C++ Painkillers

Anastasia Kazakova
About me

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

Head of Marketing & BizDev

@anastasiak2512

The Dutch C++ Group

cppunderthesea.nl
C++ Painkillers
-

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
Story #0 “Popov 🝼”
en.wikipedia.org/wiki/Aleksandr_Popov_(physicist)
Popov 📻
C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
Do we have a problem?

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

“How To Address 7 Major C++ Pain Points with CLion” © CLion blog

“C++ MythBusters” and “C++ MythBusters Strike 2” © Victor Ciura
C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
Vitamin, Painkiller, or Cure?

“We throw away the candy.
We look at vitamins.
We really like painkillers.”

© Kevin Fong, venture capitalist
Vitamin, Painkiller, or Cure?

Painkiller = a quick-fix, a tool
Vitamin = guidelines, code styles, best practices
Cure = a language feature
C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
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)
Tools help: with macros #2

static int v;

#define __NEW_VAR(name, num) static void *__v_##num = (void *)&name;
#define _NEW_VAR(name, num) __NEW_VAR(name, num)
#define NEW_VAR(name) _NEW_VAR(name, __COUNTER__)

void counter_macro_sample() {
  NEW_VAR(v)
  NEW_VAR(v)
  NEW_VAR(v)
}

Defined in: MacroDebug.cpp
Definition:
#define NEW_VAR(name) _NEW_VAR(name, __COUNTER__)
Replacement:
static void *__v_0 = (void *)&v;

Defined in: MacroDebug.cpp
Definition:
#define NEW_VAR(name) _NEW_VAR(name, __COUNTER__)
Replacement:
static void *__v_2 = (void *)&v;
Tools help: with macros #3

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_word)
{
    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()
Tools help: with macros #4

#define DECL(z, n, text) text ## n = n;
BOOST_PP_REPEAT(5, DECL, int x)

Declared in: repeat.hpp

Definition:

```cpp
#include <boost/preprocessor.hpp>

#define BOOST_PP_REPEATBOOST_PP_CAT(BOOST_PP_REPEAT_,
BOOST_PP_AUTO_REC(BOOST_PP_REPEAT_P, 4))

Replacement:

```cpp
int x0 = 0;
int x1 = 1;
int x2 = 2;
int x3 = 3;
int x4 = 4;
```
Tools help: with overloads

```cpp
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

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

Declared in: `operator_usages_highlight.cpp`

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

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

Declared in: `iostream`

```cpp
namespace std

template<typename Traits>
basic_ostream<char, Traits> &operator<<(basic_ostream<char, Traits> &os, const char *str);
```

"`operator<<" on cppreference.com"
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
Tools help: named parameters

CppCon 2018: Richard Powell “Named Arguments from Scratch”
Tools help: named parameters

