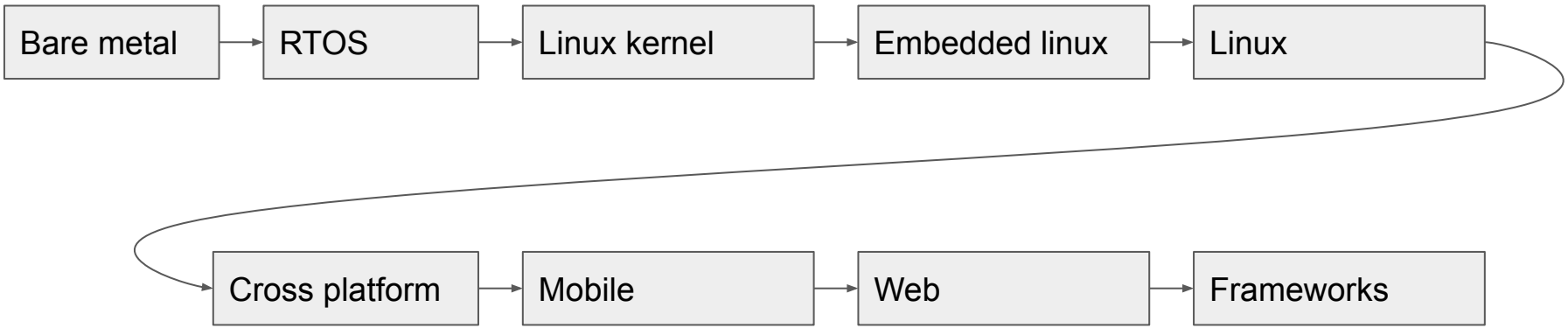
About me

Principal Software Engineer at reMarkable

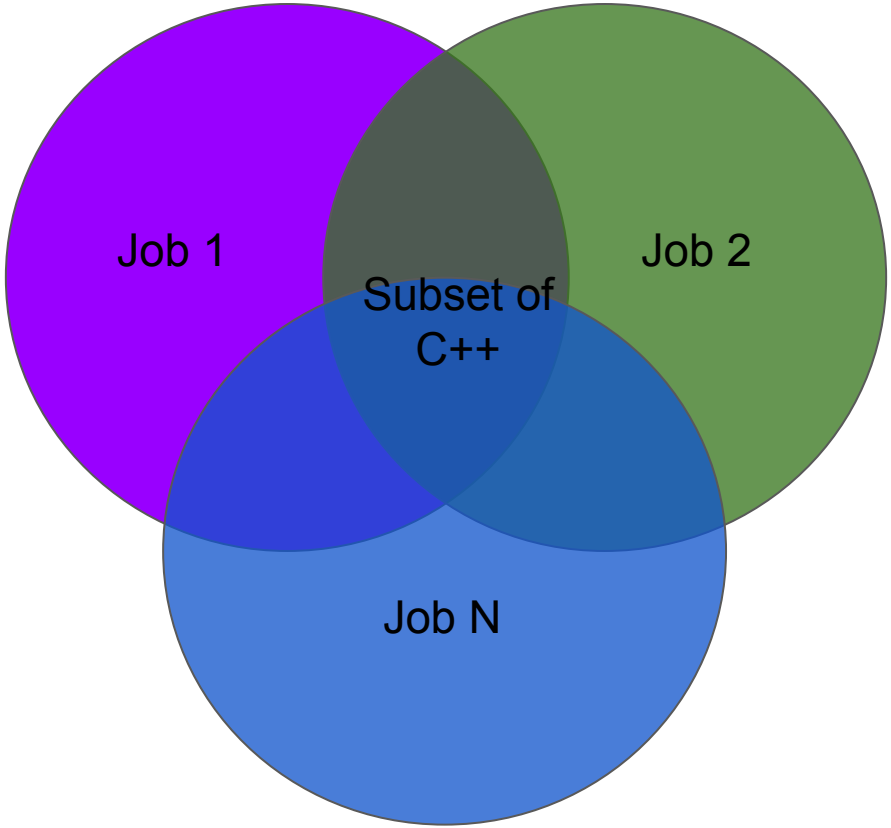
C++ programmer for last 12 years

Areas: architecture, networking, frameworks, libraries, build systems

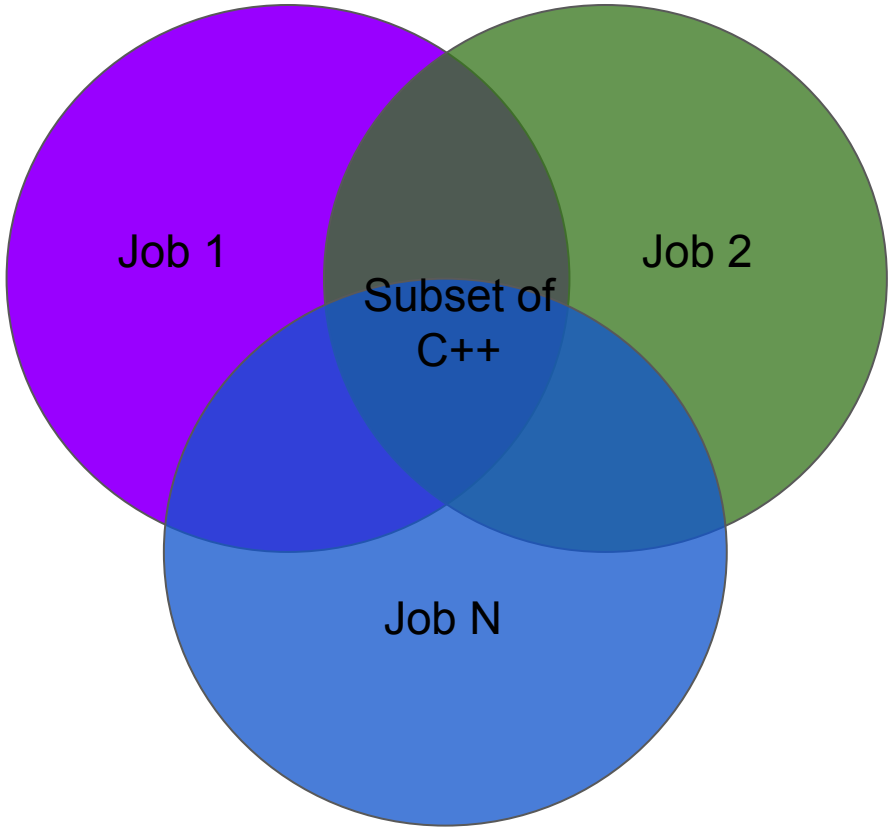
My journey


Bare metal
RTOS
Linux kernel
Embedded linux
Linux
Cross platform
Mobile
Web
Frameworks

What was in common?



What was different?



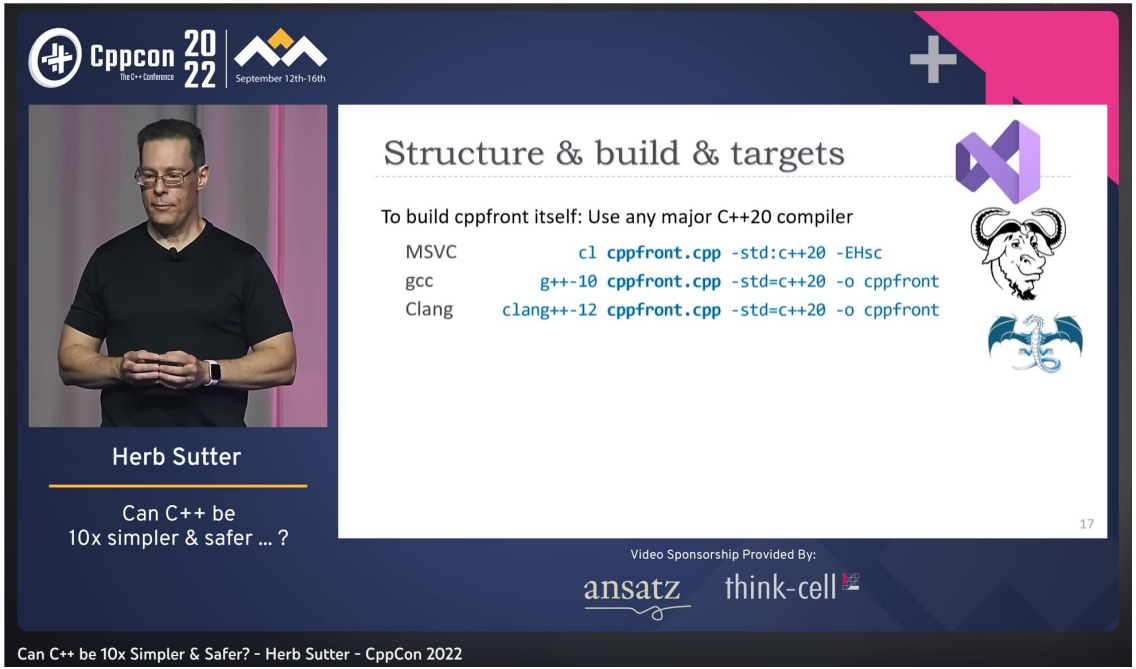
- Host OS
- Build tool
- C++ standard
- Subset of C++ (templates/virtual, exceptions/return-code, RTTI, …)
- Libs (Boost, Qt, …)
- Project layout
- CI/CD
- Dependency management
- Tools

What did I learn?

