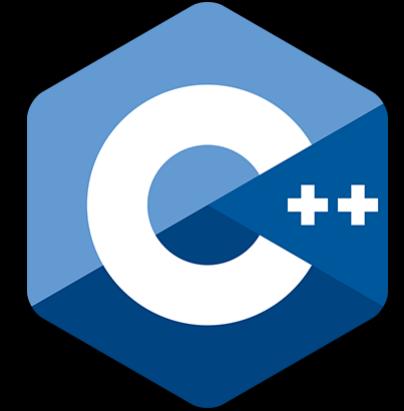


How To Address 7 Major C++ Pain Points with Tools

Anastasia Kazakova, JetBrains

NDC TechTown 2024

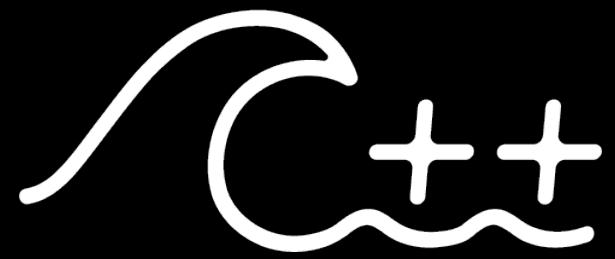
About me



C++: Embedded,
Telecom, 4G/LTE



The Dutch C++ Group



cppunderthesea.nl
11 October 2024
Breda, Netherlands



Anastasia Kazakova
Head of Marketing and BizDev



Topics to cover

-

1. Painpoints and painkillers
2. Tools DO help
3. C++ toolability is hard
4. Tools for C++ painpoints
5. Era of AI

Tell us a joke about C++ and its tooling!



Why do C++ developers make great philosophers?

Because after wrestling with the lack of tooling for hours,
they start questioning the meaning of life,
memory leaks,
and whether their compiler even exists!

Do we have a problem?

Language

Tooling

Other

Tools are on
top for years!

	Which of these do you find frustrating about C++ dev?	Major %
Language	Managing libraries my application depends on	45%
	Build times	43%
	Setting up a CI pipeline from scratch	30%
Tooling	Managing CMake projects	30%
	Concurrency safety: Races, deadlocks, performance bottlenecks	27%
	Setting up a dev env from scratch	26%
Other	Parallelism support	23%
	Managing Makefiles	20%
	Memory safety: Bounds safety issues	20%
Tools are on top for years!	Memory safety: Use-after-delete/free	20%
	Debugging issues in my code	18%
	Managing MSBuild projects	16%
Tools are on top for years!	Unicode, internationalization, and localization	16%
	Security issues: disclosure, vulnerabilities, exploits	12%
	Type safety: Using an object as the wrong type	12%
Tools are on top for years!	Memory safety: Memory leaks	12%
	Moving existing code to the latest language standard	9%

Candy, Vitamin, Painkiller

“We throw away the candy.

We look at vitamins.

We really like painkillers.”

© Kevin Fong, venture capitalist

Cure, Vitamin, Painkiller

Painkiller = a quick-fix, a tool

Vitamin = guidelines, code styles, best practices

Cure = a language feature

Simple enough to learn as a first-time programmer

-

...It even looks way more complicated than using C++...

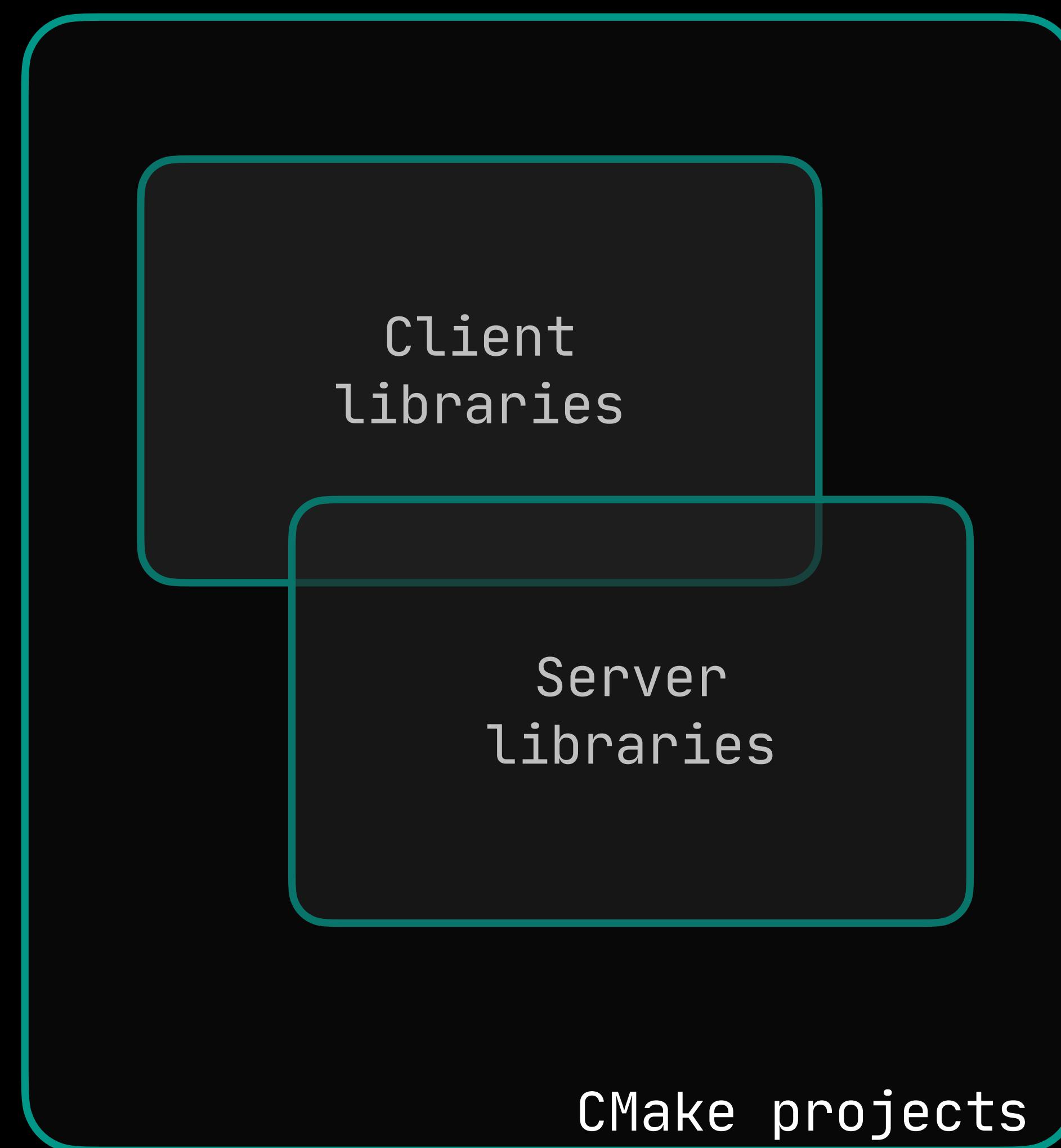
```
race:  
    # Move back to its starting position.  
    set NavResultGoToNext = Navigatable.NavigateTo(NavTargetEnd)  
  
    # Sets NPC to look at its previous position which will make it walk backwards.  
    # This is meant to show the utility of the focus interface.  
    Focus.MaintainFocus(GoToPoint)  
  
    if>ShowAIDebug?:  
        DrawDebugLookAt(Character, GoToPoint)  
    else:  
        Sleep(Inf)
```

Verse

Story time:

“Dependency validation with NetBeans”

Dependency validation with NetBeans



\$ cmake ..

CMake configures the
project

NetBeans

\$ cmake --build

CMake builds an already-
generated project binary
tree

Tools DO help.

Practical examples for C++

Tools help: with macros #1

```
#define MAGIC 100
#define CALL_DEF(val, class_name) int call_##class_name() \
{ return val; }
```

```
#define CLASS_DEF(class_name) class class_##class_name { \
public: \
    int count_##class_name; \
    CALL_DEF(MAGIC, class_name) \
};
```

```
CLASS_DEF(A)
CLASS_DEF(B)
CLASS_DEF(C)
```

Declared in: MacroDebug.cpp

Definition:

```
#define CLASS_DEF(class_name) class class_##class_name { \
public: \
    int count_##class_name; \
    CALL_DEF(MAGIC, class_name) \
};
```

Replacement:

```
class class_A { \
public: \
    int count_A; \
    int call_A() { return 100; } \
};
```



Declared in: MacroDebug.cpp

Definition:

```
#define CLASS_DEF(class_name) class class_##class_name { \
public: \
    int count_##class_name; \
    CALL_DEF(MAGIC, class_name) \
};
```

Replacement:

```
class class_C { \
public: \
    int count_C; \
    int call_C() { return 100; } \
};
```



Tools help: with macros #2

```
BOOST_AUTO_TEST_SUITE(demo_suite)

BOOST_AUTO_TEST_CASE(test_equal_1
{
    BOOST_CHECK_EQUAL(MY_ONE, 1);
}

BOOST_AUTO_TEST_CASE(test_equal_wo
{
    char *name = new char[4];
    name[1] = 't';
    name[2] = 'e';
    name[3] = 's';
    name[4] = 't';

    BOOST_CHECK_EQUAL(name, "test"
}
BOOST_AUTO_TEST_SUITE_END()
```

Declared in: unit_test_suite.hpp

Definition:

```
#define BOOST_AUTO_TEST_SUITE(...) \
    BOOST_TEST_INVOKE_IF_N_ARGS( 1, \
        BOOST_AUTO_TEST_SUITE_NO_DECOR, \
        BOOST_AUTO_TEST_SUITE_WITH_DECOR, \
        __VA_ARGS__)
```

Replacement:

```
struct test_equal_word : public BOOST_AUTO_TEST_SUITE_FIXTURE {
    void test_method();
};

static void test_equal_word_invoker() {
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" fixture ctor").str());
    test_equal_word t;
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" fixture setup").str());
    boost::unit_test::setup_conditional(t);
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" test entry").str());
    t.test_method();
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" fixture teardown").str());
    boost::unit_test::teardown_conditional(t);
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" fixture dtor").str());
}

struct test_equal_word_id {
};

static boost::unit_test::ut_detail::auto_test_unit_registrar test_equal_word_registrar162 __attribute__((__unused__))( \
    boost::unit_test::make_test_case(&test_equal_word_invoker, "test_equal_word", "_file_name_", 16), \
    boost::unit_test::decorator::collector_t::instance());

void test_equal_word::test_method()
```

Tools help: with overloads

```
class Fraction {...};

bool operator==(const Fraction& lhs, const Fraction& rhs) {...};

bool operator!=(const Fraction& lhs, const Fraction& rhs) {...};

Fraction operator*(Fraction lhs, const Fraction& rhs) {...};

std::ostream& operator<<(std::ostream& out, const Fraction& f)
{
    return out << f.num() << '/' << f.den();
}

void fraction_sample() {
    Fraction f1(3, 8), f2(1, 2);

    std::cout << f1 << " * " << f2 << " = " << f1 * f2 << '\n';
}
```