#1 Parameters of the same type, like weight, height, positions, etc.

```cpp
GameState::GameState(int fieldWidth,
                    int fieldHeight)

    : field_( aleft: 0, atop: 0, awidth: fieldWidth, aheight: fieldHeight),
      ball_( pos: QPointF( xpos: field_.width() / 2, ypos: field_.height() - 30),
             speed: QPointF( xpos: 200, ypos: -200)),
      paddle_( pos: QPointF( xpos: field_.width() / 2, ypos: field_.height() - 10),
               width: 60, height: 20),
      score_(0) {
```
Tools help: named parameters

```cpp
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

Types:
- variables
- lambda
- deduced return types
- structured bindings
Tools help: with DSLs

Unreal Engine reflection macros-based dialect
**Tools help: with guidelines**

- **Checks and fixes in ClangTidy**

<table>
<thead>
<tr>
<th>Check/Rule</th>
<th>Yes/No</th>
</tr>
</thead>
<tbody>
<tr>
<td>cppcoreguidelines-avoid-capturing-lambda-coroutines</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-avoid-const-or-ref-data-members</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-avoid-do-while</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-avoid-goto</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-avoid-non-const-global-variables</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-avoid-reference-coroutine-parameters</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-init-variables</td>
<td>Yes</td>
</tr>
<tr>
<td>cppcoreguidelines-macro-usage</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-misleading-capture-default-by-value</td>
<td>Yes</td>
</tr>
<tr>
<td>cppcoreguidelines-missing-std-forward</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-narrowing-conversions</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-no-malloc</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-no-suspend-with-lock</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-owning-memory</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-prefer-member-initializer</td>
<td>Yes</td>
</tr>
<tr>
<td>cppcoreguidelines-pro-bounds-array-to-pointer-decay</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-pro-bounds-constant-array-index</td>
<td>Yes</td>
</tr>
<tr>
<td>cppcoreguidelines-pro-bounds-pointer-arithmetic</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-pro-type-const-cast</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-pro-type-castyle-cast</td>
<td>Yes</td>
</tr>
<tr>
<td>cppcoreguidelines-pro-type-member-init</td>
<td>Yes</td>
</tr>
<tr>
<td>cppcoreguidelines-pro-type-reinterpret-cast</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-pro-type-static-cast-downcast</td>
<td>Yes</td>
</tr>
<tr>
<td>cppcoreguidelines-pro-type-union-access</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-pro-type-vararg</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-rvalue-reference-param-not-moving</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-slicing</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-special-member-functions</td>
<td></td>
</tr>
<tr>
<td>cppcoreguidelines-virtual-class-destructor</td>
<td>Yes</td>
</tr>
<tr>
<td>modernize-avoid-bind</td>
<td></td>
</tr>
<tr>
<td>modernize-avoid-c-arrays</td>
<td></td>
</tr>
<tr>
<td>modernize-concat-nested-namespaces</td>
<td></td>
</tr>
<tr>
<td>modernize-deprecated-headers</td>
<td></td>
</tr>
<tr>
<td>modernize-deprecated-ios-base-aliases</td>
<td></td>
</tr>
<tr>
<td>modernize-loop-convert</td>
<td></td>
</tr>
<tr>
<td>modernize-macro-to-enum</td>
<td></td>
</tr>
<tr>
<td>modernize-make-shared</td>
<td></td>
</tr>
<tr>
<td>modernize-make-unique</td>
<td></td>
</tr>
<tr>
<td>modernize-min-max-use-initializer-list</td>
<td></td>
</tr>
<tr>
<td>modernize-pass-by-value</td>
<td></td>
</tr>
<tr>
<td>modernize-random-shuffle</td>
<td></td>
</tr>
<tr>
<td>modernize-return-braced-init-list</td>
<td></td>
</tr>
<tr>
<td>modernize-shrink-to-fit</td>
<td></td>
</tr>
<tr>
<td>modernize-type-traits</td>
<td></td>
</tr>
<tr>
<td>modernize-unnary-static-assert</td>
<td></td>
</tr>
<tr>
<td>modernize-use-auto</td>
<td></td>
</tr>
<tr>
<td>modernize-use-bool-literals</td>
<td></td>
</tr>
<tr>
<td>modernize-use-constraints</td>
<td></td>
</tr>
<tr>
<td>modernize-use-default-member-init</td>
<td></td>
</tr>
<tr>
<td>modernize-use-designated-initializers</td>
<td></td>
</tr>
<tr>
<td>modernize-use-emplace</td>
<td></td>
</tr>
<tr>
<td>modernize-use-equals-default</td>
<td></td>
</tr>
<tr>
<td>modernize-use-equals-delete</td>
<td></td>
</tr>
<tr>
<td>modernize-use-nodiscard</td>
<td></td>
</tr>
<tr>
<td>modernize-use-noexcept</td>
<td></td>
</tr>
<tr>
<td>modernize-use-nullptr</td>
<td></td>
</tr>
<tr>
<td>modernize-use-override</td>
<td></td>
</tr>
<tr>
<td>modernize-use-starts-ends-with</td>
<td></td>
</tr>
<tr>
<td>modernize-use-std-numbers</td>
<td></td>
</tr>
<tr>
<td>modernize-use-std-print</td>
<td></td>
</tr>
<tr>
<td>modernize-use-trailing-return-type</td>
<td></td>
</tr>
<tr>
<td>modernize-use-transparent-functors</td>
<td></td>
</tr>
<tr>
<td>modernize-use-uncought-exceptions</td>
<td></td>
</tr>
<tr>
<td>modernize-use-using</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Tools help: with guidelines

Which of the following tools do you or your team use for guideline enforcement or other code quality/analysis?

<table>
<thead>
<tr>
<th>Tool</th>
<th>2022</th>
<th>2023</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool provided by my IDE</td>
<td>37%</td>
<td>31%</td>
</tr>
<tr>
<td>Clang-tidy</td>
<td>24%</td>
<td>27%</td>
</tr>
<tr>
<td>ClangFormat</td>
<td>21%</td>
<td>24%</td>
</tr>
<tr>
<td>Clang-analyzer / Clang Static Analyzer</td>
<td>17%</td>
<td>18%</td>
</tr>
<tr>
<td>Cppcheck</td>
<td>10%</td>
<td>11%</td>
</tr>
<tr>
<td>Cpplint</td>
<td>7%</td>
<td>9%</td>
</tr>
<tr>
<td>SonarLint / SonarQube / SonarCloud</td>
<td>6%</td>
<td>6%</td>
</tr>
</tbody>
</table>

---

**Bryce Adelstein Leibach**  
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++ Painkillers stories

Story #1

“Dependency validation with NetBeans”
Dependency validation with NetBeans

$ cmake ..
CMake configures the project

$ cmake --build
CMake builds an already-generated project binary tree
C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
Role of the C++ tooling

C++Now 2017
A look at C++ through the glasses of a language tool

CppCon 2017
New standards to the rescue: the view through an IDE’s glasses

CppCon 2017
Tools from the C++ eco-system to save a leg

CppCon 2018
Debug C++ Without Running
C++ toolability: sameness

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

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