- Everybody use the C++, but differently
- Everybody builds C++, but differently
- Everybody uses some tooling, but with different options
- Some best practises are the same

Motivation

- Rust
- Go
- cppfront



Let’s create a modern C++ project

C++ project

```cpp
#include <iostream>

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

- g++ main.cpp && ./a.out

```
Hello, world!
```

- clang++ main.cpp && ./a.out

```
Hello, world!
```

- cl.exe main.cpp && main.exe

```
fatal error C1034: iostream: no include path
```

- **vcvars64.bat** && cl.exe main.cpp && main.exe

```
Hello, world!
```

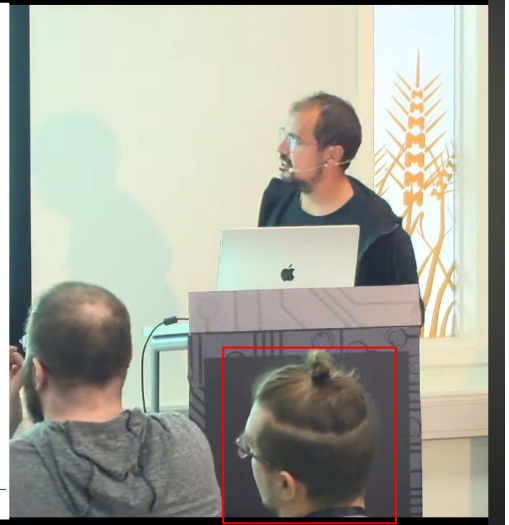
- clang-cl.exe main.cpp && main.exe

```
Hello, world!
```

Is it a modern C++ project?

Modern C++ project

```
#include <print>

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



NDC { TechTown }

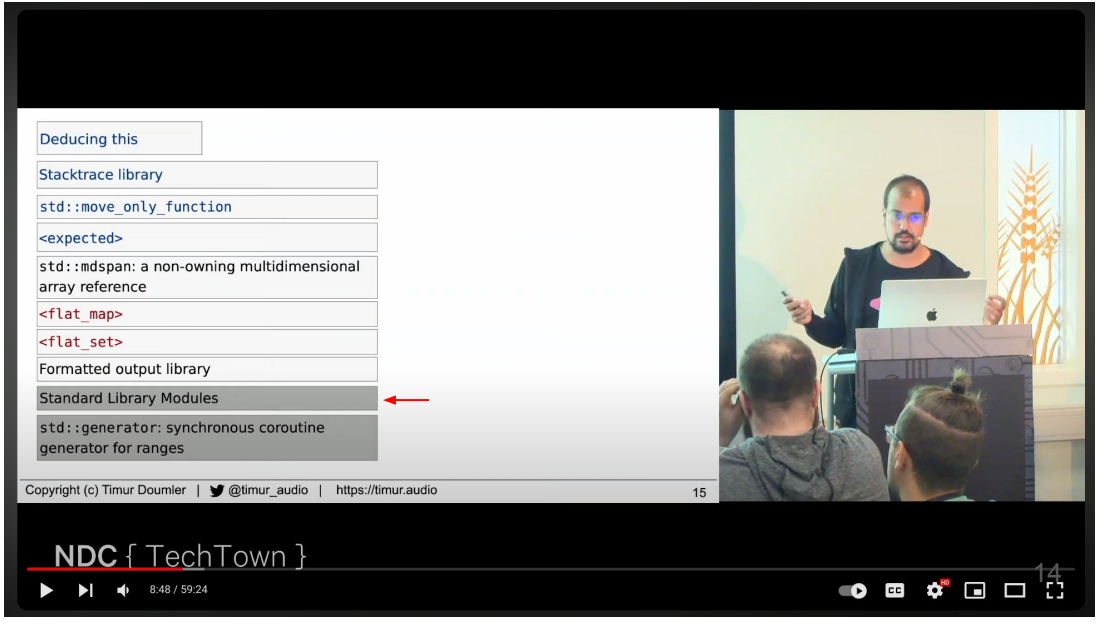
How C++23 changes the way we write code - Timur Doumler - NDC TechTown 2022

Modern C++ project

```cpp
#include <print>

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

Modern C++ project

```cpp
#include <print>

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



Modern C++ project

```
import std;

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

Modern C++ project

```
import std;

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

- g++ main.cpp

```
error: unknown type name 'import'
```

- g++ **-std=c++23** main.cpp

```
fatal error: module 'std' not found
```

- clang++ **-std=c++2b** main.cpp

```
fatal error: module 'std' not found
```

- cl.exe **/std:c++latest** main.cpp

```
error C2230: could not find module 'std'
```

- cl.exe /std:c++latest "%VCToolsInstallDir%\modules\std.ixx"
- cl.exe /std:c++latest main.cpp std.obj && main.exe

```
Hello, World!
```

Maybe it was too modern

Modern C++ project

```
import std;

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

Modern C++ project

```cpp
#include <print>

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

- g++ -std=c++23 main.cpp && ./a.out

```
fatal error: print: No such file or directory
```

- clang++ -std=c++23 main.cpp && ./a.out

```
fatal error: print: No such file or directory
```

- cl.exe /std:c++latest main.cpp

```
Hello, World!
```

Modern C++ project

```cpp
#include <fmt/core.h>

int main() {
  fmt::print("Hello, World!\n");
}
```

- g++ -std=c++23 **...** main.cpp && ./a.out

```
Hello, World!
```

- clang++ -std=c++23 **...** main.cpp && ./a.out

```
Hello, World!
```

- cl.exe /std:c++latest **...** main.cpp

```
Hello, World!
```

- **But** I still want **`<print>`** if it is available

Modern C++ project

```cpp
#include <version>

#if defined(__cpp_lib_print)
#  include <print>
#else
#  include <fmt/core.h>
#endif

int main() {
#if defined(__cpp_lib_print)
  std::print("Hello, from std world!\n");
#else
  fmt::print("Hello, from fmt world\n");
#endif
}
```

Modern C++ project

```cpp
#include <version>

#if defined(__cpp_lib_print)
#  include <print>
#else
#  include <fmt/core.h>
#endif

int main() {
#if defined(__cpp_lib_print)
  std::print("Hello, from std world!\n");
#else
  fmt::print("Hello, from fmt world\n");
#endif
}
```

- g++ -std=c++23 **...** main.cpp && ./a.out

```
Hello, from fmt world!
```

- clang++ -std=c++23 **...** main.cpp && ./a.out

```
Hello, from fmt world!
```

Modern C++ project

```cpp
#include <version>

#if defined(__cpp_lib_print)
#  include <print>
#else
#  include <fmt/core.h>
#endif

int main() {
#if defined(__cpp_lib_print)
  std::print("Hello, from std world!\n");
#else
  fmt::print("Hello, from fmt world\n");
#endif
}
```

- g++ -std=c++23 **...** main.cpp && ./a.out

```
Hello, from fmt world!
```

- clang++ -std=c++23 **...** main.cpp && ./a.out

```
Hello, from fmt world!
```

- cl.exe /std:c++latest main.cpp

```
Hello, from std world!
```

What did we learn?

- It might be tricky to start a modern C++ project
- Compilers are not the same (flags, standard library, …)
- Building is tricky
- Code could be become messy very quickly

What did we learn?