Tools help: with overloads

1

```
std::cout << f1 << " * " << f2 << " = " << f1 * f2 << '\n';
```

Declared in: operator_usages_highlight.cpp

```
std::basic_ostream<char> &operator<<(std::basic_ostream<char> &out, const Fraction & :)
```

2

```
std::cout << f1 << " * " << f2 << " = " << f1 * f2 << '\n';
```

Declared in: ostream

namespace std

template<_Traits>

```
basic_ostream<char, _Traits> &operator<<(basic_ostream<char, _Traits> &__os, const char *__str)
```

"operator<<" on cppreference.com ↗



3

```
std::cout << f1 << " * " << f2 << " = " << f1 * f2 << '\n';
```

Declared in: ostream

namespace std

public:

```
basic_ostream &basic_ostream::operator<<(int __n)
```

Tools help: named parameters

Named arguments, by Ehsan Akhgari and Botond Ballo, [N4172](#)

Self-explanatory Function Arguments, by Axel Naumann, [P0671](#)

Boost.Parameter, by David Abrahams and Daniel Wallin, [docs](#)

[CppCon 2018: Richard Powell “Named Arguments from Scratch”](#)

Tools help: named parameters

#1 Distinguish
parameters of
the same type

```
4
5 → GameState::GameState(int fieldWidth,
6                               int fieldHeight)
7           : field_( aleft: 0, atop: 0, awidth: fieldWidth, aheight: fieldHeight),
8           ball_( pos: QPointF( xpos: field_.width() / 2, ypos: field_.height() - 30),
9                     speed: QPointF( xpos: 200, ypos: -200)),
10          paddle_( pos: QPointF( xpos: field_.width() / 2, ypos: field_.height() - 10),
11                         width: 60, height: 20),
12          score_(0) {
```

Tools help: named parameters

#2 Pass-by-value and pass-by-reference

```
→ void GameState::processCollisions() {
    if (applyCollision( & ball_, type: getCollisionWithWalls( b: ball_, bounding: getField()))

        for (auto iter : iterator<Brick *> = bricks_.begin(); iter != bricks_.end(); ++iter) {
            if (applyCollision( & ball_, type: getCollisionWithBrick( b: ball_, brick: iter->aabb))
                bricks_.erase( position: iter);
                score_ += 1;
                return;
            }

        if (applyCollision( & ball_, type: getCollisionWithBrick( b: ball_, brick: paddle_.aabb()))
    }
```

Tools help: type hints

#1 auto variables

#2 lambda types

#3 deduced return types

#4 structured bindings

```
template<typename T, typename U>
auto doOperation(T t, U u) -> decltype(t + u) {
    return t + u;
}

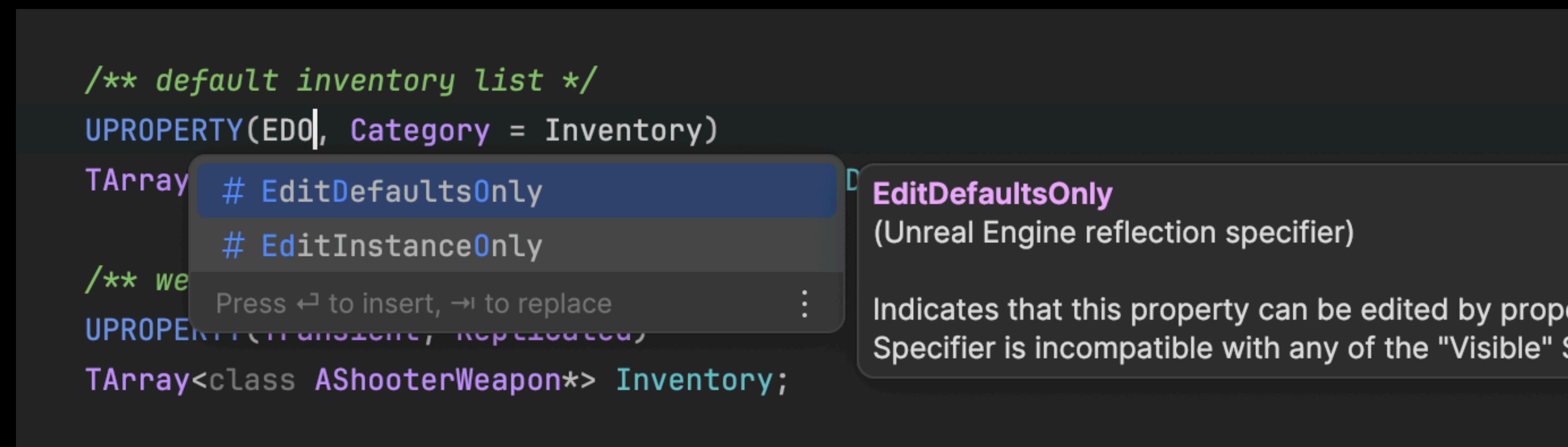
void fun_type() {
    auto op :int = doOperation( t: 3, u: 0);
    auto op1 :long = doOperation( t: 3L, u: 0);
    auto op2 :double = doOperation( t: 3.0, u: 0);

    std::cout << op << " " << op1 << " " << op2;
}

void type_hints_bind() {
    int a = 1, b = 2;
    const auto&[x :int &, y :int &] = std::tie( &a, &b);
    const auto tuple :const tuple<int, char, double> = std::tuple(1, 'a', 2.3);
    const auto& [i :const int , c :const char , d :const double ] = tuple;
}
```

Tools help: with DSLs

#1 Unreal Engine
reflection
macros-based dialect



#2 Qt signals



Tools help: with guidelines

Checks and fixes in ClangTidy

cppcoreguidelines-avoid-capturing-lambda-coroutines	
cppcoreguidelines-avoid-const-or-ref-data-members	
cppcoreguidelines-avoid-do-while	
cppcoreguidelines-avoid-goto	
cppcoreguidelines-avoid-non-const-global-variables	
cppcoreguidelines-avoid-reference-coroutine-parameters	
cppcoreguidelines-init-variables	Yes
cppcoreguidelines-interfaces-global-init	
cppcoreguidelines-macro-usage	
cppcoreguidelines-misleading-capture-default-by-value	Yes
cppcoreguidelines-missing-std-forward	
cppcoreguidelines-narrowing-conversions	
cppcoreguidelines-no-malloc	
cppcoreguidelines-no-suspend-with-lock	
cppcoreguidelines-owning-memory	
cppcoreguidelines-prefer-member-initializer	Yes
cppcoreguidelines-pro-bounds-array-to-pointer-decay	
cppcoreguidelines-pro-bounds-constant-array-index	Yes
cppcoreguidelines-pro-bounds-pointer-arithmetic	
cppcoreguidelines-pro-type-const-cast	
cppcoreguidelines-pro-type-cstyle-cast	Yes
cppcoreguidelines-pro-type-member-init	Yes
cppcoreguidelines-pro-type-reinterpret-cast	
cppcoreguidelines-pro-type-static-cast-downcast	Yes
cppcoreguidelines-pro-type-union-access	
cppcoreguidelines-pro-type-vararg	
cppcoreguidelines-rvalue-reference-param-not-moved	
cppcoreguidelines-slicing	
cppcoreguidelines-special-member-functions	
cppcoreguidelines-virtual-class-destructor	Yes

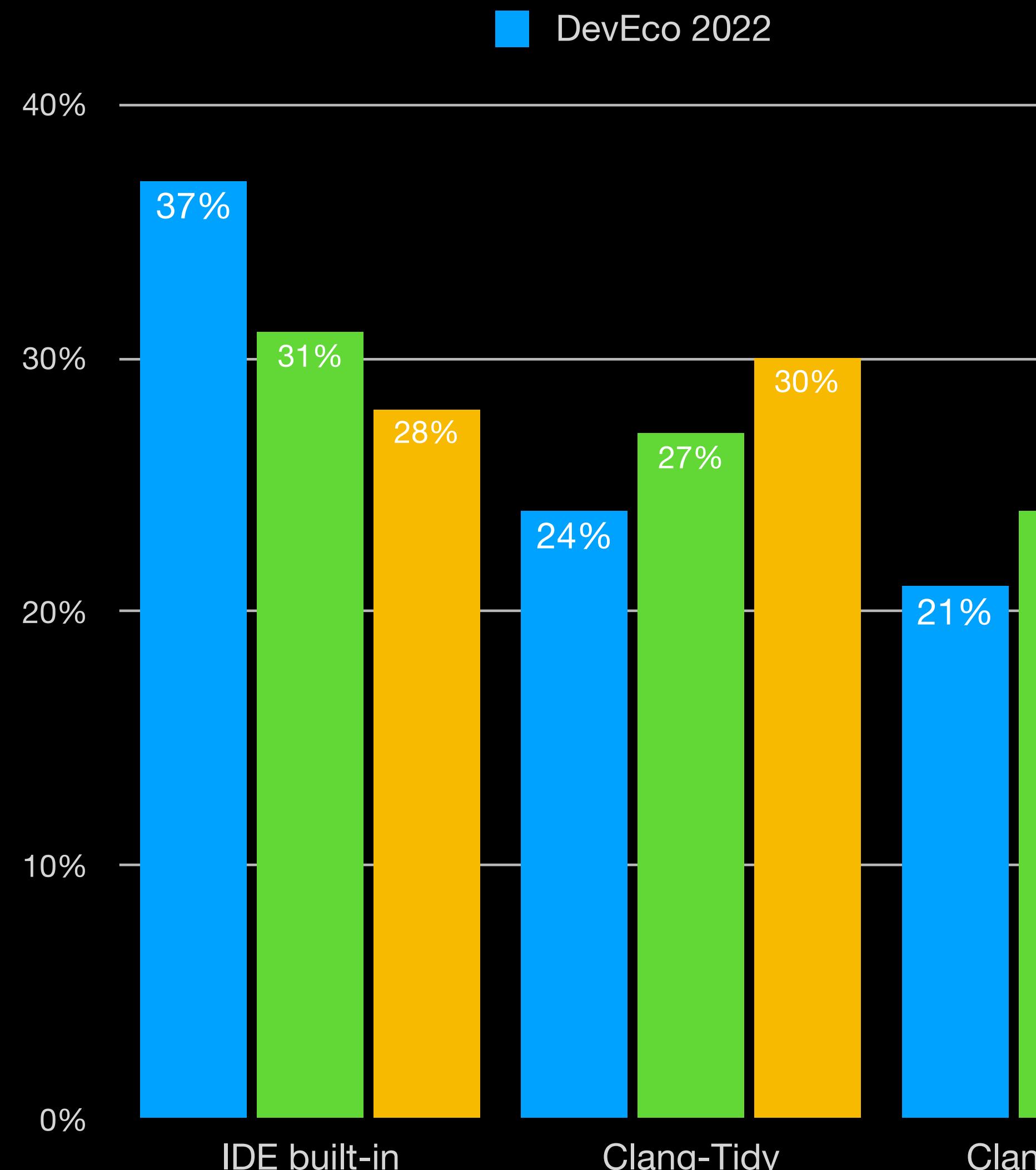
modernize-avoid-bind	Yes
modernize-avoid-c-arrays	Yes
modernize-concat-nested-namespaces	Yes
modernize-deprecated-headers	Yes
modernize-deprecated-ios-base-aliases	Yes
modernize-loop-convert	Yes
modernize-macro-to-enum	Yes
modernize-make-shared	Yes
modernize-make-unique	Yes
modernize-min-max-use-initializer-list	Yes
modernize-pass-by-value	Yes
modernize-raw-string-literal	Yes
modernize-redundant-void-arg	Yes
modernize-replace-auto_ptr	Yes
modernize-replace-disallow-copy-and-assign-macro	Yes
modernize-replace-random-shuffle	Yes
modernize-return-braced-init-list	Yes
modernize-shrink-to-fit	Yes
modernize-type-trait	Yes
modernize-unary-static-assert	Yes
modernize-use-auto	Yes
modernize-use-bool-literals	Yes
modernize-use-constraints	Yes
modernize-use-default-member-init	Yes
modernize-use-designated-initializers	Yes
modernize-use-emplace	Yes
modernize-use-equals-default	Yes
modernize-use-equals-delete	Yes
modernize-use-nodiscard	Yes
modernize-use-noexcept	Yes
modernize-use-nullptr	Yes
modernize-use-override	Yes
modernize-use-starts-ends-with	Yes
modernize-use-std-numbers	Yes
modernize-use-std-print	Yes
modernize-use-trailing-return-type	Yes
modernize-use-transparent-functors	Yes
modernize-use-ungaught-exceptions	Yes
modernize-use-using	Yes

