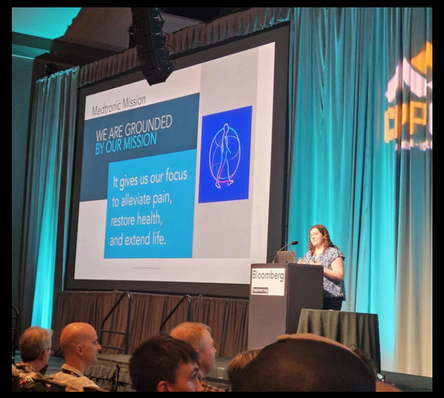
CppCon 2024

Trip Report for 2024 CPPCON



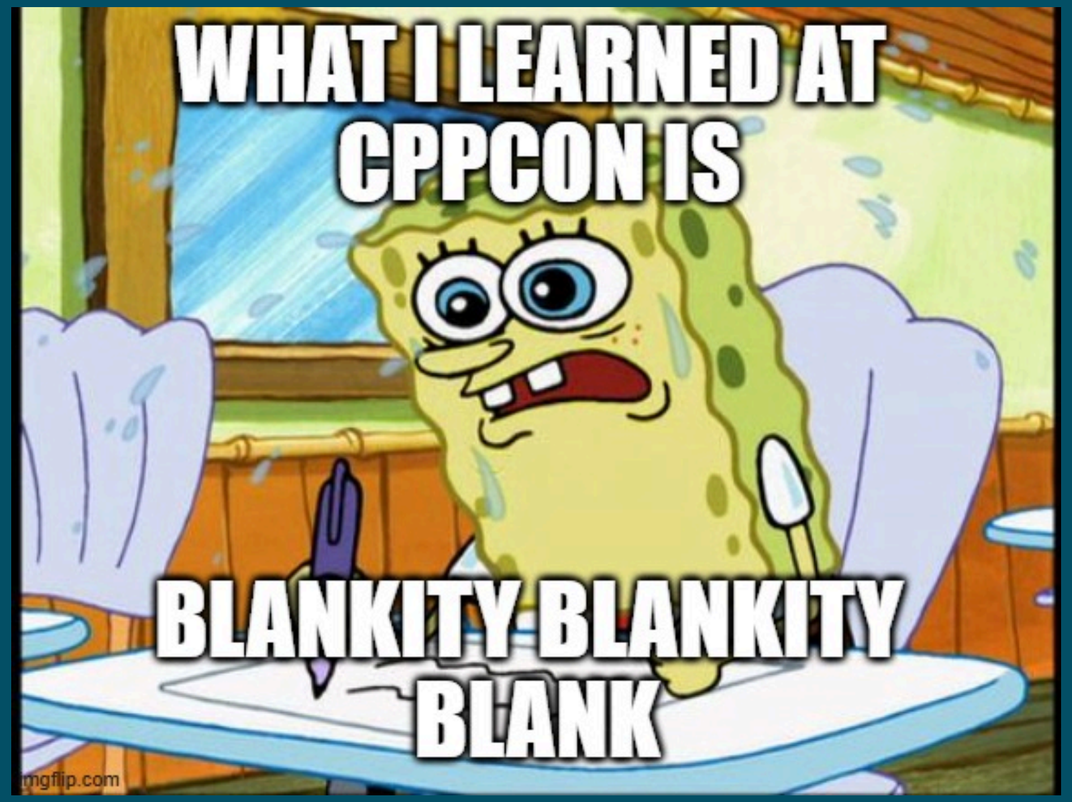
Emily Durie-Johnson
LinkedIn : https://www.linkedin.com/in/emily-e-durie
GitHub: https://github.com/edjineer

November 2024

Medtronic

My Goals

1. Convince you to Consider Going to Future CppCons
2. Direct you to AWESOME Technical Talks
3. Encourage Knowledge Sharing



Agenda

1. CppCon Logistics
2. My CppCon Journey
3. Current State of C++ as a Coding Language
4. Talk Highlights

Image Credit: Sakshi Verma

1

CppCon

LOGISTICS

What is CppCon?

01 5 Full Days

Options to Extend with a CppAcademy Course Before or After

02 Hosted at Gaylord Rockies Convention Center

(PS They have a sweet waterpark)