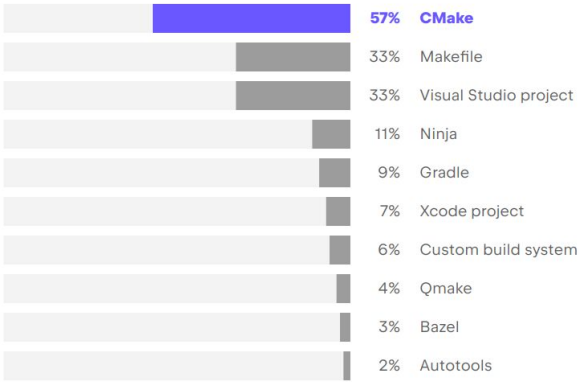
- It might be tricky to start a modern C++ project
- **Compilers are not the same (flags, standard library, ...)**
- **Building is tricky**
- **Code could be become messy very quickly**

We have a lot of tools in C++

CMake - has become the de facto standard

Which project models or build systems do you regularly use?

Overall | Embedded | Games



I am on record as likening CMake to Stockholm syndrome for C++ engineers. It has become the de facto standard, for better or worse, as demonstrated by the clear lead it holds over its competitors.

Guy Davidson
Head of Engineering Practice, Creative Assembly

2022 Annual C++ Developer Survey "Lite"

ANSWER CHOICES	RESPONSES	
CMake	80.87%	951
Ninja	41.92%	493
MSBuild	37.41%	440
Make/nmake	37.07%	436
distcc/ccache	15.65%	184
Xcode projects	10.20%	120
Other (please specify)	10.03%	118
QMake	9.27%	109
Autotools	8.93%	105
Gradle	6.97%	82
IncrediBuild	6.04%	71
Boost Build (bjam)	5.27%	62
Meson	5.27%	62

CMake - has become the de facto standard

2022 Annual C++ Developer Survey "Lite"

	MAJOR PAIN POINT	MINOR PAIN POINT	NOT A SIGNIFICANT ISSUE FOR ME	TOTAL	WEIGHTED AVERAGE
Managing libraries my application depends on	47.63% 563	34.77% 411	17.60% 208	1,182	2.30
Build times	43.94% 515	38.65% 453	17.41% 204	1,172	2.27
Setting up a continuous integration pipeline from scratch (automated builds, tests, ...)	33.73% 394	40.75% 476	25.51% 298	1,168	2.08
Setting up a development environment from scratch (compiler, build system, IDE, ...)	27.83% 329	42.98% 508	29.19% 345	1,182	1.99
Concurrency safety: Races, deadlocks, performance bottlenecks	25.04% 293	46.67% 546	28.29% 331	1,170	1.97
Managing CMake projects	29.34% 343	38.15% 446	32.51% 380	1,169	1.97

CMake - quick intro

- Meta-build system (generating platform-specific build)
- Cross platform
- Configurable-ish
- Scalable-ish

CMake - quick intro

- main.cpp
- **CMakeLists.txt**

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp LANGUAGES CXX)

add_executable(app)

target_sources(app
 PRIVATE
  main.cpp
)

set_target_properties(app
 PROPERTIES
   CXX_STANDARD 20
   CXX_STANDARD_REQUIRED ON
)
```

CMake - quick intro

- main.cpp
- CMakeLists.txt

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp LANGUAGES CXX)

add_executable(app)

target_sources(app
 PRIVATE
  main.cpp
)

set_target_properties(app
 PROPERTIES
   CXX_STANDARD 20
   CXX_STANDARD_REQUIRED ON
)
```

CMake - quick intro

- main.cpp
- CMakeLists.txt

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp LANGUAGES CXX)

add_executable(app)

target_sources(app
 PRIVATE
  main.cpp
)

set_target_properties(app
 PROPERTIES
   CXX_STANDARD 20
   CXX_STANDARD_REQUIRED ON
)
```

CMake - quick intro

- main.cpp
- CMakeLists.txt

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp LANGUAGES CXX)

add_executable(app)

target_sources(app
 PRIVATE
  main.cpp
)

set_target_properties(app
 PROPERTIES
   CXX_STANDARD 20
   CXX_STANDARD_REQUIRED ON
)
```

CMake - quick intro

- main.cpp
- CMakeLists.txt

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp LANGUAGES CXX)

add_executable(app)

target_sources(app
 PRIVATE
  main.cpp
)

set_target_properties(app
 PROPERTIES
   CXX_STANDARD 20
   CXX_STANDARD_REQUIRED ON
)
```

CMake - quick intro

- cmake -B build .
- cmake --build build
- ./build/app

```
Hello, World!
```

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp LANGUAGES CXX)

add_executable(app)

target_sources(app
 PRIVATE
  main.cpp
)

set_target_properties(app
 PROPERTIES
   CXX_STANDARD 20
   CXX_STANDARD_REQUIRED ON
)
```

CMake - presets

- CMakeLists.txt
- **CMakePresets.json**
- **CMakeUserPresets.json**
- main.cpp

```
{
 "include": [...],
 "configurePresets": [...],
 "buildPresets": [...],
 "testPresets": [...],
 "packagePresets": [...],
 "workflowPresets": [...],
}
```

CMake - configure presets

- cmake --preset <preset name>
- cmake --preset **base**

```
"configurePresets": [
 {
   "name": "base",
   "inherits": [...],
   "generator": "Ninja Multi-Config",
   "binaryDir": "${sourceDir}/build/${presetName}",
   "toolchainFile": "...",
   "cacheVariables": [...],
   "environment": [...],
   "condition": {...}
   ...
 },
]
```

CMake - build presets

- cmake --build --preset <preset name> --config <Config>
- cmake --build --preset **base**
- cmake --build --preset **base** --config **Debug**
- cmake --build --preset **base** --config **Release**

```
"buildPresets": [
 {
   "name": "base",
   "inherits": [...],
   "configurePresets": "base",
   "targets": [...],
   "environment": [...],
   "condition": {...}
   ...
 },
]
```

CMake - test presets

- ctest --preset <preset name> -C <Config>
- ctest --preset **base**
- ctest --preset **base** -C **Debug**
- ctest --preset **base** -C **Release**

```
"testPresets": [
 {
   "name": "base",
   "inherits": [...],
   "configurePresets": "base",
   "execution": {...},
   "environment": [...],
   "condition": {...}
   ...
 },
]
```

CMake - package presets

- cpack --preset <preset name> --config <Config>
- cpack --preset **base**
- cpack --preset **base** --config **Debug**
- cpack --preset **base** --config **Release**

```
"packagePresets": [
 {
   "name": "base",
   "inherits": [...],
   "configurePresets": "base",
   "generators": ["TGZ"],
   "configurations": [...],
   "variables": [...],
   ...
 },
]
```

CMake - workflow presets

- cmake --workflow <preset name>
- cmake --workflow **base**

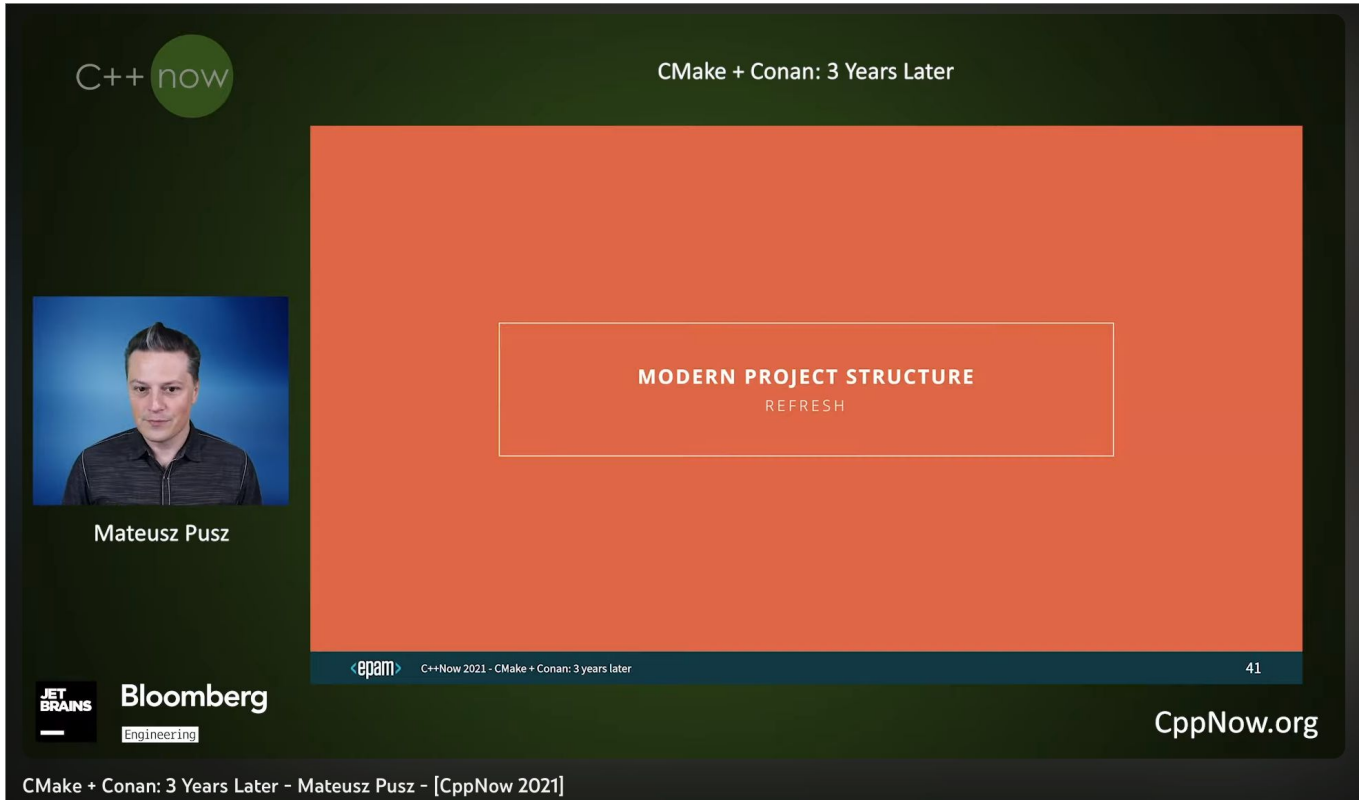
```
"workflowPresets": [
 {
   "name": "base",
   "steps": [
     { "type": "configure", "name": "base" },
     { "type": "build", "name": "base" },
     { "type": "test", "name": "base" }
     { "type": "package", "name": "base" }
   ],
 },
]
```