```cpp
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

```cpp
template<int>
struct x {
    int z;
    bool p;
    x(int i) {
    };

    void test(int y) {
        const int a = 100;
        auto k = x<a>(0);
        auto l = y<a>(0);
    }

    x<100> k = x<a>(0)
    Size = 8 bytes

    auto k = x<a>(0);

    bool l = y<a>(0)
    Size = 1 byte
```
```cpp
void test() {
    struct x {
    };

    struct y {
        y(x) {};
        x(z);
    };
}
```
C++ toolability

```cpp
void test() {
    struct x {
        x(int) { };
    };

    int y = 100;

    auto a = (x)-5;
    auto b = (y)-5;
}
```

- `x a = (x)-5
Size = 1 byte`
- `int b = (y)-5
Size = 4 bytes`
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

Fuzzy parsing is kinda possible but...
C++ toolability
-
Resolve depends on:
- order of the definitions
- default arguments
- overload resolution
- project model context

//foo.h
#ifdef 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);
}
C++ toolability

You wanna standard C++ Build System?
You got One!
It’s called CMake!
© Bryce Adelstein Lelbach, CppNow 2021
C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
## Tools for C++ painpoints

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

Which tools to use?
Tools for C++ painpoints

#1 Managing libraries
Tools for C++ painpoints

#1 Managing libraries

40% repos with CMakeLists.txt inside
Tools for C++ painpoints

- #2 Build times
- Build insights
- Include cleanup
- Include Diagnostics

```cpp
#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>
```
Tools for C++ painpoints

#2 Build times

← CMake configuration profiling
Tools for C++ painpoints

- #3 CI pipelines
- test frameworks
- CI integrations
- variety of toolchains
- visual tools
- IDEs integrations
Tools for C++ painpoints

#4 Managing CMake

A Motivating Example (Poll #2)