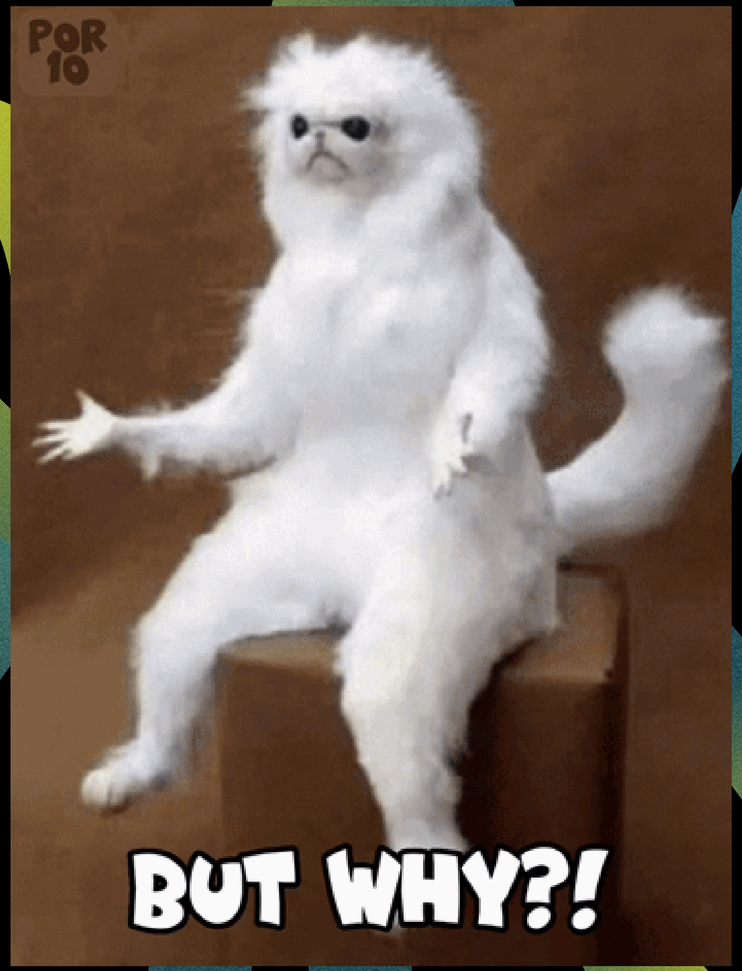
03 Largest Annual Cpp Gathering

700-1000 attendees per year

04 All Talks are on Youtube

Published intermittenly after conference

Why CppCon?



01 Hosted in Aurora, Colorado until 2029

02 Network with Experts, Beginners, and Everyone in between!

03 Great Opportunity to Learn Something New!

04 Presenters attend for Free!

05 FRIDAY IS FREE TO ANYONE

Sessions at 2024 Conference

Technical Tracks

- Robotics
- Embedded
- GameDev
- Tooling
- SW Design

Networking Events

- Career Fair
- Field Trip
- Community Dinner
- Womens' Networking Lunch

Other

- Poster Sessions
- CppCon Academy
- Fireside Chats
- Lightning Talks


2025
September 13-19
Auora, Colorado, USA

2

My CppCon
JOURNEY

Volunteering

Responsibilities

5+ Hour Shift Per Day

Answer Attendee Questions

Collect and Set Up Food

Registration Table

Code of Conduct Training

Sell Merch at the Store

Set-Up and Tear Down

Facilitator for Talks

Recruit Testimonials

General Support

Perks!

Free Registration

Free Community Dinner

Free Food

Free Gifts

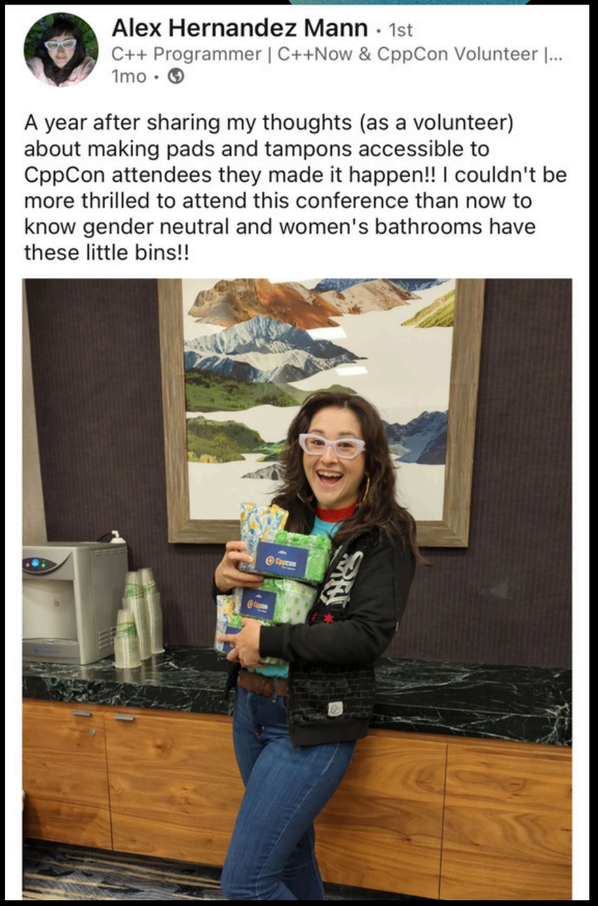
Free Access to Early Recordings

Talking Points

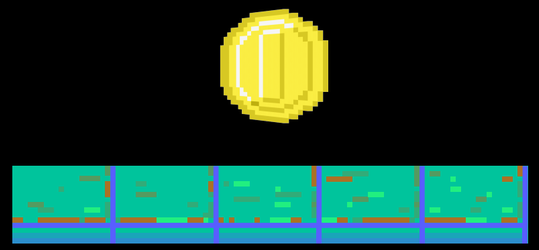
Discount on CppAcademy Courses

2024 Volunteers

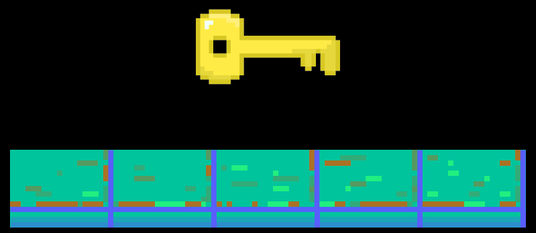
Image Credit: Jonathon Phillips



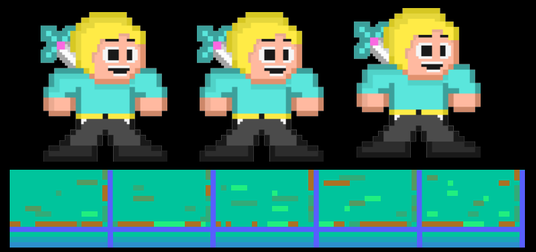
Tailoring CppCon to Your Stage of Growth