CMake - presets names

- cmake --preset <arch>-<os>-<compiler>-<linkage>
- cmake --preset x64-linux-gcc-static
- cmake --preset x64-linux-gcc-dynamic
- cmake --preset x86-windows-msvc-static

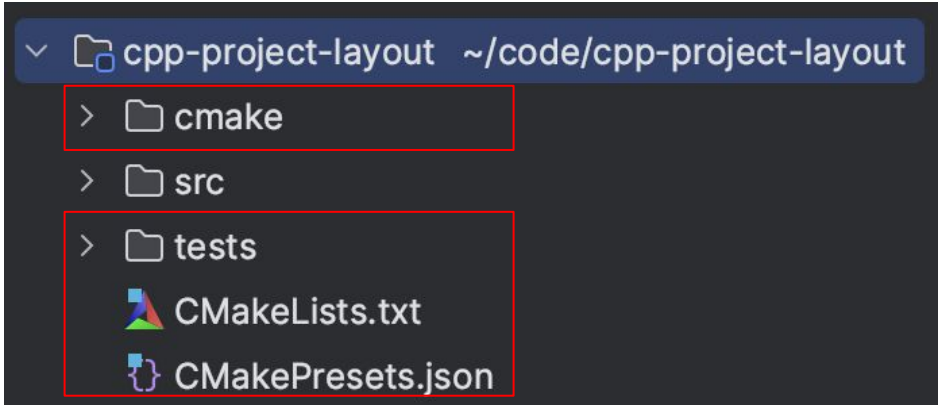
How to structure C++ project?

CMake - modern project layout



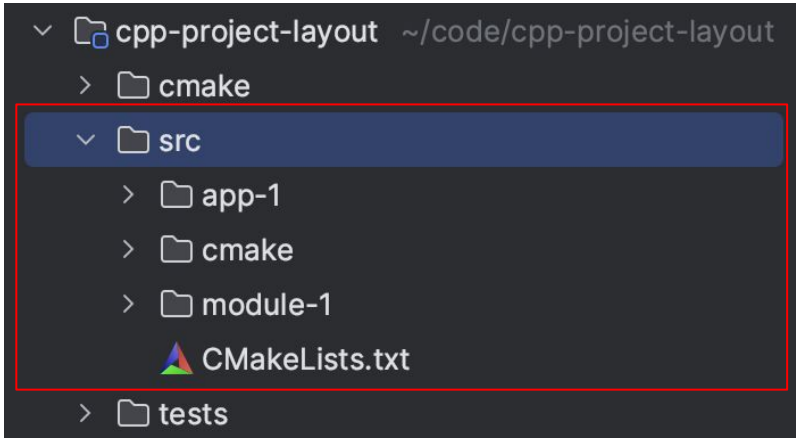
CMake - modern project layout

- Simple project wrapper
- Entry point for development
- Enables all development features
  - dependency management
  - warnings
  - tests
  - docs
  - cache
  - …



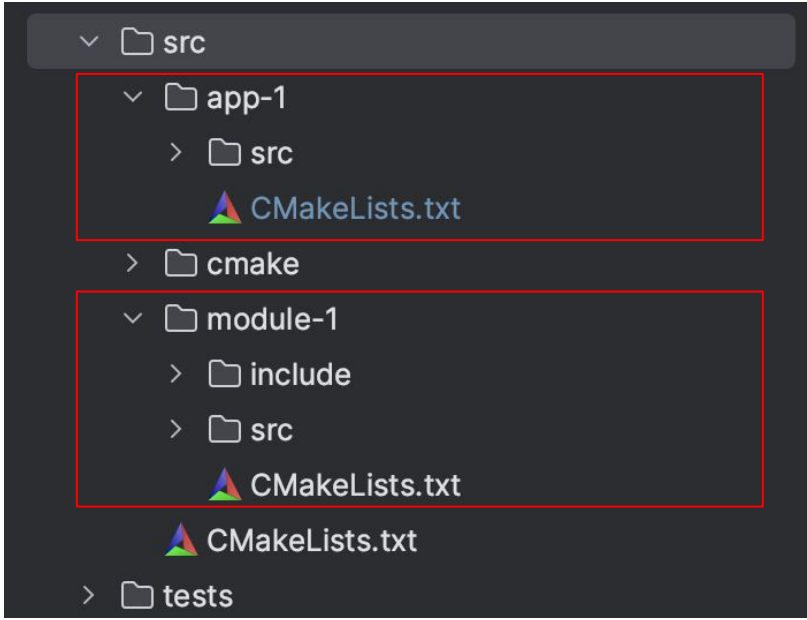
CMake - modern project layout

- Standalone CMake project file
- Entry point for consumers
  - add_subdirectory
  - package managers
- Does not affect development environment



CMake - modern project layout

- Implementation of the project modules, apps
- Easy to enable / disable



Let’s add CMake to our C++ project

Modern C++ project layout

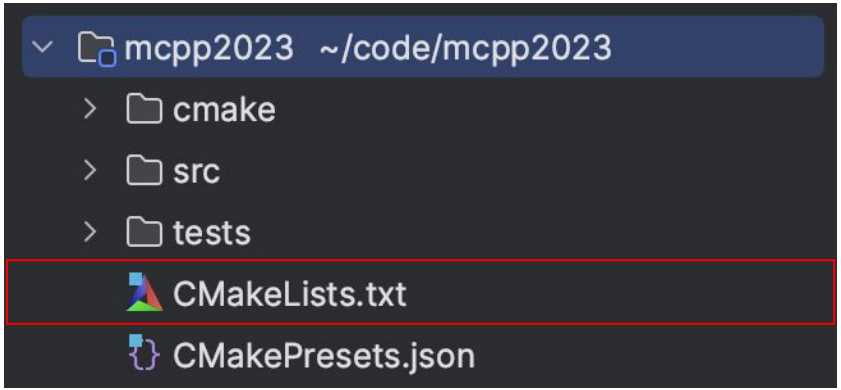
```cpp
#include <version>

#if defined(__cpp_lib_print)
#  include <print>
#else
#  include <fmt/core.h>
#endif

int main() {
#if defined(__cpp_lib_print)
  std::print("Hello, from std world!\n");
#else
  fmt::print("Hello, from fmt world\n");
#endif
}
```

Modern C++ project layout

```cpp
#include "mcpp/log/debug.hpp"

int main() {
 mcpp::log::debug("Hello, world!\n");
}
```