What will be printed during the configuration phase?

```cpp
set(foo ON)
message(foo)
message("Foo")
message($<foo>)
if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if($<foo>)
  message("#3")
endif()
```
Tools for C++ painpoints

- #4 Managing CMake
  - CMake as a language
  - Debugging CMake
  - Profiling CMake
  - CMake Presets
  - CMake File API
  - Help from AI
Tools for C++ painpoints

#4 Managing CMake
- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI
Tools for C++ painpoints

- #4 Managing CMake
  - CMake as a language
  - Debugging CMake
  - Profiling CMake
  - CMake Presets
  - CMake File API
  - **Help from AI**
Tools for C++ painpoints

#6 Setting up dev env
- start instantly
- onboard easily
- zero configuration time
- local & remote
- reproducible dev artefacts
Tools for C++ painpoints

#8 Thread, memory, and type safety

Painkillers
- Data Flow Static Analysis
- Sanitizers
- Valgrind
Tools for C++ painpoints

DFA: lifetime safety

```cpp
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
Tools for C++ painpoints

- DFA: lifetime safety

```cpp
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.

```cpp
const char *ptr = view.begin()
```
Tools for C++ painpoints

DFA: bounds safety

```cpp
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

58
Tools for C++ painpoints

DFA: Use-after-delete/free

```cpp
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

```cpp
int *int_ptr = new int
```
Tools for C++ painpoints

DFA: NULL dereferencing

```cpp
class Deref {
    int* foo() {
        return nullptr;
    }

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

Pointer may be null

```cpp
int *Deref::buffer = foo()
```
Tools for C++ painpoints

Which of these do you find frustrating about C++ dev?

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

Makefiles

MSBuild

Debug

C++ modernisation
Tools for C++ painpoints

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

Story #2

“Your data w/o your tools”
Your data w/o your tools

ide1: BM-DMA at 0xc008-0xc00f, BIOS settings: hda:pio, hdd:pio
ne2k-pci.c:v1.03 9/22/2003 D. Becker/P. Gortmaker
http://www.scylld.com/network/ne2k-pci.html
hda: QEMU HARDDISK, ATA DISK drive
ide0 at 0x1f0-0x1f7, 0x3f6 on irq 14
hdc: QEMU CD-ROM, ATAPI CD/DVD-ROM drive
ide1 at 0x170-0x177, 0x376 on irq 15
ACPI: PCI Interrupt Link [LMXC] enabled at IRQ 10
ACPI: PCI Interrupt 0000:00:03.0[A] -> Link [LMXC] -> GSI 10 (level, low) -> IRQ 10
eth0: RealTek RTL-8029 found at 0xc100, IRQ 10, 52:54:00:12:34:56.
hda: max request size: 512KiB
hda: 180224 sectors (92 MB) w/256KiB Cache, CHS=178/255/63, (U)DMA
hda: set_multmode: status=0x41 { DriveReady Error }
hda: set_multmode: error=0x04 { DriveStatusError }
ide: failed opcode was: 0xef
hda: cache flushes supported
hda: hdal
hdc: ATAPI 4X CD-ROM drive, 512kB Cache, (U)DMA
Uniform CD-ROM driver Revision: 3.20
Done.
Begin: Mounting root file system...

/init: /init: 151: Syntax error: 0xforce=panic
Kernel panic - not syncing: Attempted to kill init!
C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
Tools and language evolution

SG15, Tooling

Topics related to creation of developer tools for standard C++, including but not limited to modules and package management.
Tools and language evolution

Many new C++ language features now start with Clang-based implementation
Tools and language evolution

---

**Modules (since C++20)**

Most C++ projects use multiple translation units, and so they need to share declarations and definitions across those units. The usage of headers is prominent for this purpose, an example being the standard library whose declarations can be provided by including the corresponding header.

Modules are a language feature to share declarations and definitions across translation units. They are an alternative to some use cases of headers. Modules are orthogonal to namespaces.