CHECK OUT THE CPPCON YOUTUBE

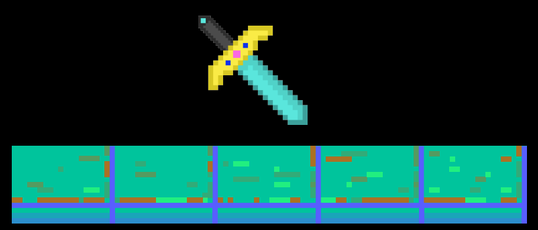


BACK TO BASICS TRACK AND KEYNOTES



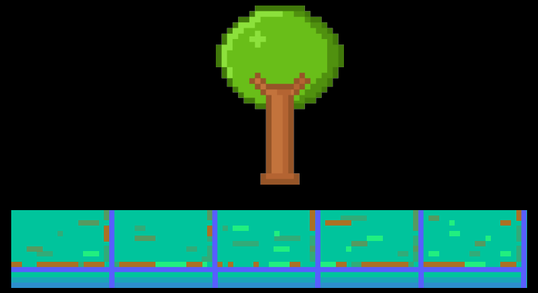
CONNECT WITH COMMUNITY

- “HALLWAY TRACK”
- DENVER C++ MEETUPS
- REDDIT
- PODCASTS



EXPERIMENT

ACCEPT THAT YOU DON’T NEED TO UNDERSTAND EVERYTHING



BE PATIENT WITH YOURSELF

3

Current State of

C++

Evolution of C++ Standards

1979 — “C with Classes”

- Created by Bjarne Stroustrup
- Released publicly and Renamed to C++ in 1984

1991 — C++ 2nd Edition

- RAII added
- Interfaces
- Multiple Inheritance

1997 — C++ 3rd Edition

- ISO C++ Standardization Efforts
- namespaces
- STL

2011 — C++ 11

- Move Semantics
- Lambdas
- STL adds threads and locks
- Uniform Initialization
- auto
- constexpr

2014 — C++ 14

- Core Guidelines released in 2015
- Generic Lambda Fcns

2017 — C++ 17

- Fold Expressions
- File System Library
- Variant and Optional Types

2020 — C++ 20

- Three Way Comparison Operator
- consteval and constinit
- Coroutines
- Modules

2024 — C++ 23

- “Deducing this”
- Extended Floating Point Types
- New Preprocessor Directives

2026 — C++ 26?

- First steps to Reflection and more safety features

References: “Tour of C++ “ 3rd Edition by Bjarne Stroustroup, and https://en.cppreference.com/w/cpp/links

C++ 23

- Proposal Finalized in early 2023
- ISO Standard Updated Oct 2024

- New Features!
  - Full List at CppReference

ISO/IEC 14882:2024

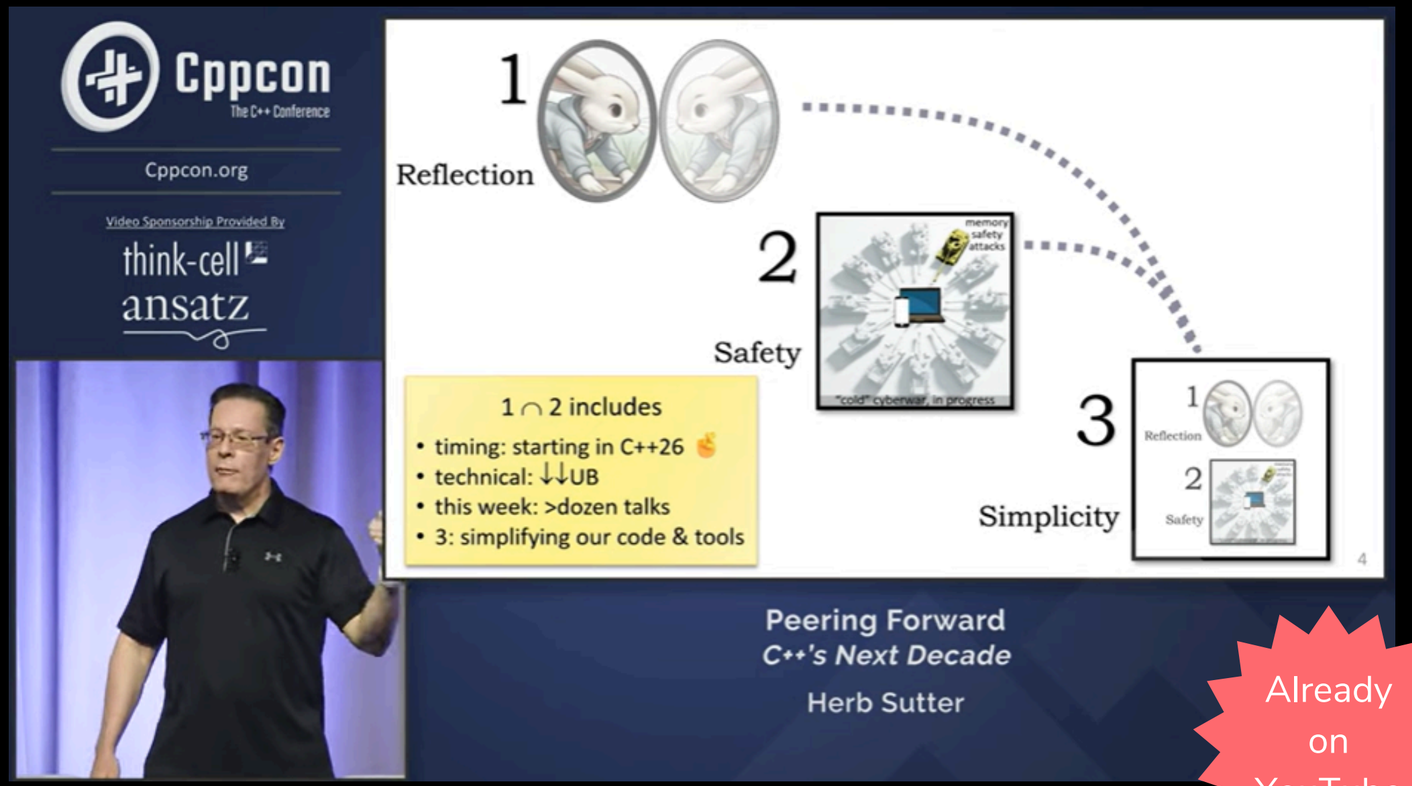
Programming languages — C++

Published (Edition 7, 2024)