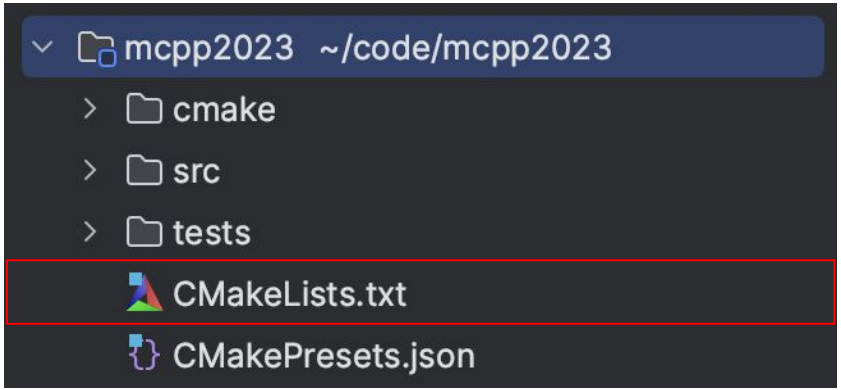
- g++/clang++: `mcpp::print` → `fmt::print`
- cl.exe: `mcpp::print` → `std::print`

Modern C++ project layout



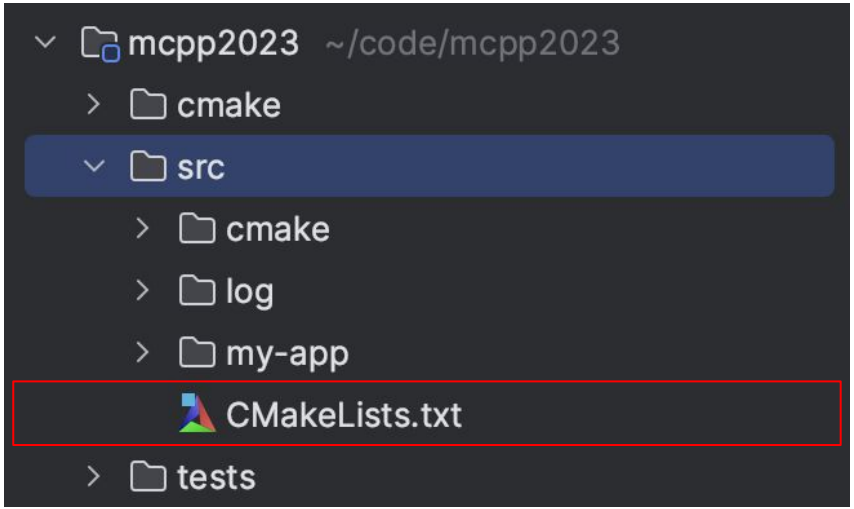

```cmake
# CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp-dev LANGUAGES CXX)

list(APPEND
  CMAKE_MODULE_PATH "${PROJECT_SOURCE_DIR}/cmake")
# enable cache, extra warnings,
# docs generation and etc

add_subdirectory(src)

enable_testing()
add_subdirectory(tests)
```

Modern C++ project layout



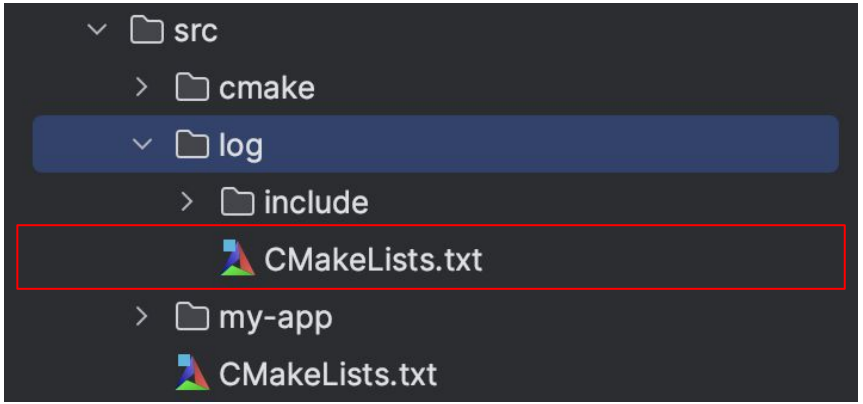

```cmake
# CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp-dev LANGUAGES CXX)

list(APPEND
  CMAKE_MODULE_PATH "${PROJECT_SOURCE_DIR}/cmake")
# enable cache, extra warnings,
# docs generation and etc

add_subdirectory(src)

enable_testing()
add_subdirectory(tests)
```

Modern C++ project layout



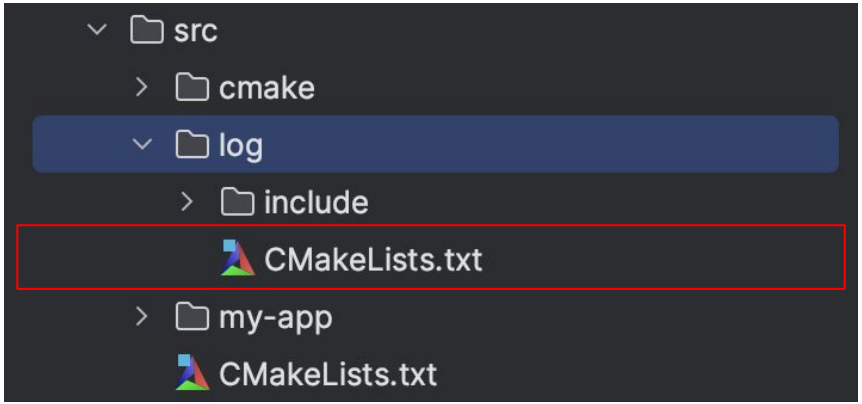

```cmake
# src/CMakeLists.txt
cmake_minimum_required(VERSION 3.25)
project(mcpp VERSION 0.0.1 LANGUAGES CXX)

list(APPEND
 CMAKE_MODULE_PATH "${PROJECT_SOURCE_DIR}/cmake")

include(add_mcpp_module)

add_subdirectory(log)
add_subdirectory(my-app)
```

Modern C++ project layout



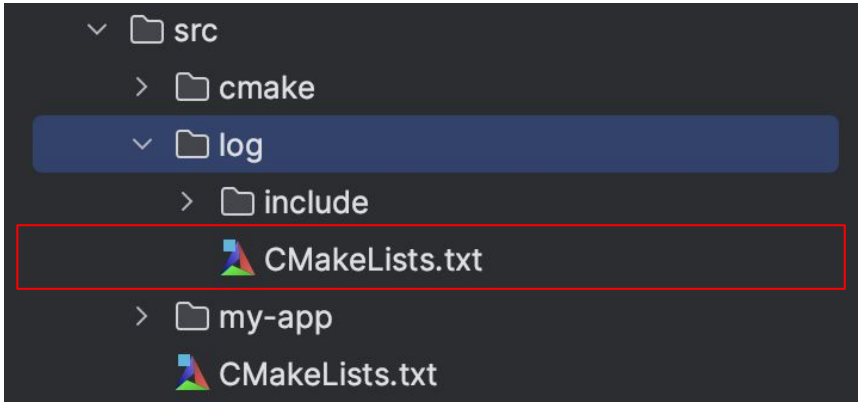

```
# src/log/CMakeLists.txt
add_mcpp_module(log INTERFACE)

target_sources(${mcpp_module_target}
  PRIVATE include/mcpp/log/debug.hpp)

if(NOT MSVC)
   find_package(fmt CONFIG REQUIRED)
   target_link_libraries(${mcpp_module_target}
INTERFACE fmt::fmt)
   target_compile_definitions(${mcpp_module_target}
INTERFACE mcpp_USE_FMT)
endif()
```

Modern C++ project layout



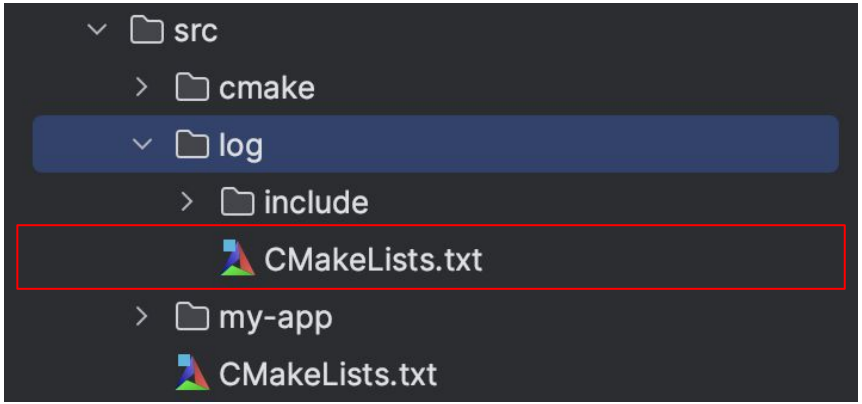

```
# src/log/CMakeLists.txt
add_mcpp_module(log INTERFACE)

target_sources(${mcpp_module_target}
  PRIVATE include/mcpp/log/debug.hpp)

if(NOT MSVC)
   find_package(fmt CONFIG REQUIRED)
   target_link_libraries(${mcpp_module_target}
INTERFACE fmt::fmt)
   target_compile_definitions(${mcpp_module_target}
INTERFACE mcpp_USE_FMT)
endif()
```