```cpp
// helloworld.cpp
export module helloworld; // module declaration
import <iostream>;       // import declaration
export void hello()      // export declaration
{
    std::cout << "Hello world!\n";
}

// main.cpp
import helloworld;       // import declaration
int main()
{
    hello();
}
```
Tools and language evolution: Modules

CMake + VS generator

```
add_executable(ModuleSample a.ixx main.cpp)
```
Tools and language evolution: Modules

CMake +

GCC/Clang compiler flags

function(add_module name)
  file(MAKE_DIRECTORY ${PREBUILT_MODULE_PATH})
  add_custom_target(${name}.pcm
    COMMAND
    ${CMAKE_CXX_COMPILER}
    -std=gnu++20
    -x c++
    -fmodules
    -c
    ${CMAKE_CURRENT_SOURCE_DIR}/${ARGN}
    -Xclang -emit-module-interface
    -o ${PREBUILT_MODULE_PATH}/${name}.pcm
  )
endfunction()
Tools and language evolution: Modules

Natively in CMake 3.25

(CMake examples)

```cmake
add_executable(simple)
target_sources(simple
  PRIVATE
    main.cxx
  PRIVATE
    FILE_SET CXX_MODULES
    BASE_DIRS
      "$\{CMAKE_CURRENT_SOURCE_DIR\}"
    FILES
      importable.cxx)
target_compile_features(simple PUBLIC cxx_std_20)
add_test(NAME simple COMMAND simple)
```
Tools and language evolution: Modules

CMake 3.26: support for modules was added into file-api
Tools and language evolution: Modules

Is your current project planning to use any of the following C++20 features in the next 12 months?

- Concepts: 48% (2022), 53% (2023)
- Modules: 43% (2022), 49% (2023)
- Coroutines: 32% (2022), 29% (2023)
- Other: 3% (2022), 3% (2023)
- None: 29% (2022), 21% (2023)
Tools and language evolution: Contracts

C++26?

Contracts
Tools and language evolution: Reflection

C++26?

Reflection
Tools and language evolution: Reflection

C++26?

Reflection

```cpp
struct TU_Ticket {
    static constexpr int next() {
        // ...
    }
};

void a() {
    TU_Ticket::next();
}

std::integral_constant<int, TU_Ticket::next()> b();

https://lists.isocpp.org/sg7/2024/02/0480.php
Tools and language evolution

P2656

C++ Ecosystem International Standard

Users should be able to mix and match their preferred build systems, compilers, linkers, package managers, static analyzers, runtime analyzers, debuggers, profilers, etc. without needing the tools to have vendor specific knowledge of each other. Vendors should be able to focus on direct tool improvements, rather than figuring out how to interact with yet another proprietary interface.

1. Definitions.
2. Build System ↔ Package Manager Interoperation.
4. Tool introspection.
5. Portable diagnostics format via SARIF.
6. Command line portability.
C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
Era of AI

Writing context-aware code
Era of AI

Improving the code

```cpp
#include <array>
#include <iostream>

int main() {
    const std::array Numbers{1, 3, 5, 6, 7, 9, 10, 11, 13};

    size_t oddNumberCount = 0;
    for (auto &number : const & Numbers) {
        if (number % 2) {
            oddNumberCount++;
        }
    }

    std::cout << oddNumberCount;

    return 0;
}
```
Era of AI

Summarizing for VCS or documentation
Era of AI: real-world examples

How Copilot is being used by the Time Travel Debugging team for repetitive C++ coding © Microsoft

Can I script a Unity Game with AI Assistant in under 15 minutes? © Maarten Balliauw, JetBrains
GDC: Are generative AI tools being used at your company or department?

49% Generative AI tools are currently being used in their company

31% personally use Generative AI tools

21% of AAA developers said their companies have banned the use of such tools. Compared to 9% indie.

*This question was shown only to the developers who chose “None” in the previous question.
C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
9. Questions