C++23 feature	Paper(s)	GCC	Clang	MSVC
Literal suffix for (signed) `size_t`	P0330R8	11	13	
Make `()` more optional for lambdas	P1102R2	11	13	
`if consteval`	P1938R3	12	14	
Removing Garbage Collection Support	P2186R2	12	N/A	19.30*
Narrowing contextual conversions in `static_assert` and constexpr if	P1401R5	9	13 (partial)* 14	
Trimming whitespaces before line splicing	P2223R2	Yes	Yes	
Make declaration order layout mandated	P1847R4	Yes	Yes	Yes
Removing mixed wide string literal concatenation	P2201R1	Yes	Yes	Yes

C++ 26… and beyond!

- Code Freeze for Cpp 2026 planned in 2025
- New Features:
  - Reflection
  - Memory Safety
    - Safety Profiles
    - Contracts



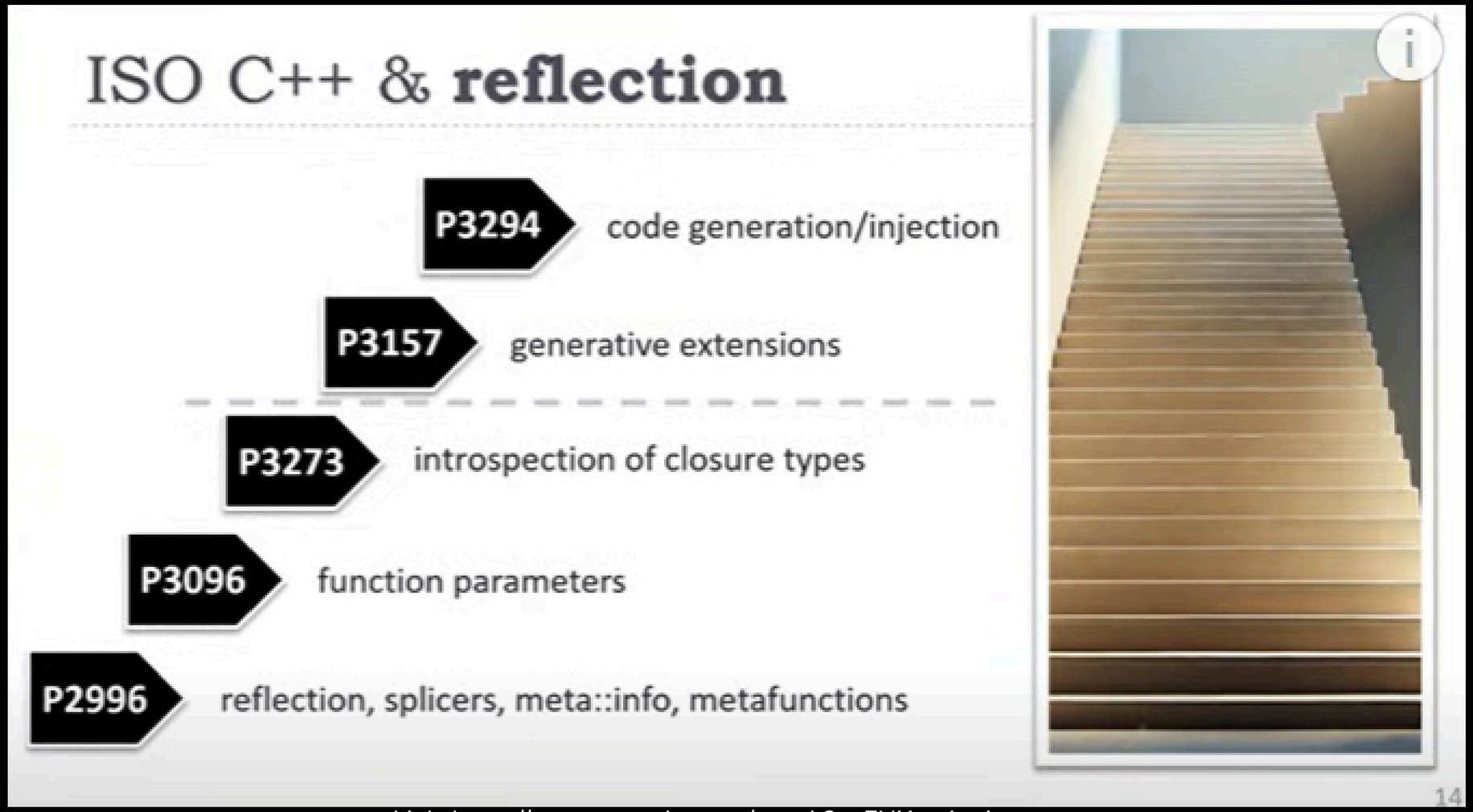
Link: https://www.youtube.com/watch?v=FNi1-x4pojs