Modern C++ project layout



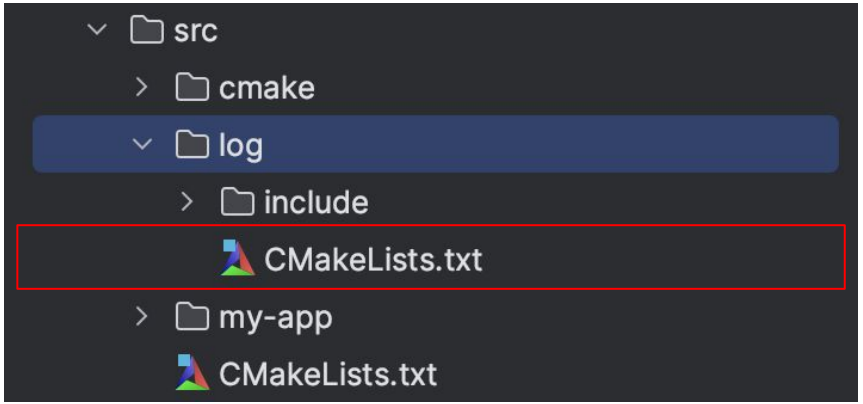

```
# src/log/CMakeLists.txt
add_mcpp_module(log INTERFACE)

target_sources(${mcpp_module_target}
  PRIVATE include/mcpp/log/debug.hpp)

if(NOT MSVC)
   find_package(fmt CONFIG REQUIRED)
   target_link_libraries(${mcpp_module_target}
INTERFACE fmt::fmt)
   target_compile_definitions(${mcpp_module_target}
INTERFACE mcpp_USE_FMT)
endif()
```

Modern C++ project layout



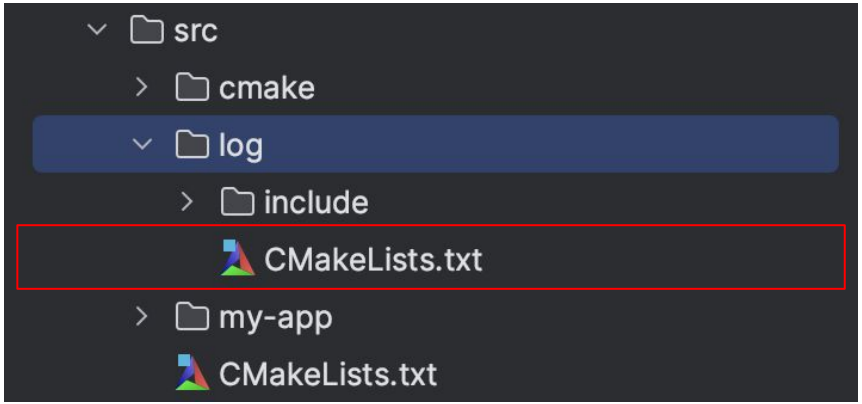

```
# src/log/CMakeLists.txt
add_mcpp_module(log INTERFACE)

target_sources(${mcpp_module_target}
  PRIVATE include/mcpp/log/debug.hpp)

if(NOT MSVC)
   find_package(fmt CONFIG REQUIRED)
   target_link_libraries(${mcpp_module_target}
INTERFACE fmt::fmt)
   target_compile_definitions(${mcpp_module_target}
INTERFACE mcpp_USE_FMT)
endif()
```

Modern C++ project layout



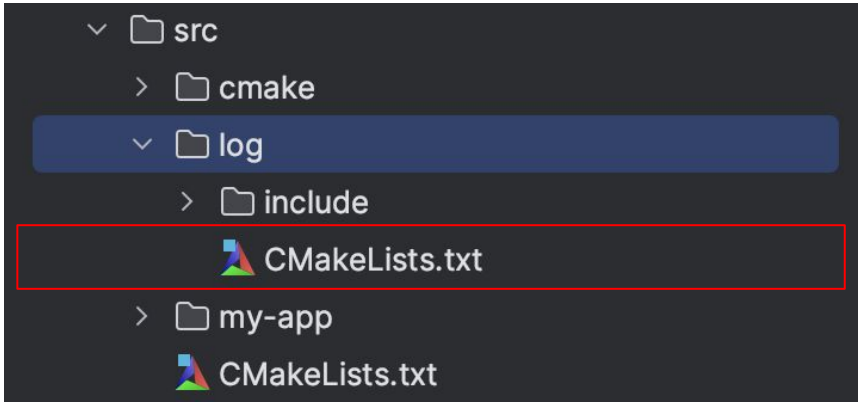

```
# src/log/CMakeLists.txt
add_mcpp_module(log INTERFACE)

target_sources(${mcpp_module_target}
  PRIVATE include/mcpp/log/debug.hpp)

if(NOT MSVC)
   find_package(fmt CONFIG REQUIRED)
   target_link_libraries(${mcpp_module_target}
INTERFACE fmt::fmt)
   target_compile_definitions(${mcpp_module_target}
INTERFACE mcpp_USE_FMT)
endif()
```

Modern C++ project layout



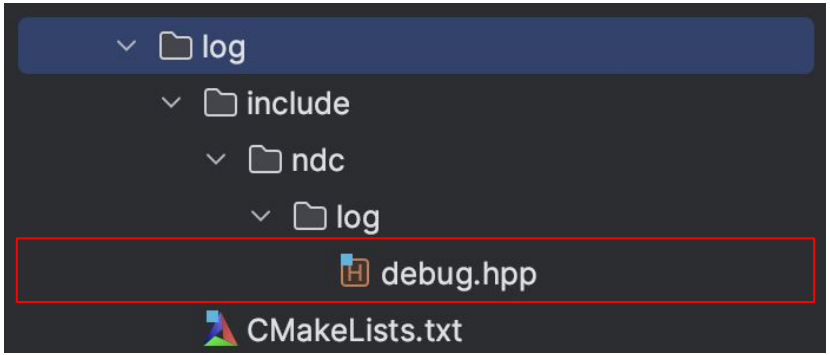

```
# src/log/CMakeLists.txt
add_mcpp_module(log INTERFACE)

target_sources(${mcpp_module_target}
  PRIVATE include/mcpp/log/debug.hpp)

if(NOT MSVC)
   find_package(fmt CONFIG REQUIRED)
   target_link_libraries(${mcpp_module_target}
INTERFACE fmt::fmt)
   target_compile_definitions(${mcpp_module_target}
INTERFACE mcpp_USE_FMT)
endif()
```

Modern C++ project layout



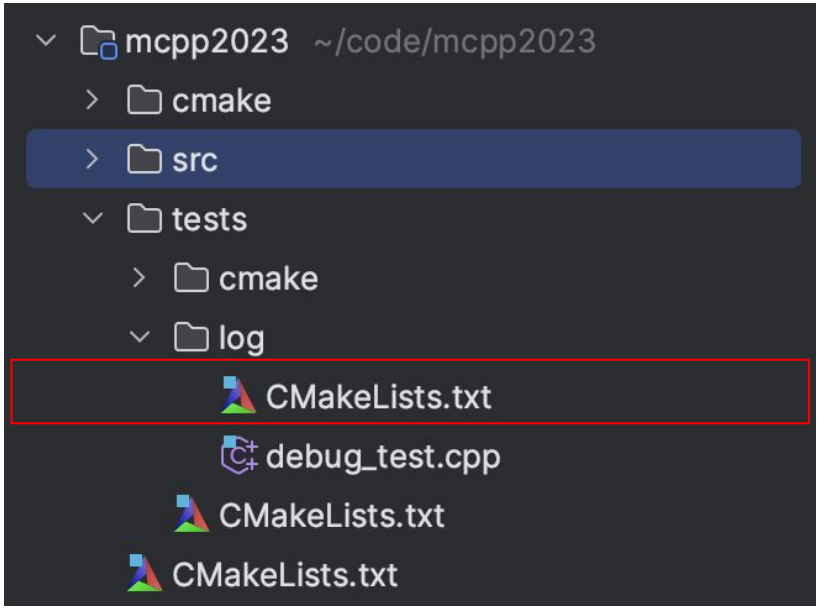

```
# src/log/CMakeLists.txt
add_mcpp_module(log INTERFACE)

target_sources(${mcpp_module_target}
  PRIVATE include/mcpp/log/debug.hpp)

if(NOT MSVC)
   find_package(fmt CONFIG REQUIRED)
   target_link_libraries(${mcpp_module_target}
INTERFACE fmt::fmt)
   target_compile_definitions(${mcpp_module_target}
INTERFACE mcpp_USE_FMT)
endif()
```