Tools help: with guideline



Bryce Adelstein Lelbach

Principal Architect at NVIDIA



I think the decrease in IDE-provided analysis tools here is indicative of people incorporating static analysis into their CI, like running the Clang-tidy/ClangFormat/Clang static analyzer in GitHub Actions.

X (formerly Twitter)

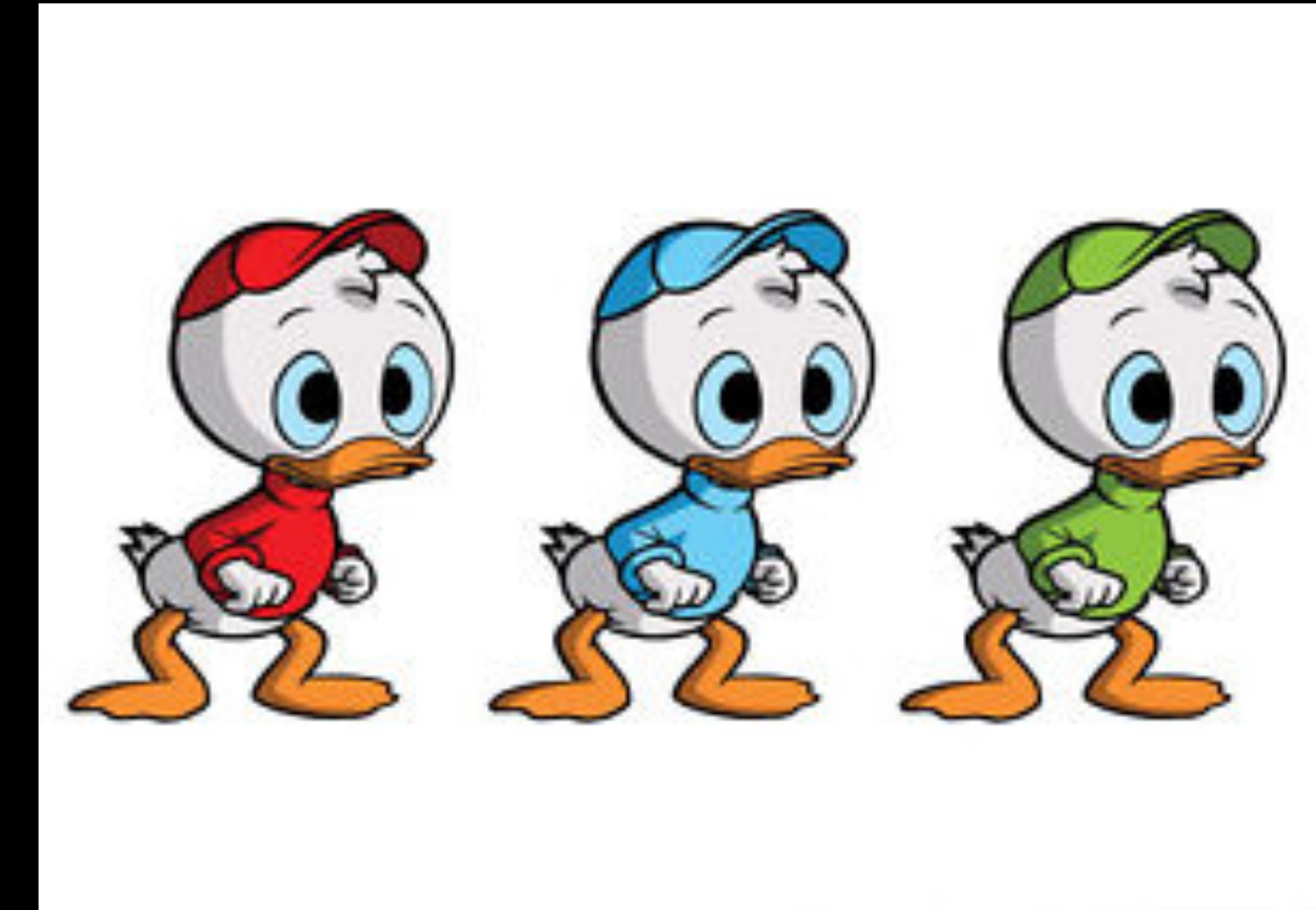
C++ and Toolability: A story about friendship?

C++ toolability: sameness

```
template<class T, int ... X>
T pi(T(X...));
```



```
int main() {
    return pi<int, 42>;
}
```



```
constexpr auto rexpr = ^int;
typename[:rexpr:] a = 42;
```

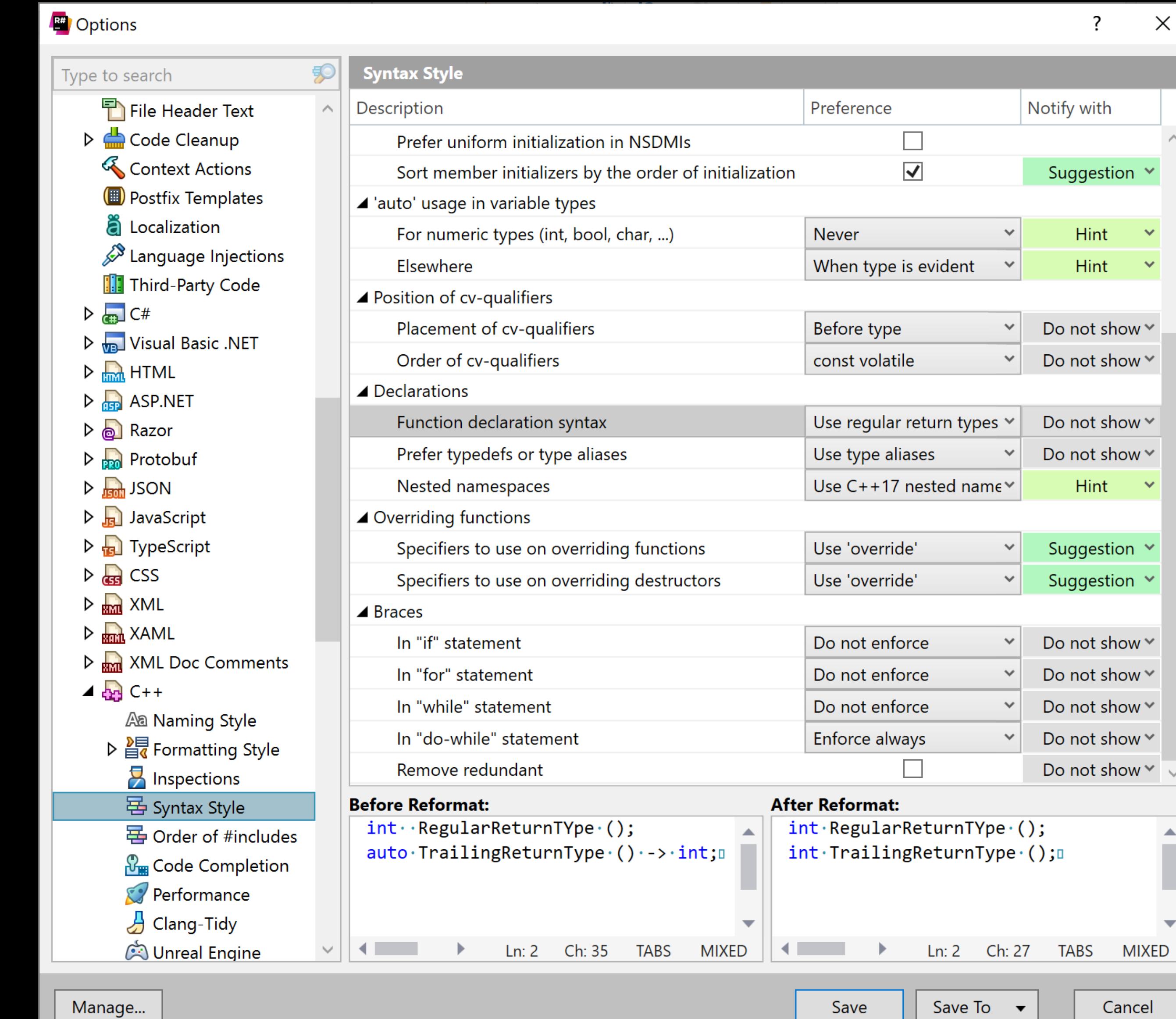
C++ toolability: sameness

Almost Always Auto or when type is evident or never for numeric types.

Const before or after the type it applies to.

Trailing return type for lambdas or always.

Virtual explicitly (UE) or override/ final and no virtual (C++ Core Guidelines).