Already on YouTube

Status of C++26 Proposals
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/

Reflection

Proposals

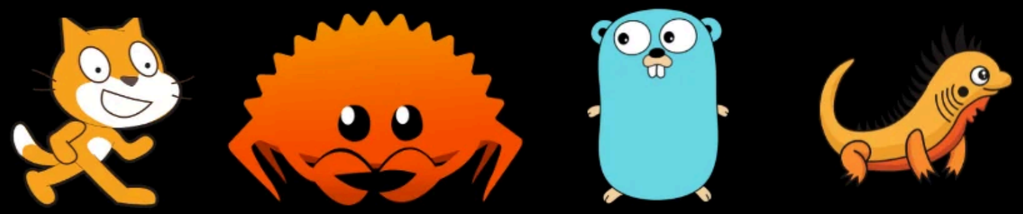


Link: https://www.youtube.com/watch?v=FNi1-x4pojs

Memory Safety

On Track for C++ 26

Some Languages Likes to have cute mascots



Meanwhile C++ ...

The official mascot for C++ is an obese, diseased rat named Keith, whose hind leg is missing because it was blown off. The above image is a contemporary version drawn by Richard Stallman.

Link to Image Credit

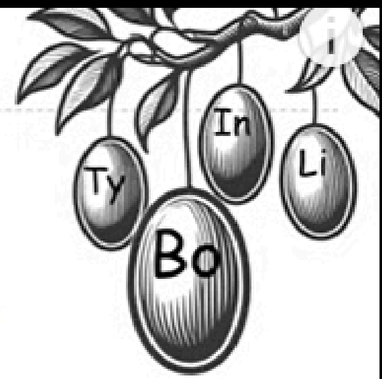
Memory Safety

On Track for C++ 26

tl;dr

The actual problem: Safety parity (not perfection)

- 4 low-hanging priority targets:
  - type, **bounds**, initialization & lifetime safety
- == 4 most severe CWE categories, 4 that all MSLs do better at



Progress

- Key **safe libraries** moving from 3rd-party to standard
  (e.g., C++20 `std::span<T>` to replace pointer math)
- Safety-related **undefined behavior** being removed
  (e.g., Mar 2024: reading uninit stack vars not UB!)
- Key **static safety rules Profiles:** most known, "shift-left" to compile time
- Add **dynamic safety checks** as needed (e.g., bounds, null)

Link: https://www.youtube.com/watch?v=FNi1-x4pojs

"watch out!"
Be sure to remember to code safely

→

"opt out"
of safety guarantees when you REALLY want to

Contracts for Cpp

Timur Doulmer

Not Released Yet

Link to ABSTRACT

- Enforce Preconditions and Postconditions

Option 1
Comments!

```
// Returns a reference to the element at position `index`.
// The behaviour is undefined unless `index < size()`.
T& operator[] (size_t index) {
  return _data[index];
}
```

Option 2
Assert

```
#include <cassert>

T& operator[] (size_t index) {
  assert (index < size());
  return _data[index];
}
```

CppCast

Contracts for Cpp

Timur Doulmer

Link to ABSTRACT

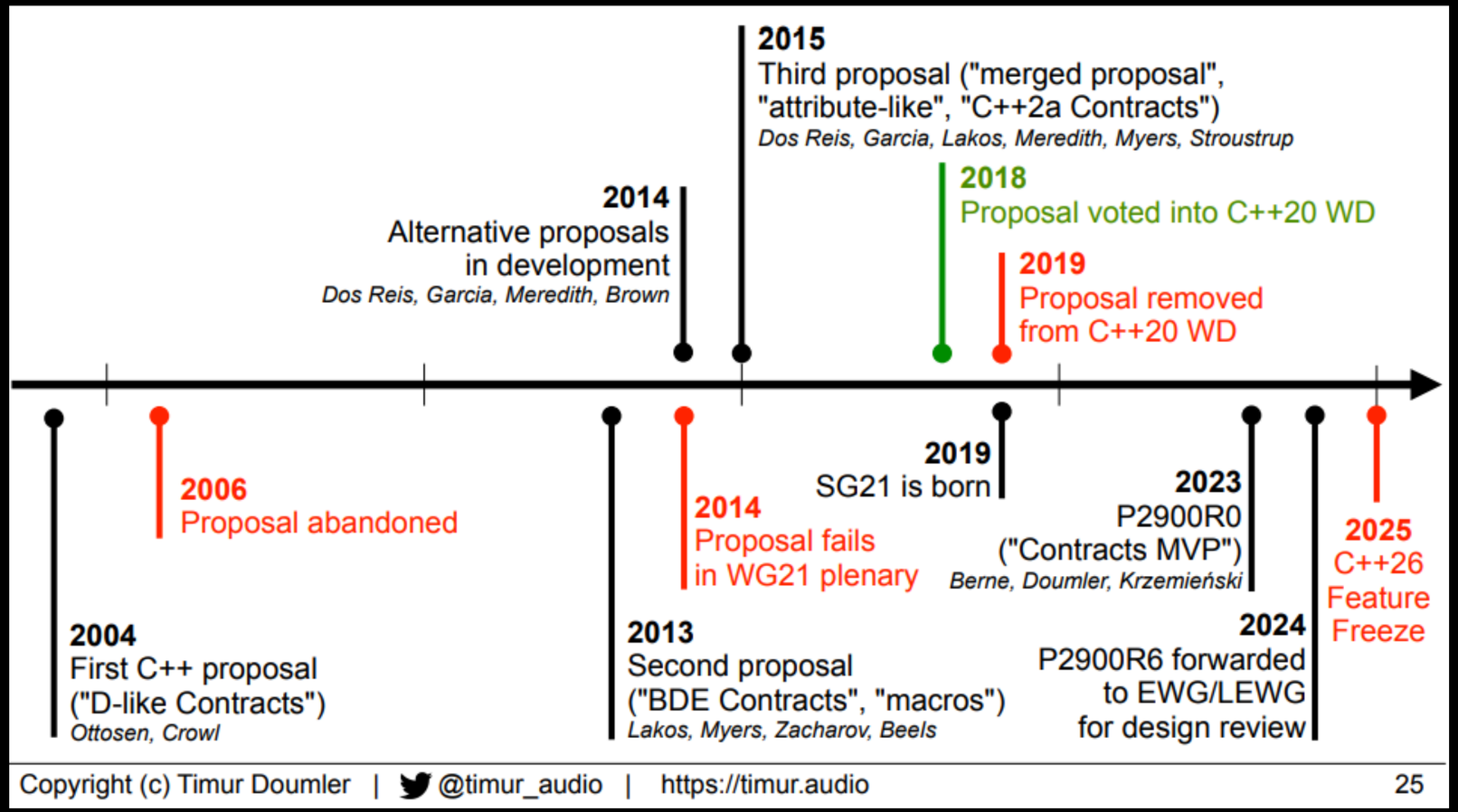
- Using Contracts
  - Post can reference a return value
    - “`post (r : r.isValid())`”
- When Contract Check Fails:
  - Calls a Contract-Violation Handler
    - Default = Log to stdout and terminate
    - Customizable!

```
// vector.h
T& operator[] (size_t i)
  pre (i < size());

void resize(size_t n)
  post (size() == n);

// implementation:
T& operator[] (size_t i)
pre (i < size())  // optional here!
{
  return _data[i];
}
```