Modern C++ project layout



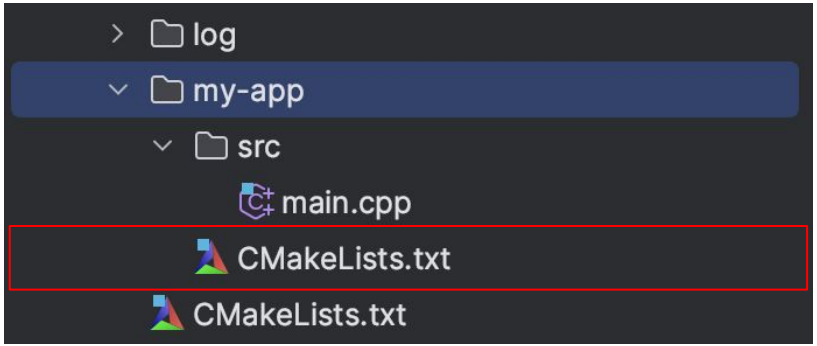

```cpp
// src/log/include/mcpp/log/debug.hpp
#ifdef mcpp_USE_FMT
# include <fmt/core.h>
#else
# include <print >
#endif

namespace mcpp::log {
...
} // namespace mcpp::log
```

Modern C++ project layout




```cpp
// test/log/debug_test.cpp
#include "mcpp/log/debug.hpp"

#include <catch2/catch_test_macros.hpp>

TEST_CASE("header sanity check") {
}
```

Modern C++ project layout




```
// src/my-app/src/CMakeLists.txt
add_mcpp_executable(my-app)

target_sources(${mcpp_executable_target}
 PRIVATE src/main.cpp)

target_link_libraries(${mcpp_executable_target}
 PRIVATE mcpp::log)
```

Modern C++ project layout

- cmake --workflow --preset x64-linux-gcc-dynamic
- ./build/x64-linux-gcc-dynamic/src/my-app

```
Hello, World!
```

- cmake --workflow --preset x64-linux-clang-dynamic
- ./build/x64-linux-clang-dynamic/src/my-app

```
Hello, World!
```

Version control

Why do you need version control?

- History tracking
- Collaboration
- Backup and recovery
- Choose what suits you the most ~~(git/github)~~

Git tips and tricks

- .git-blame-ignore-revs-file
- .gitignore

Let’s share with a friend

- git clone git@github.com:msvetkin/mcpp2023.git
- cd mcpp2023
- cmake --workflow --preset x64-linux-gcc-dynamic

```
-- The CXX compiler identification is GNU
13.2.1
-- Detecting CXX compiler ABI info
...
Could not find a package configuration
file provided by "fmt" with any of
  the following names:
    fmtConfig.cmake
    fmt-config.cmake
Call Stack (most recent call first):
  src/log/CMakeLists.txt:9 (find_package)
```

Package Managers

Why do you need package manager?

For other distros, get the separate components below.

Build essentials

Ubuntu and/or Debian:	`sudo apt-get install build-essential perl python3 git`
Fedora 30	`su - -c "dnf install perl-version git gcc-c++ compat-openssl10-devel harfbuzz-devel double-conversion-devel libzstd-devel at-spi2-atk-devel dbus-devel mesa-libGL-devel"`
OpenSUSE:	`sudo zypper install git-core gcc-c++ make`

Libxcb

Libxcb is now the default window-system backend for platforms based on X11/Xorg, and you should therefore install libxcb and its accompanying packages. Qt5 should build with whatever libxcb version is available in your distro's packages (but you may optionally wish to use v1.8 or higher to have threaded rendering support). The README lists the required packages.

Ubuntu/Debian:	`sudo apt-get install '^libxcb.*-dev' libx11-xcb-dev libglu1-mesa-dev libxrender-dev libxi-dev libxkbcommon-dev libxkbcommon-x11-dev`
Fedora 30:	`su - -c "dnf install libxcb libxcb-devel xcb-util xcb-util-devel xcb-util-*-devel libX11-devel libXrender-devel libxkbcommon-devel libxkbcommon-x11-devel libXi-devel libdrm-devel libXcursor-devel libXcomposite-devel"`
OpenSUSE 12+:	`sudo zypper in xorg-x11-libxcb-devel xcb-util-devel xcb-util-image-devel xcb-util-keysyms-devel xcb-util-renderutil-devel xcb-util-wm-devel xorg-x11-devel libxkbcommon-x11-devel libxkbcommon-devel libXi-devel`
ArchLinux/Manjaro:	`sudo pacman -S --needed libxcb xcb-proto xcb-util xcb-util-image xcb-util-wm libxi`
Chakra Linux:	Install the ArchLinux packages, plus xcb-util-keysyms. It's available from CCR.
Mandriva/ROSA/Unity:	`urpmi 'pkgconfig(xcb)' 'pkgconfig(xcb-icccm)' 'pkgconfig(xcb-image)' 'pkgconfig(xcb-renderutil)' 'pkgconfig(xcb-keysyms)' 'pkgconfig(xrender)'`
Linux Mint:	`apt-get install libx11-xcb-dev libxcb-composite0-dev libxcb-cursor-dev libxcb-damage0-dev libxcb-dpms0-dev libxcb-dri2-0-dev libxcb-dri3-dev libxcb-glx0-dev libxcb-icccm4-dev libxcb-image0-dev libxcb-keysyms1-dev libxcb-present-dev libxcb-randr0-dev libxcb-render-util0-dev libxcb-render0-dev libxcb-shape0-dev libxcb-shm0-dev libxcb-sync-dev libxcb-util-dev libxcb-xfixes0-dev libxcb-xinerama0-dev libxcb-xkb-dev libxcb-xtest0-dev libxcb1-dev`

Centos 5/6	Install missing Qt build dependencies:
	`yum install libxcb libxcb-devel xcb-util xcb-util-devel`
	Install Red Hat DevTools 1.1 for CentOS-5/6 x86_64, they are required due to outdated GCC shipped with default CentOS:
	`wget http://people.centos.org/tru/devtools-1.1/devtools-1.1.repo -O /etc/yum.repos.d/devtools-1.1.repo`
	`yum install devtoolset-1.1`
	Initialise your newly installed dev tools:
	`scl enable devtoolset-1.1 bash`
	`# Test - Expect to see gcc version 4.7.2 (not gcc version 4.4.7)`
	`gcc -v`
	For more info on preparing the environment on **CentOS**, see this thread.

Centos 7	Update to gcc 7:
	`yum install centos-release-scl`
	`yum install devtoolset-7-gcc*`
	`scl enable devtoolset-7 bash`
	Install missing Qt build dependencies (Qt 5.13):
	`yum install libxcb libxcb-devel xcb-util xcb-util-devel mesa-libGL-devel libxkbcommon-devel`

https://wiki.qt.io/Building_Qt_5_from_Git

What are the options?