C++ toolability

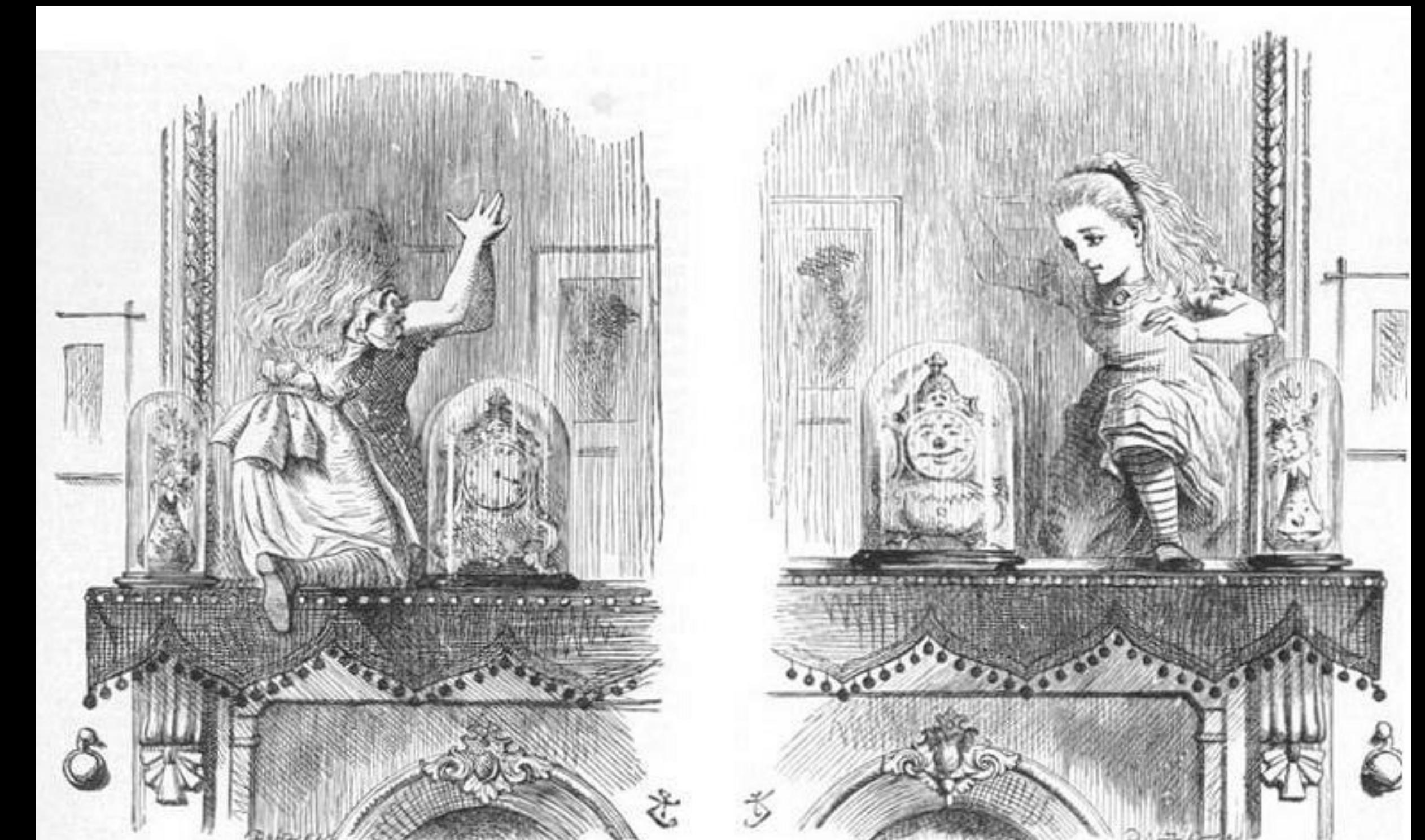
```
int(x), y, a, b, c, d, e, f, *z;
```

```
int(x), y, a, b, c, d, e, f, new int;
```



C++ toolability

To parse C++
we need to distinguish
types from non-types



C++ toolability

Resolve depends on:

- order of the definitions
- default arguments
- overload resolution
- project model context

```
//foo.h
#ifndef MAGIC
template<int>
struct x {
    x(int i) { }
};
#else
int x = 100;
#endif

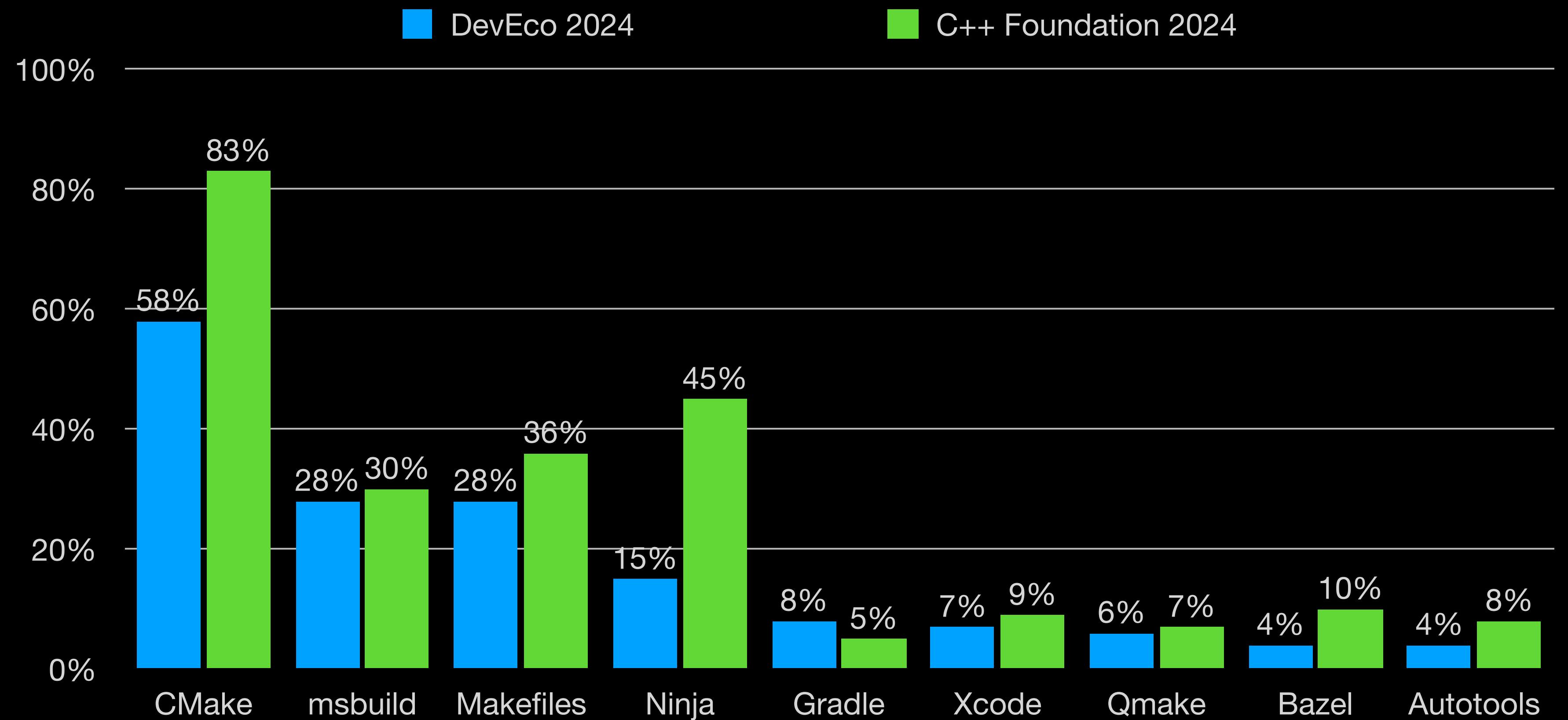
//foo.cpp
#include "foo.h"
void test(int y) {
    const int a = 100;
    auto k = x<a>(0);
}
```

x<100> k = x<a>(0)
Size = 1 byte

bool k = x<a>(0)
Size = 1 byte

C++ toolability

- CMake!
- msbuild for GameDev
- Makefiles for Embedded
- Bazel for monorepos



C++ and Tools:

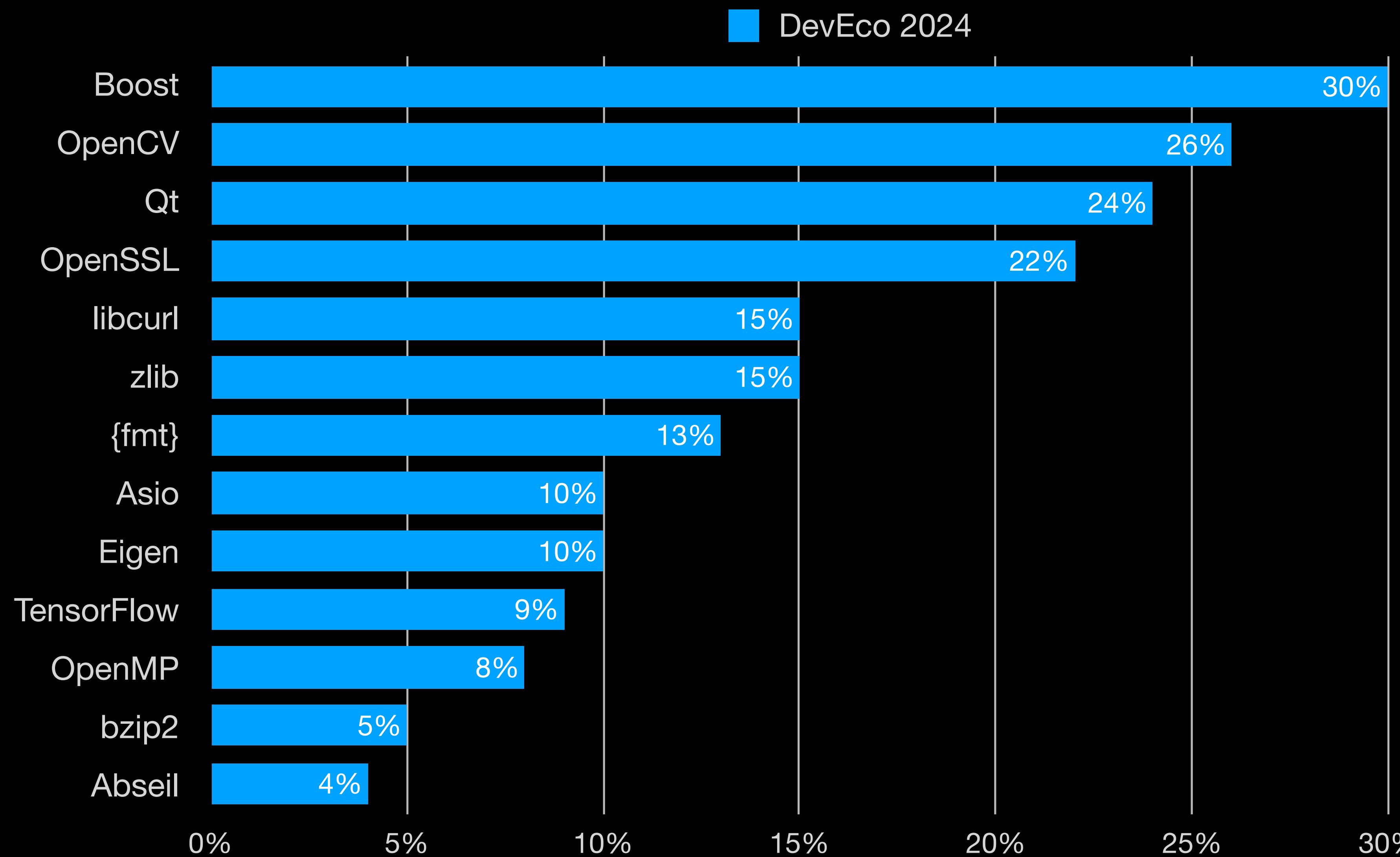
Assign a painkiller

Tools for C++ painpoints

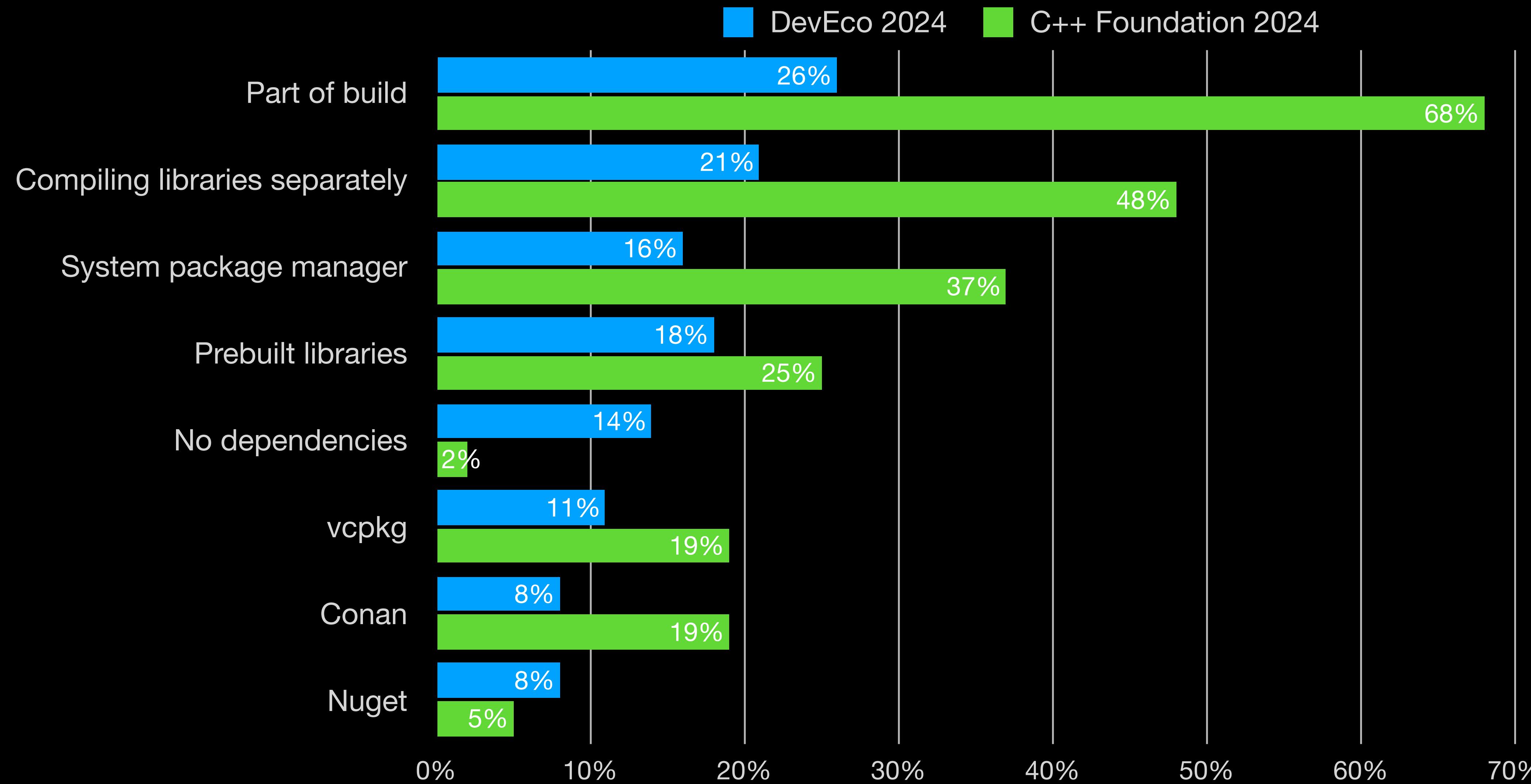
Which tools to use?

<i>Which of these do you find frustrating about C++ dev?</i>	<i>Major %</i>
Managing libraries my application depends on	45%
Build times	43%
Setting up a CI pipeline from scratch	30%
Managing CMake projects	30%
Concurrency safety: Races, deadlocks, performance bottlenecks	27%
Setting up a dev env from scratch	26%
Parallelism support	23%
Managing Makefiles	20%
Memory safety: Bounds safety issues	20%
Memory safety: Use-after-delete/free	20%
Debugging issues in my code	18%
Managing MSBuild projects	16%
Unicode, internationalization, and localization	16%
Security issues: disclosure, vulnerabilities, exploits	12%
Type safety: Using an object as the wrong type	12%
Memory safety: Memory leaks	12%
Moving existing code to the latest language standard	9%