Contracts for Cpp

Timur Doulmer

Link to ABSTRACT



Contracts for Cpp

Timur Doulmer

Link to ABSTRACT

- WG21 Identified 196 Use Cases for Contracts

4

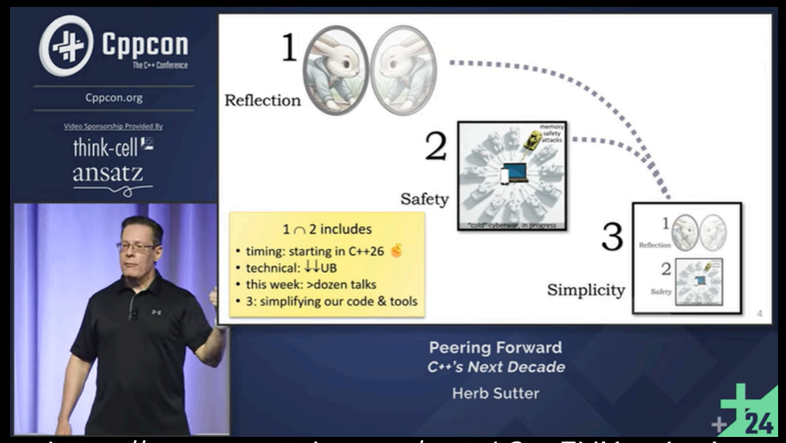
Session HIGHLIGHTS

Link to Presenter Slides

Keynotes

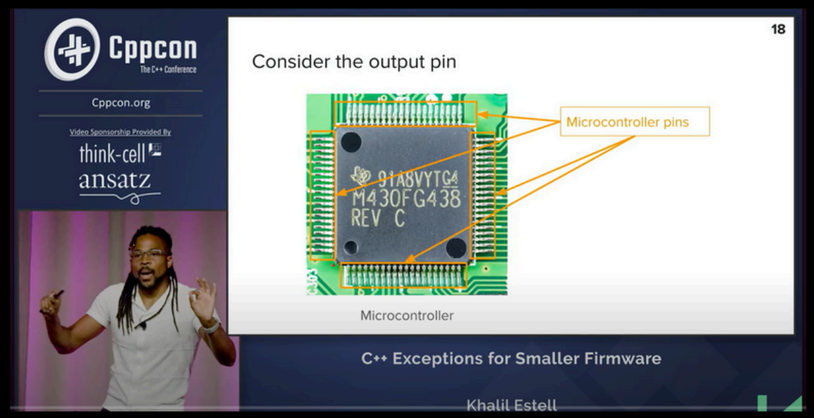
Already Available

Peering Forward: C++'s Next Decade
Herb Sutter



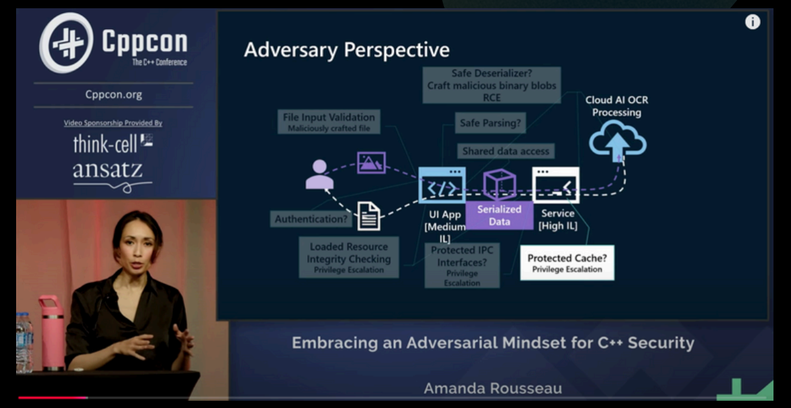
https://www.youtube.com/watch?v=FNi1-x4pojs