- conan
- vcpkg

Let’s add vcpkg to our C++ project

vcpkg - CMake integration

```
# CMakePresets.json
"configurePresets": [
 {
   "name": "base",
   ...
   "toolchainFile": "<vcpkg>/vcpkg.cmake",
   ...
 },
]
```

```
# vcpkg.json
{
 "name": "mcpp",
 "version-string": "0.0.1",
 "dependencies": [
   "fmt",
   "catch2"
 ]
}
```

Let’s share with a friend - part 2

me:

- git add cmake/presets/base.json vcpkg.json
- git commit -m “feat(cmake): add vcpkg”
- git push origin main

friend:

- git pull --rebase
- cmake --workflow --preset x64-linux-gcc-dynamic

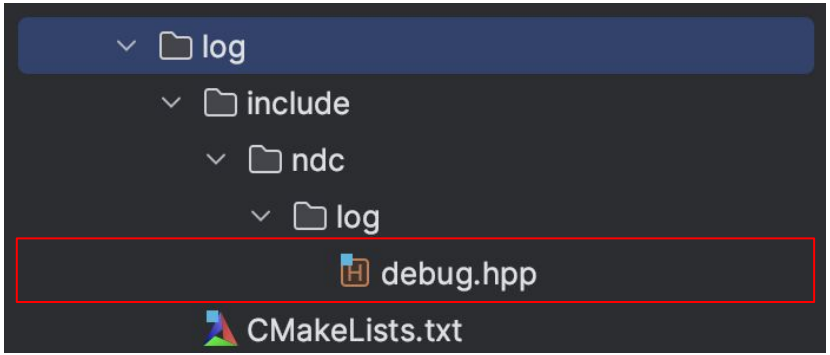
```
-- Running vcpkg install
The following packages will be built and
installed:
    fmt:x64-linux-dynamic → 10.0.0
    catch2:x64-linux-dynamic → 3.4.0
...
-- The CXX compiler identification is GNU
13.2.1
...
-- Configuring done (1.6s)
-- Generating done (0.0s)
-- Building done
-- Testing done
```

Let’s share with a friend - part 3

- git clone git@github.com:msvetkin/mcpp2023.git
- cd mcpp2023
- cmake --workflow --preset x64-windows-msvc-static

```
-- The CXX compiler identification is MSVC
...
-- Configuring done (1.6s)
-- Generating done (0.0s)
-- Building
fatal error: print : No such file or directory
```

Let’s share with a friend - part 3




```cpp
// src/log/include/mcpp/log/debug.hpp
#ifdef mcpp_USE_FMT
# include <fmt/core.h>
#else
# include <print>
#endif

namespace mcpp::log {
...
} // namespace mcpp::log
```

Continuous Integration

What to do with CI?

- Build at least on: windows, linux, mac
- Build at least with: gcc, clang (stdlibc++/libc++) msvc
- Build at least Debug and Release versions
- Run at least: cppcheck, clang-tidy

Let’s add github-actions to our C++ project

Github actions

```yaml
name: ci

on: [push, pull_request]

jobs:

 build:

   runs-on: ${{matrix.target.os}}

   strategy:

     fail-fast: false

     matrix:

       ...
     steps:

       ...
```

Github actions

```yaml
name: ci

on: [push, pull_request]

jobs:
 build:
   runs-on: ${{matrix.target.os}}
   strategy:
     fail-fast: false
     matrix:
       ...
     steps:
       ...
```

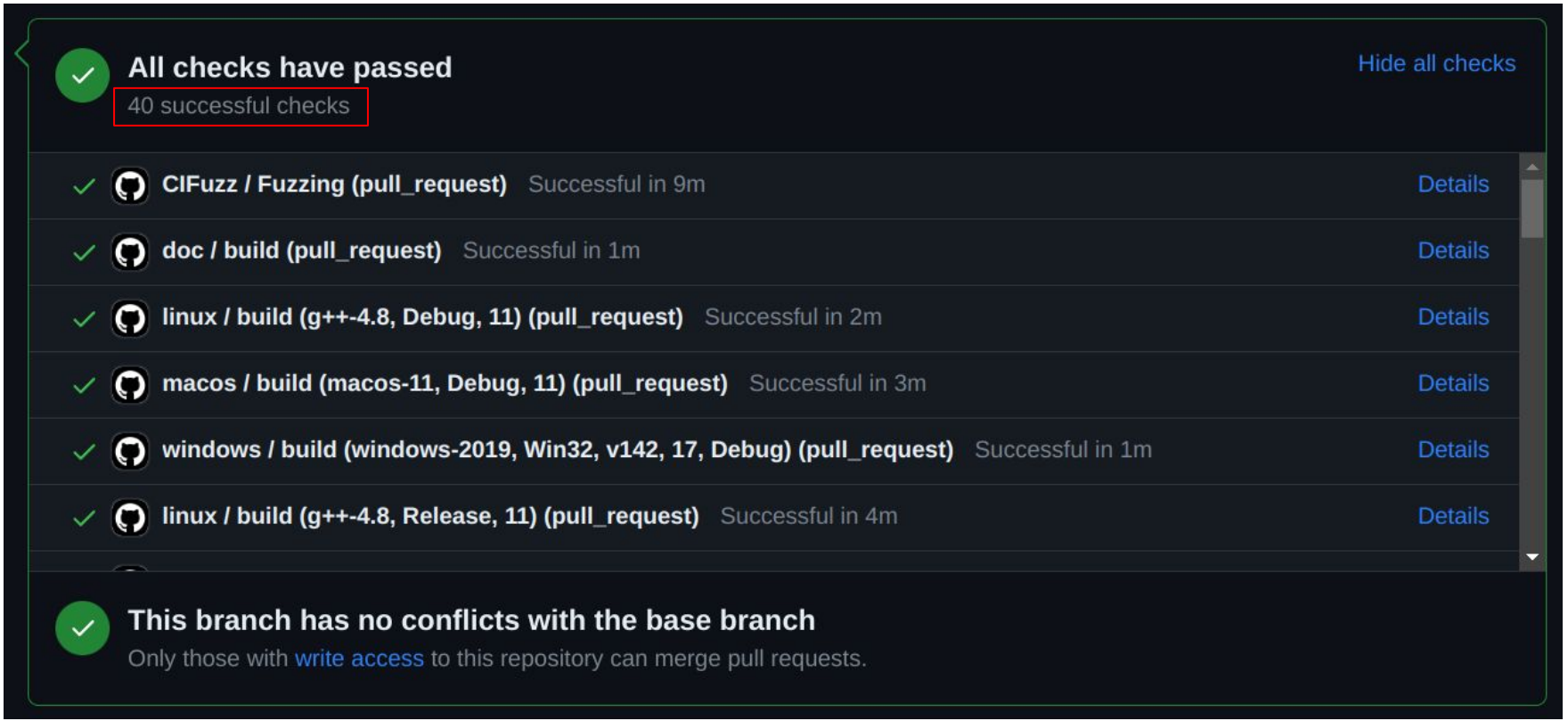
```yaml
matrix:
 target: [
   { os: ubuntu-latest, preset: x64-linux-gcc-dynamic },
   { os: ubuntu-latest, preset: x86-linux-gcc-dynamic },
   { os: windows-latest, preset: x64-windows-msvc-dynamic },
   { os: macos-latest, preset: x86-osx-gcc-dynamic },
 ]
```

Github actions

```yaml
name: ci

on: [push, pull_request]

jobs:

 build:

   runs-on: ${{matrix.target.os}}

   strategy:

     fail-fast: false

     matrix:

       ...
     steps:

       ...
```

```yaml
steps:

- uses: actions/checkout@v3


- name: Install system dependencies (compiler, cmake, ninja, ...)

...


- name: cmake

  shell: bash

  run: |

     cmake --workflow --preset ${{matrix.target.preset}}
```

Github actions - https://github.com/fmtlib/fmt/pull/3636



C++ Tooling

- Formaters:
  - clang-format
  - cmake-format
- Linters:
  - clang-tidy
  - clazy
  - cppcheck
  - include-what-you-use
  - commercial linters(pvs-studio, sonar, …)
- Pre-commit

What do if I just want to write code

Select a good IDE

- VIM / Emacs / VSCode
- CLion
- Visual Studio

Select a good IDE

- ~~VIM / Emacs / VSCode~~
- CLion
- Visual Studio

CLion - pros

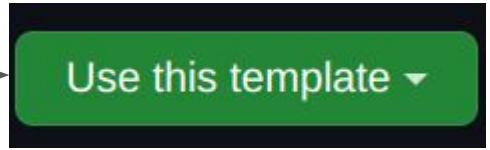
- The most complete code model
- Bundled / Integration with C++ tooling
- Actively developed and maintained
- Emacs like user experience ;-)

Let’s try to open our project in CLion

So how to start a modern C++ project?

cpp-project-template

- https://github.com/msvetkin/cpp-project-template
- git clone git@github.com:msvetkin/cpp-project-template.git
- cmake -P init.cmake --project <name> --module <name> --header <name>



Thank you

Links

- Developer Ecosystem C++ - JetBrains
- C++ Developer Survey "Lite" - isocpp
- CMake + Conan: 3 Years Later - Mateusz Pusz
- How C++23 changes the way we write code - Timur Doumler - mcpp TechTown 2022
- Standard C++ Toolset - Anastasia Kazakova - C++ on Sea 2023
- Dependency Management in C++ - Patricia Aas - mcpp TechTown 2021
- Dependency management in C++ - Xavier Bonaventura - code::dive 2019
- Package management in C++ - Mikhail Svetkin - mcpp TechTown 2022
- cppfront

Questions?