Painpoints: #1 Managing libraries



Painpoints: #1 Managing libraries



Painpoints: #1 Managing libraries

40% repos with
`CMakeLists.txt` inside

Query summary

Query type: **Count**

✓ Forks dropped
✓ Archived dropped

Primary language of repositories: `C` `C++`

Licenses: Any

Updated after: **01/01/2020, 00:00:00**

Stars: **[10 .. Any]**

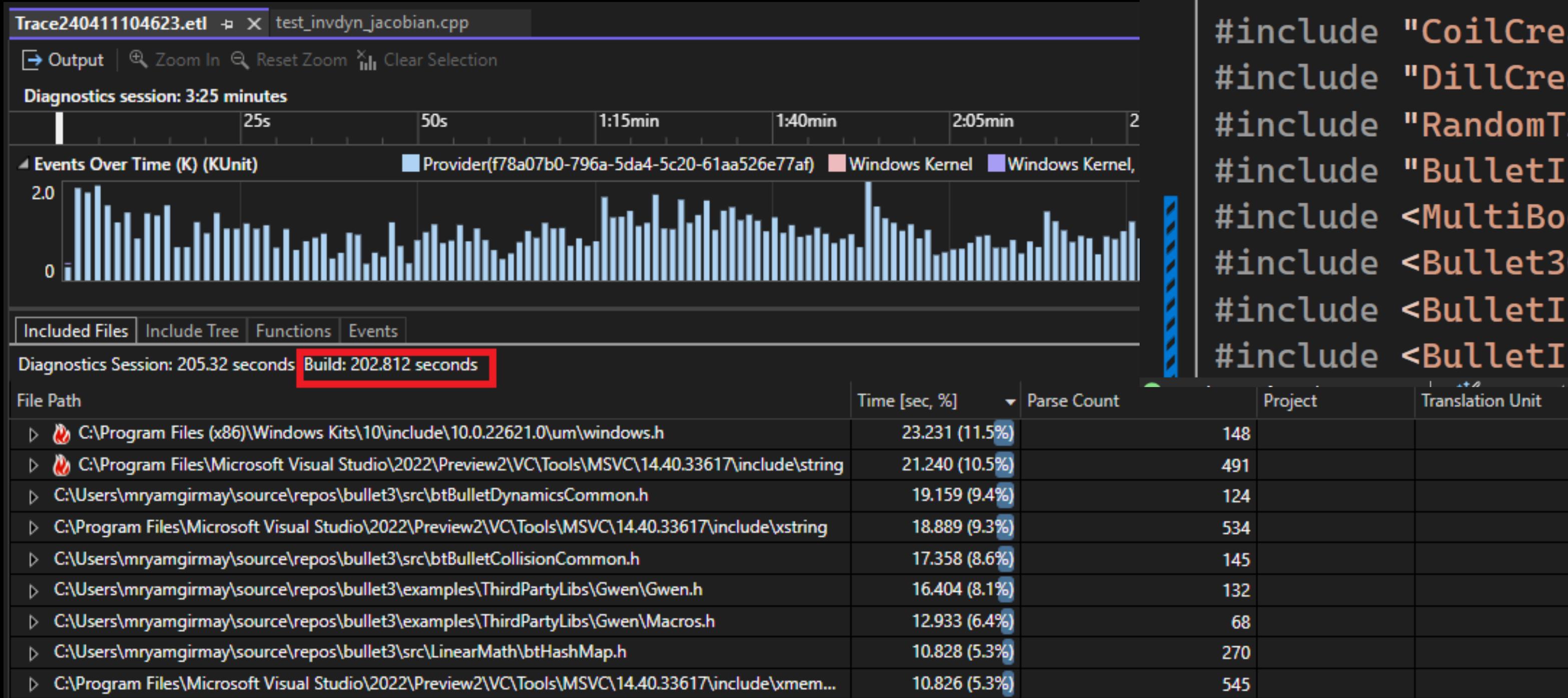
File's language: `Text` `CMake`

File's path **REGEX: CMakeLists.txt**

Percentage of repositories to analyze, %: **100**

Painpoints: #2 Build times

- Build insights
- Include cleanup
- Include Diagnostics

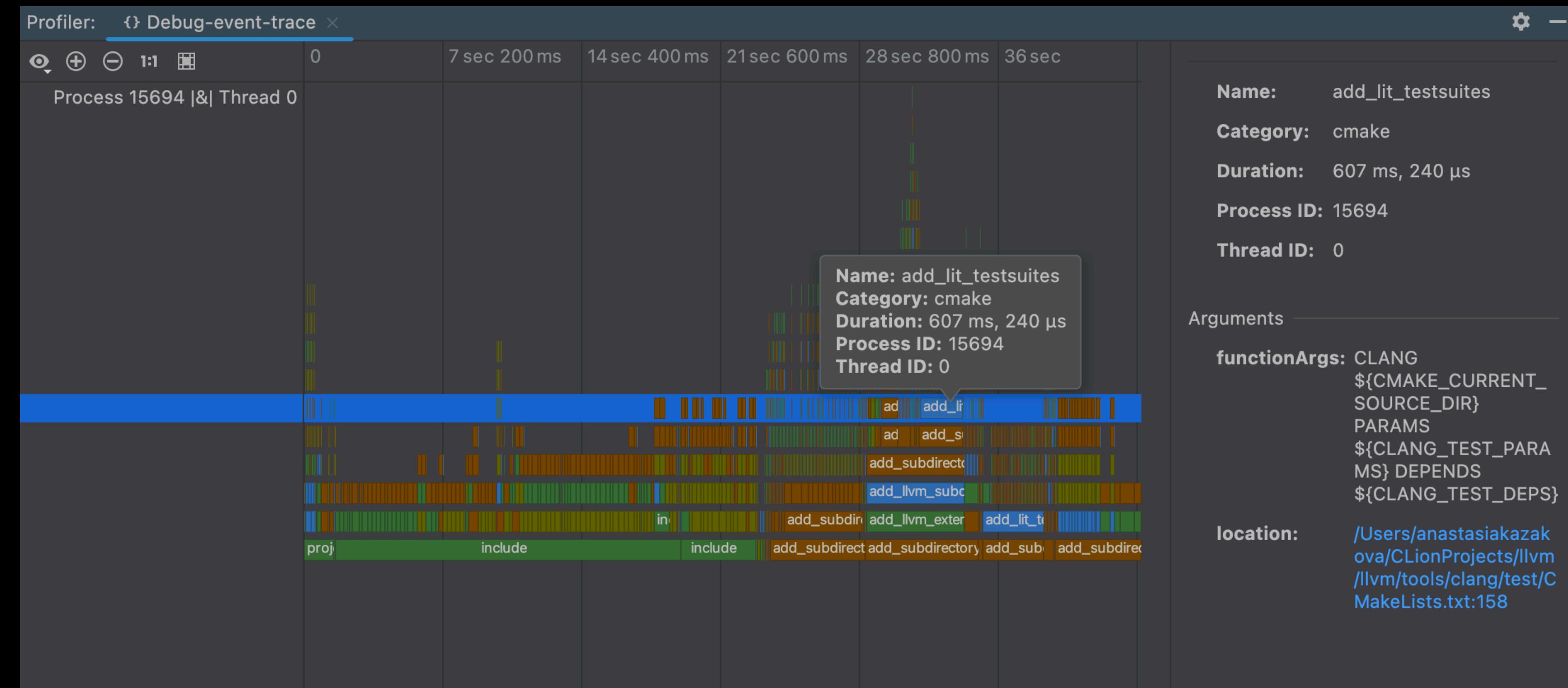


```
✓// Test of kinematic consistency: check if finite differences of
|// match positions
▶
✓#include <gtest/gtest.h>
#include "Bullet3Common/b3Random.h"

#include "CloneTreeCreator.hpp"
#include "CoilCreator.hpp"
#include "DillCreator.hpp"
#include "RandomTreeCreator.hpp"
#include "BulletInverseDynamics/MultiBodyTree.hpp"
#include <MultiBodyTreeCreator.hpp>
#include <Bullet3Common/b3Scalar.h>
#include <BulletInverseDynamics/details/IDLinearMathInterface.hpp>
#include <BulletInverseDynamics/IDConfig.hpp>
```

Painpoints: #2 Build times

Make configuration
profiling



Painpoints: #3 CI pipelines

- test frameworks
- CI integrations
- variety of toolchains
- visual tools
- IDEs integrations

The screenshot shows the TeamCity Pipelines interface for a project named "Open Source Project". The current run is labeled "Run #15" and is shown as "Running" with a status bar indicating "In queue 1m 23s" and "Running time 20m". The pipeline is optimized with a -20% reduction. The triggered by section shows "Noah Schneider at 8 Aug 16:45" and the repository is "github.com/Noah/Open-Source-project +2".

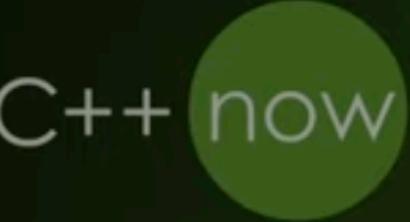
The pipeline consists of several stages:

- Frontend - Build_1: 1m 3s, Success, Tests passed: 12
- Backend - Build: 2m, Success
- Tests: 1m 45s, Success, Tests passed: 3
- Frontend - Build_2: 1m 45s, Success
- Publish artifacts: Success
- Deployment: Step 1/2, In progress

A detailed log for the "Backend - Deploy" stage is visible on the right, showing the following timeline:

- 10:19:05 Finalize build settings
- ... 10:19:05 > Collecting changes in 1 VCS root 1s
- ... 10:19:05 > Parallel tests: freeze 1s
- 10:19:05 The build is removed from the queue to be prepared for the start
- ... 10:19:05 > Step 1/1: simpleRunner (Command Line) 10s
- 10:19:05 Will use agent side checkout
- 10:19:05 Clearing temporary directory: /mnt/agent/temp/buildTmp
- 10:19:05 Container wrapper: prepare reusable container
- 10:19:05 Process exited with code 1
- 10:19:05 Container wrapper: prepare reusable container
- 10:19:05 Process exited with code 1
- 10:19:05 Process exited with code 1
- 10:19:05 Process exited with code 1
- 10:19:05 The build is removed from the queue to be prepared for the start

Painpoints: #4 Managing CMake



CMake + Conan: 3 Years Later

A Motivating Example (Poll #2)

What will be printed during the configuration phase?

```
set(foo ON)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

8

 Bloomberg Engineering

CppNow.org

Painpoints: #4 Managing CMake

- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI

The screenshot shows a code editor with CMake code. A completion dropdown is open at the cursor position, showing suggestions for 'arkanoid'. The suggestions include:

- @ arkanoid (in Debug_demo/CMakeLists.txt)
- @ arkanoidLib (in Debug_demo/CMakeLists.txt)
- @ arkanoidTest (in Debug_demo/CMakeLists.txt)

The code editor also displays a tooltip for the 'add_executable' command, which includes:

- Contents
 - add_executable
 - Normal Executables
 - Imported Executables
 - Alias Executables
 - See Also
- Add an executable to the project using the specified source files.

Painpoints: #4 Managing CMake

—

- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI

The screenshot shows the CLion IDE interface. At the top, there's a status bar with 'demoMore' and 'master'. Below it is a file browser showing a folder structure. The main editor window displays the 'CMakeLists.txt' file with the following content:

```
22
23 if (WIN32)
24     copy_dll(arkanoid Core)
25     copy_dll(arkanoid Gui)
26     copy_dll(arkanoid Widgets)
27
28     if (MINGW) # hacky way to make things work; proper way would be building Qt with correct
29         add_custom_command(
30             TARGET arkanoid POST_BUILD
31             COMMAND ${CMAKE_COMMAND} -E copy_if_different
32             ${TARGET_FILE_DIR}:Qt${QT_VERSION}::Widgets>/libstdc++-6.dll
33             ${TARGET_FILE_DIR}:arkanoid>
34     )
35 endif()
36
37 add_subdirectory(3rdparty/googletest)
38 add_executable(arkanoidTest test.cpp)
39 target_link_libraries(arkanoidTest gtest_main arkanoidLib)
```

The line 'endif()' at line 34 is highlighted in green. The line 'add_subdirectory(3rdparty/googletest)' at line 37 is highlighted in blue. The line 'target_link_libraries(arkanoidTest gtest_main arkanoidLib)' at line 39 is also highlighted in blue.

At the bottom, the debugger interface is visible, showing a list of symbols and variables. The 'Debug' tab is selected. The list includes:

- CMake
- add_subdirectory Debug_demo/CMa
- CLionFullDemo/CMakeLists.txt:36

Details for 'add_subdirectory Debug_demo/CMa':

- > ENV
- > CMakeCache.txt
- > targets
- ≡ QT_VERSION = "6"
- ≡ SOURCE_FILES = "main.cpp;GameWidget.cpp;GameWidget.h"

Other tabs in the debugger include 'Threads & Variables' and 'Console'. A status bar at the bottom indicates the current frame: CLionFullDemo > Debug_demo > CMakeLists.txt.

Painpoints: #4 Managing CMake

- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI

The screenshot shows a dark-themed AI assistant interface. At the top, there are buttons for "AI Assistant", "All Chats", "New Chat", and a menu icon. In the main area, a message from "Me" asks "How to link openmp to CMake". The AI Assistant responds with a code snippet:

```
CMake
find_package(OpenMP)

if(OpenMP_CXX_FOUND)
    add_executable(your_executable your_source_files.cpp)
    target_link_libraries(your_executable PUBLIC
        ${OpenMP_CXX_LIBRARIES})
```

Below the code, a note says "✓ 7 more line(s)".

In the above example:

- `find_package(OpenMP)` will look for the OpenMP in your system. If CMake is able to find OpenMP, the boolean variable `OpenMP_CXX_FOUND` is set to true, else false.

Painpoints: #6 Setting up dev env

- start instantly
- onboard easily
- zero configuration time
- local & remote
- reproducible dev artifacts

The image shows two screenshots related to setting up a development environment.

Top Screenshot: A screenshot of the Microsoft Remote Development extension page on the Microsoft Store. It features a large blue icon with white arrows pointing right and left. The title "Remote Development" is displayed, along with the developer "Microsoft" and the URL "microsoft.com". It shows "5,438,310 installs" and a 5-star rating "(111)". The text describes it as an extension pack for opening any folder in a container or remote machine. A green "Install" button and a "Trouble Installing?" link are present.

Bottom Screenshot: A screenshot of the VS Code interface. On the left, the Explorer sidebar shows a repository named "cool-project" with a single item "cool-repo". The main editor area displays the contents of "cool-project/main/README.md". On the right, a "Dev environment" tab is selected in the top navigation bar. A tooltip explains what a Devfile is: "A devfile is like a template for a development environment. Use it to define which version of the IDE to work with, configure required parameters, and pre-fill environment variables." Red arrows point to the "Start coding" button in the top right and the "Dev environment" tab in the bottom right.

Painpoints: #8 Thread, memory, and type safety

Painkillers

- Data Flow Static Analysis
- Sanitizers
- Valgrind

Painpoints: #8 Thread, memory, and type safety

DFA: lifetime safety

```
void sample() {  
    int *p = nullptr;  
    {  
        int x = 0;  
        p = &x;  
        std::cout << *p;  
    }  
    std::cout << *p;  
}
```

Local variable 'p' may point to memory which is out of scope :

int *p = nullptr



Painpoints: #8 Thread, memory, and type safety

DFA: lifetime safety

```
const char *sample() {
    auto string = std::string("text");
    auto view = std::string_view(string);
    auto ptr = view.begin();
    return ptr;
}
```

The address of the local variable 'view' may escape the function :

const char *ptr = view.begin()  :

Painpoints: #8 Thread, memory, and type safety

DFA: bounds safety

```
class Test {
    static const int width = 200;
    static const int height = 100;

    int matrix[height][width];

public:
    void test() {
        for (int i = 0; i < width; i++)
            for (int j = 0; j < height; j++)
                matrix[i][j] = 0;
    }
};
```

Index may have a value of '100' which is out of bounds :

Declared in: arrayIndexOutOfBounds.cpp

private:

int[height][width] Test::matrix



Painpoints: #8 Thread, memory, and type safety

DFA: Use-after-delete/free