C++ Exceptions for Smaller Firmware
Khallil Estell



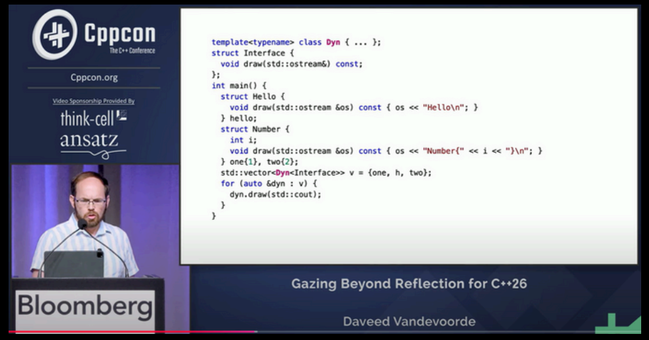
https://www.youtube.com/watch?v=bY2FlayomIE

Embracing an Adversarial Mindset for C++ Security
Amanda Rosseau



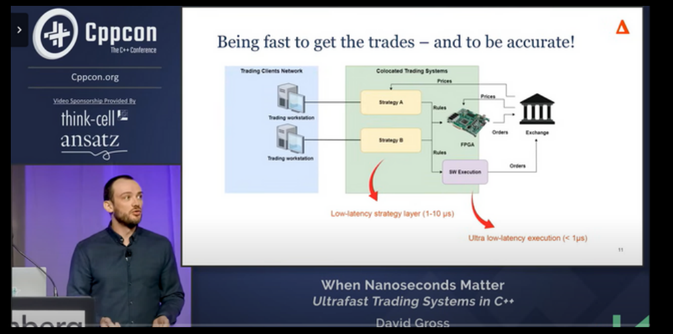
https://www.youtube.com/watch?v=gIkMbNLogZE

Gazing Beyond Reflection for C++26
Daveed Vandevoorde



https://www.youtube.com/watch?v=wpjiowJW2ks

When Nanoseconds Matter: Ultrafast Trading Systems in C++
David Gross



https://www.youtube.com/watch?v=sX2nF1fW7k

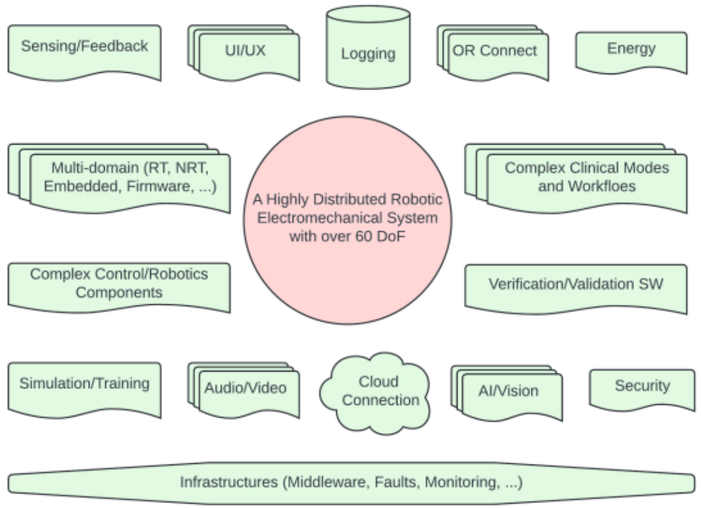
Building Safe and Reliable Surgical Robotics with C++

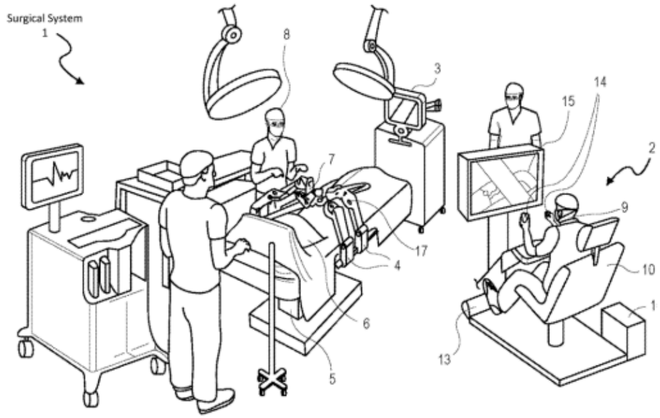
Not Released Yet

Milaud Khaledyan and Alexander Drew
Johnson&Johnson

Robotically Assisted Surgical Platform

Our use case: A Class-C Medical Device





Millions of Lines of C++ Code

J&J MedTech

Link to Slides

Building Safe and Reliable Surgical Robotics with C++

- EXCELLENT Overview for Medical Device SW
  - Concise Intro to Relevant Standards
  - Coding Standards Overview and Examples
- Using C++ in Real Time Setting
  - Avoid Dynamic Memory Allocation
  - Blocking Calls
  - Use alternatives to Exceptions

Link to Slides

Building Safe and Reliable Surgical Robotics with C++

Not Released Yet

Link to Slides

CI/CT/CD

Prefer pre-merge, pre-commit, and local dev before CI: Limited CD in Medical Devices

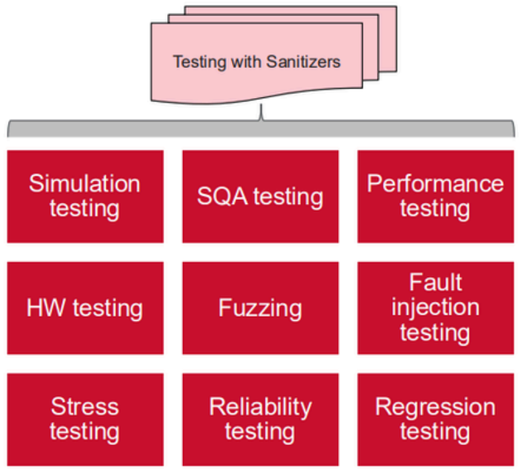
Rich IDE | Use of AI tools | Static analyzers/QG | Sanitizers