```
static void delete_ptr(int *ptr) {  
    delete ptr;  
}  
  
int handle_pointer() {  
    int* int_ptr = new int;  
  
    delete_ptr(int_ptr);  
    *int_ptr = 1;  
                        
    return 0;  
}
```

Local variable 'int_ptr' may point to deallocated memory :

int *int_ptr = new int



Painpoints: #8 Thread, memory, and type safety

DFA: NULL dereferencing

```
class Deref {  
    int* foo() {  
        return nullptr;  
    }  
  
public:  
    void bar() {  
        int* buffer = foo();  
        buffer[0] = 0;  
    }  
};
```

Pointer may be null

int *Deref::buffer = foo()

Tools for C++ painpoints

+

Makefiles

MSBuild

Debug

C++ modernisation

<i>Which of these do you find frustrating about C++ dev?</i>	<i>Major %</i>
Managing libraries my application depends on	45%
Build times	43%
Setting up a CI pipeline from scratch	30%
Managing CMake projects	30%
Concurrency safety: Races, deadlocks, performance bottlenecks	27%
Setting up a dev env from scratch	26%
Parallelism support	23%
Managing Makefiles	20%
Memory safety: Bounds safety issues	20%
Memory safety: Use-after-delete/free	20%
Debugging issues in my code	18%
Managing MSBuild projects	16%
Unicode, internationalization, and localization	16%
Security issues: disclosure, vulnerabilities, exploits	12%
Type safety: Using an object as the wrong type	12%
Memory safety: Memory leaks	12%
Moving existing code to the latest language standard	9%

Tools build a safe environment for C++
developers where they can simply focus
on ideas and code them.

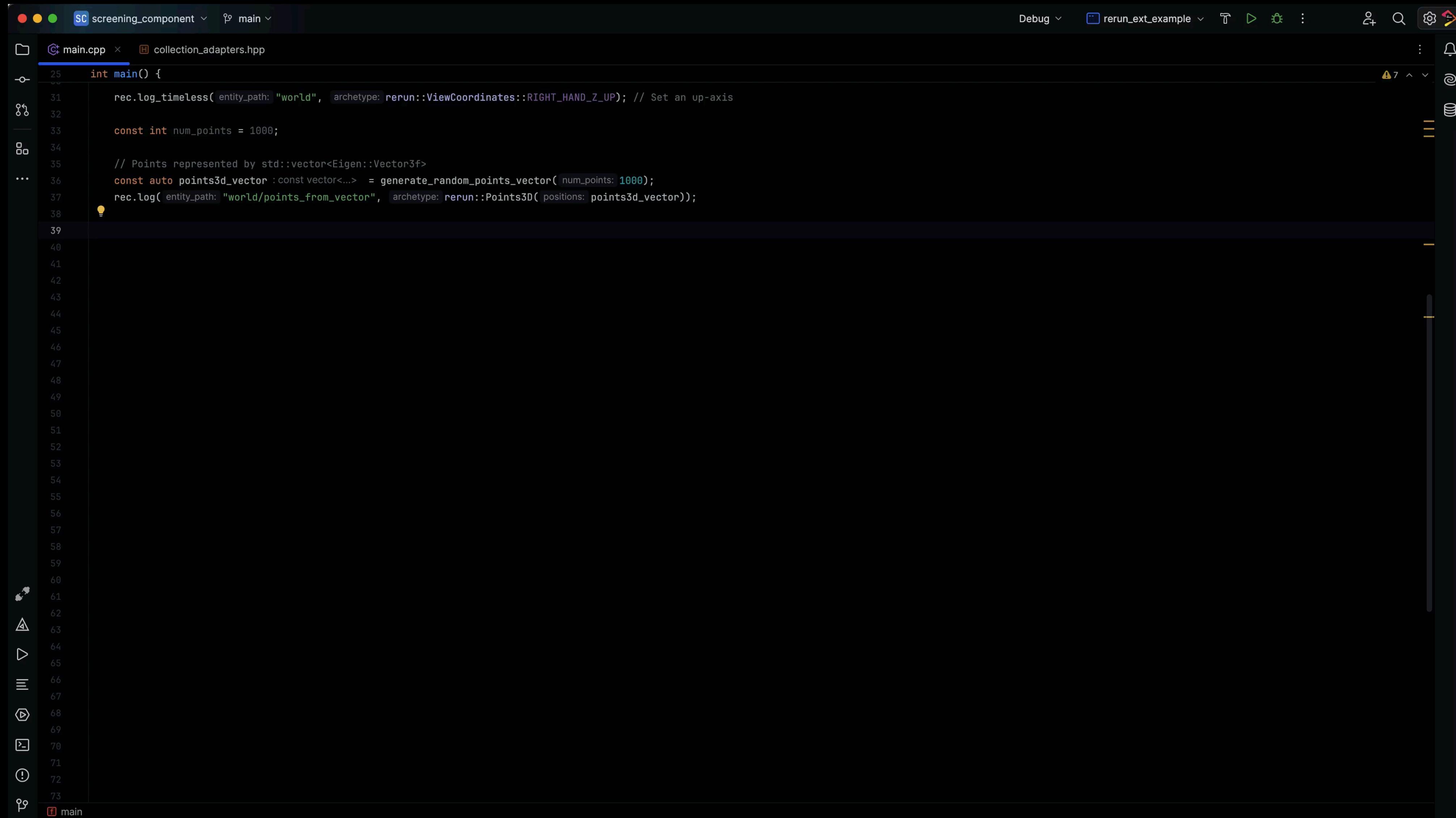
Happy AI-based coding in C++

Era of AI

- Knows open-source libraries
- Knows good (and bad) practices
- Knows the context of your project
- Teaches and guides you step-by-step

Era of AI

Writing
context-aware
code



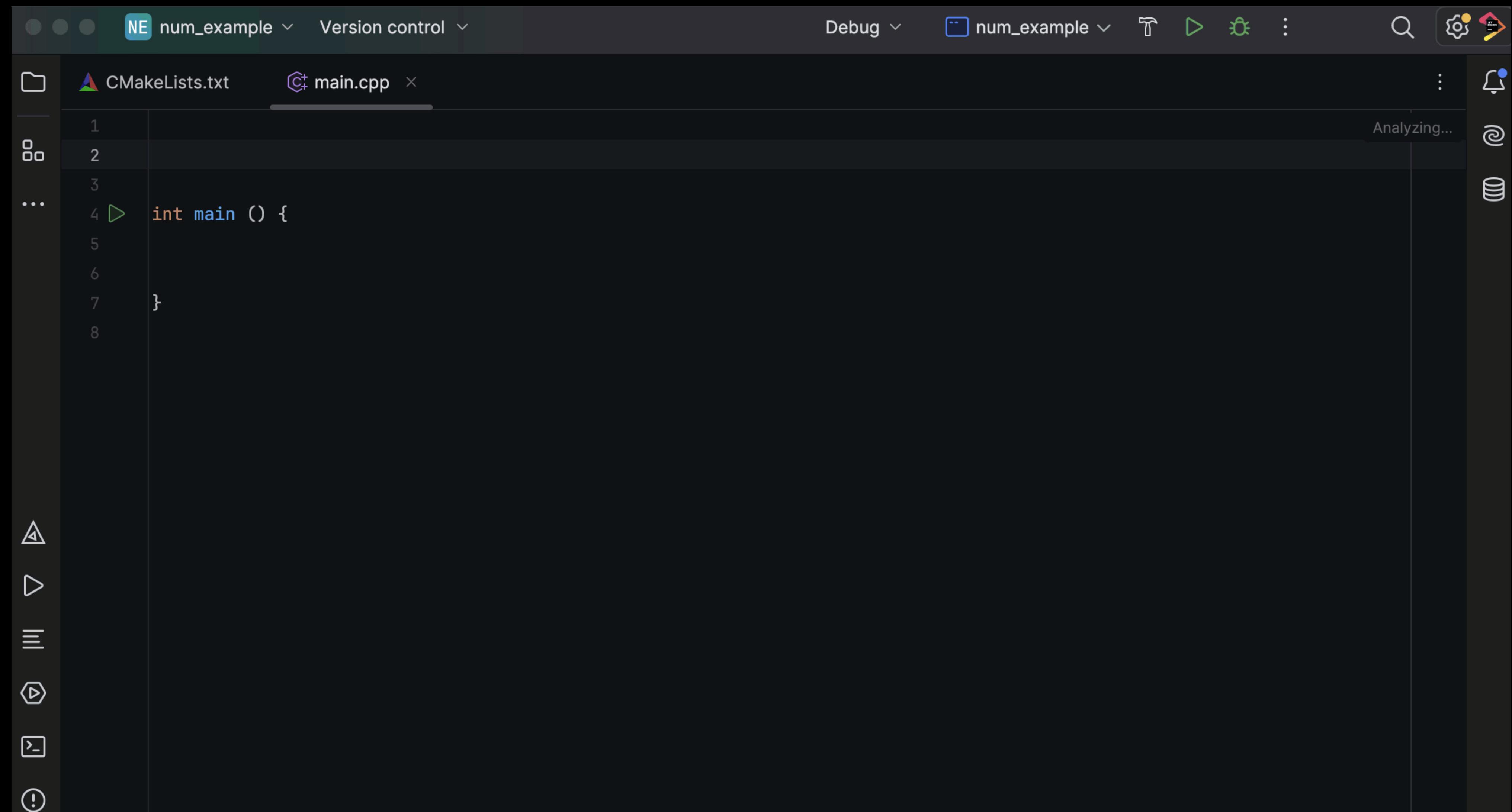
The screenshot shows a dark-themed code editor interface. At the top, there are tabs for 'main.cpp' and 'collection_adapters.hpp'. The main editor area displays the following code:

```
int main() {
    rec.log_timeless(entity_path: "world", archetype: rerun::ViewCoordinates::RIGHT_HAND_Z_UP); // Set an up-axis
    const int num_points = 1000;
    // Points represented by std::vector<Eigen::Vector3f>
    const auto points3d_vector :const vector<...> = generate_random_points_vector( num_points: 1000);
    rec.log( entity_path: "world/points_from_vector", archetype: rerun::Points3D( positions: points3d_vector));
}
```

The code uses the `rerun` library to log data to a session named "world". It sets an up-axis and generates 1000 random 3D points represented as `Eigen::Vector3f` and logs them under the path "world/points_from_vector". The code editor has a sidebar with icons for file operations like new, open, save, and delete.

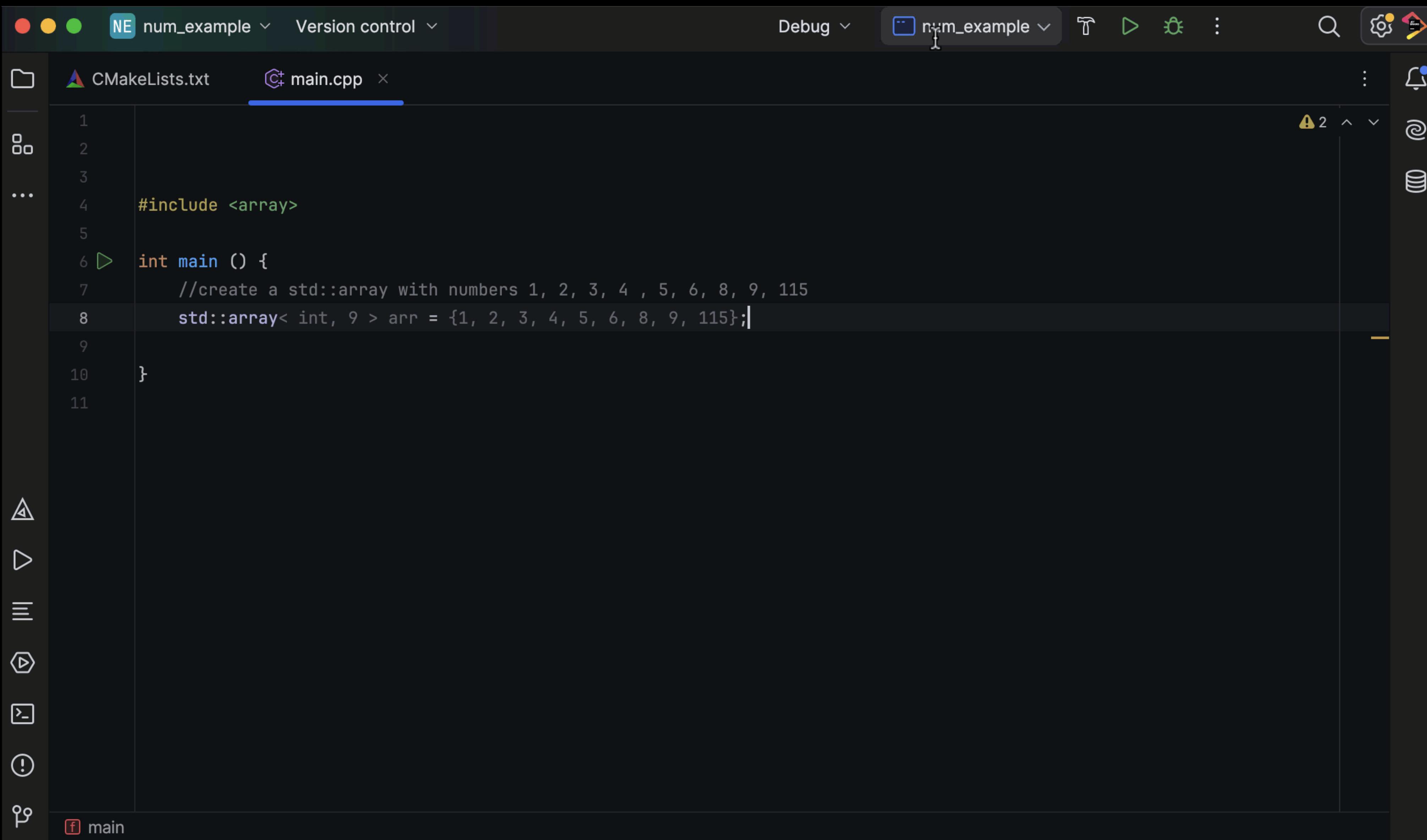
How to code productively
with AI in C++?

Rule 0: be specific



```
1
2
3
...
4 ▶ int main () {
5
6
7 }
8
```

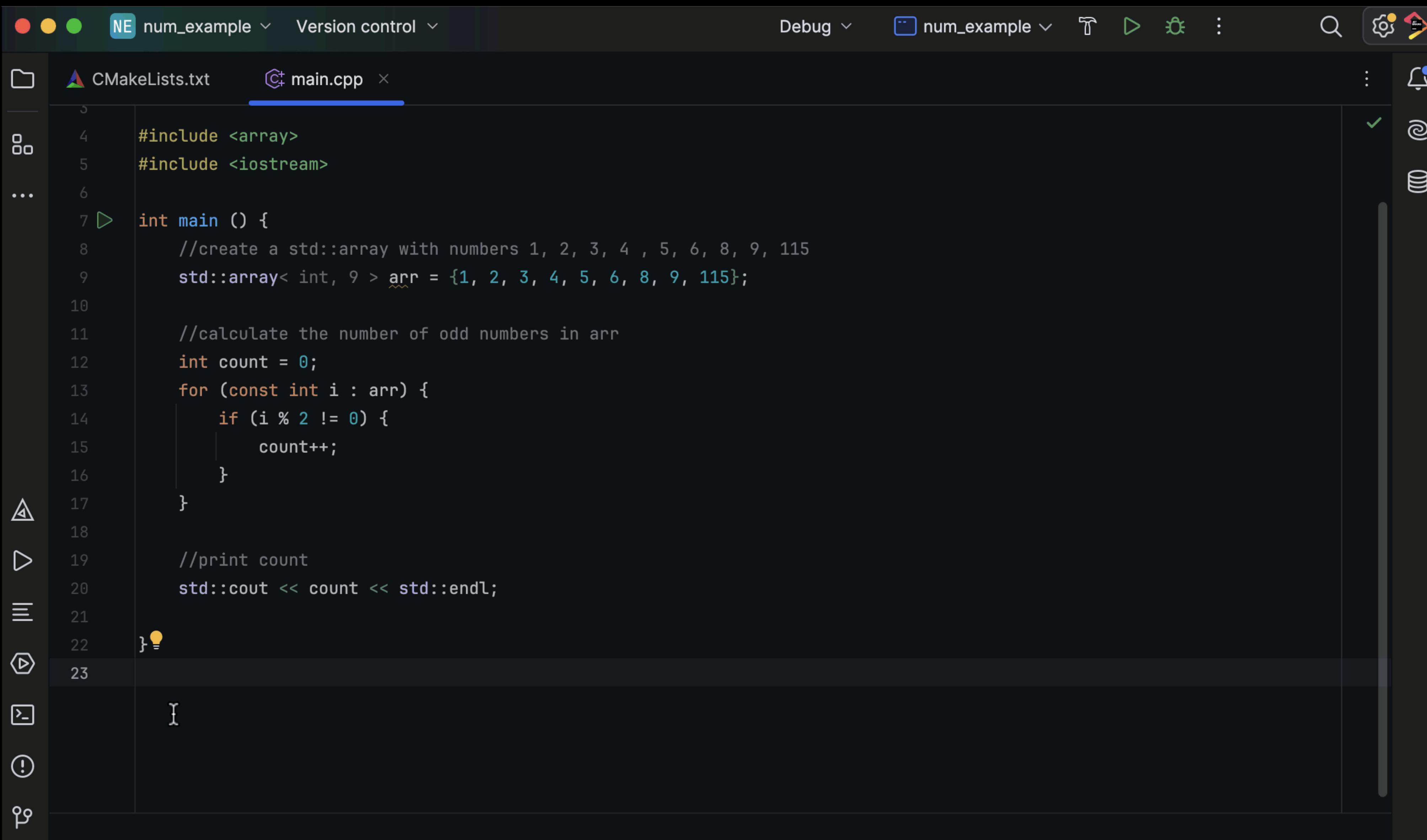
Rule 1: follow code analysis tips on top



A screenshot of a dark-themed IDE interface, likely Visual Studio Code, showing a C++ file named `main.cpp`. The code includes an `#include <array>` directive and a `main` function that creates a `std::array` with elements 1 through 115. A warning icon (a yellow triangle with a question mark) is visible in the status bar at the bottom right. The status bar also shows the file name `main`.

```
1
2
3
...
4 #include <array>
5
6 int main () {
7     //create a std::array with numbers 1, 2, 3, 4 , 5, 6, 8, 9, 115
8     std::array< int, 9 > arr = {1, 2, 3, 4, 5, 6, 8, 9, 115};|
9
10 }
11
```

Rule 2: refactor with AI



```
num_example Version control Debug num_example T D O : Search Settings Help

CMakeLists.txt main.cpp

5
4 #include <array>
5 #include <iostream>
...
7 int main () {
8     //create a std::array with numbers 1, 2, 3, 4 , 5, 6, 8, 9, 115
9     std::array< int, 9 > arr = {1, 2, 3, 4, 5, 6, 8, 9, 115};
10
11    //calculate the number of odd numbers in arr
12    int count = 0;
13    for (const int i : arr) {
14        if (i % 2 != 0) {
15            count++;
16        }
17    }
18
19    //print count
20    std::cout << count << std::endl;
21
22 }💡
23
```

Rule 3: re-iterate as in the learning process

My goal was to guide you gradually from the basic concept of code abstraction into more complex.

the lambda function `[](int i){ return i % 2 != 0; }` within snippet for counting elements satisfying specific condition.

Me :

Why didn't you suggest it initially?



I apologize if the initial suggestion did not meet your expectations. My goal was to guide you gradually from the basic concept of code abstraction into more complex, yet highly efficient ways of encapsulate the logic, allowing reusability and better readability.

The suggestion to use the `std::count_if` along with other Standard Template Library (STL) algorithms is an integral part of the learning process. It simplifies code and makes it more readable.

That's the learning process — sometimes it's more beneficial to start with simple concepts and gradually shift to more complex ones.

That's the learning process — sometimes it's more beneficial to start with simple concepts and gradually shift to more complex ones. However, if your preference is to receive the most optimal responses accordingly in future discussions.

Era of AI

Summarizing for
VCS or documentation

```
Collisions.cpp ×

7     if (ballBB.top() < bounding.top()) return Top;
8     return None;
9 }
10
11 → CollisionType getCollisionWithBrick(const Ball& b, const QRectF& bri
12     auto ballBB:QRectF = b.aabb();
13     auto inters:QRectF = ballBB.intersected(brick);
14     if (inters.isEmpty()) {
15         return None;
16     }
17     else {
18         if (inters.width() > inters.height()){
19             if(inters.center().y()
```

Tab typing



	Full line code completion	JetBrains AI Assistant
Feature set	Limited to code completion only	Code completion is just one of the features
Type of code completion provided	Single-line	Multi-line
Completion suggestion priority	Lower than multi-line code completion	Highest if enabled
Use case	When you know exactly what code you would like to write and want to save ~20% of the required keystrokes	When you know what your goal is but aren't sure exactly how to implement it
Local or cloud	Local	Cloud
Pricing	Included in your JetBrains IDE subscription	Comes at an additional cost* <small>* a free trial is available</small>

Thank you!

1. Painpoints and painkillers
2. Tools DO help
3. C++ toolability is hard
4. Tools for C++ painpoints
5. Era of AI

Q&A