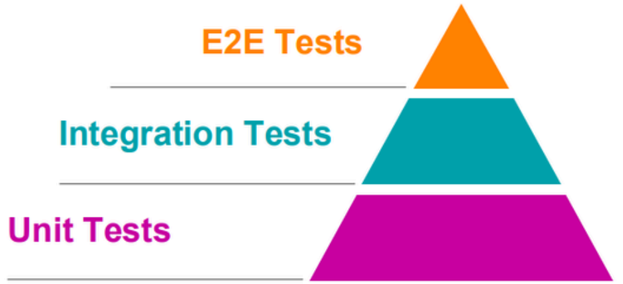
Local Dev → Pre-Commit Hooks → Multi-platform Build → Multi-compiler Build → Test Build → ASan → TSan → LSan → UBSan → MSan → RTSan → Quality Gate → Code Review → Merge

- No new bugs are introduced
- No new vulnerabilities are introduced
- All new security hotspots are reviewed
- New code has limited technical debt
- New code has limited duplication
- New code is properly covered by tests

Continuous Testing

Shift left: Stress test the SW





Safety and Security Panel

Not Released Yet

America's Cyber Defense Agency

NATIONAL COORDINATOR FOR CRITICAL INFRASTRUCTURE SECURITY AND RESILIENCE

Development in Memory Unsafe Languages (CWE[1]-119 and related weaknesses)

The development of new product lines for use in service of critical infrastructure or NCFs in a memory-unsafe language (e.g., C or C++) where there are readily available alternative memory-safe languages that could be used is dangerous and significantly elevates risk to national security, national economic security, and national public health and safety.

Link: https://www.cisa.gov/resources-tools/resources/product-security-bad-practices

A Path Toward Secure and Measurable Software

FEBRUARY 2024

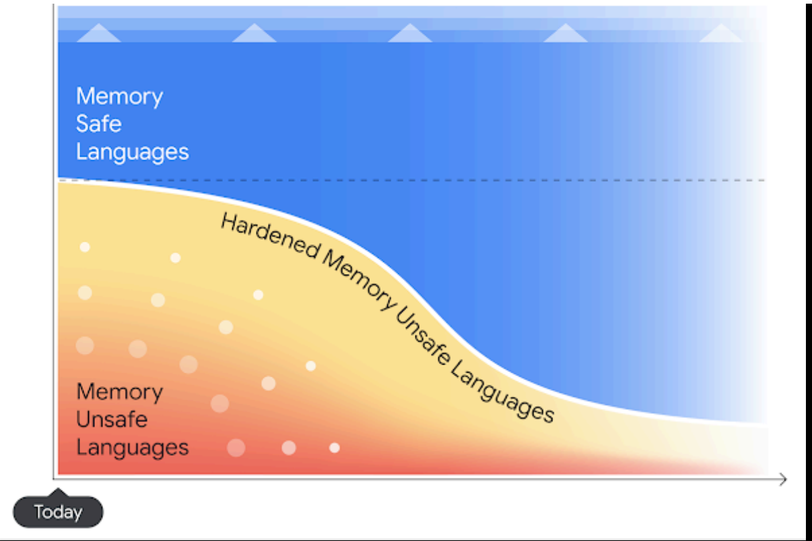
Memory safety vulnerabilities are a class of vulnerability affecting how memory can be accessed, written, allocated, or deallocated in unintended ways.[iii] Experts have identified a few programming languages that both lack traits associated with memory safety and also have high proliferation across critical systems, such as C and C++.[iv] Choosing to use memory safe programming languages at the outset, as recommended by the Cybersecurity and Infrastructure Security Agency's (CISA) Open-Source Software Security Roadmap is one example of developing software in a secure-by-design manner.[v]

Link: https://www.cisa.gov/resources-tools/resources/product-security-bad-practices

Safer with Google: Advancing Memory Safety

October 15, 2024

Posted by Alex Rebert, Security Foundations, and Chandler Carruth, Jen Engel, Andy Qin, Core Developers



Link: https://security.googleblog.com/2024/10/safer-with-google-advancing-memory.html

Safety and Security Panel

Michael Wong, Andreas Weis, Gabriel Dos Reis, Herb Sutter, Lisa Lippincott, Timur Doumler

Not Released Yet

- How does C++ compare to Rust (today, and the future?)
  - Sean Baxter's Implementation of Borrow-Checking in C++
- Parity vs Perfection related to Memory Safety Goals
- Using C++ In the Automotive Industry
  - MISRA
  - Safety and AI ISO/PAS 8800

Link to ABSTRACT
https://cppcon2024.sched.com/event/1kYBw/safety-and-security-panel

Compile Time Validation

Already Available

Alon Wolf, Medtronic 🎉

- Great Overview of Validation Techniques available
  - Bounds Checking
  - Options for balancing Safety and Flexibility
  - Exploring User Generated Error Messages
  - Stateful Metaprogramming
- Introduced his Compile Time Validation Library (Mitzi)
  - Multiple Control Flows
  - Validate Borrow States

Link to Talk
https://cppcon.programmingarchive.com/video/compile-time-validation-alon-wolf-cppcon-2024/

Newer Isn't Always Better : Investigating Legacy Design Trends and Their Modern Replacements

Kathryn Rocha, Atmos Space. SW "Historian/Geneologist"

Not Released Yet

Investigative Process

Timeline

- When was the original trend introduced?
- When did the trend transition?

Original Trend

- What is the original trend?
- Why is it used?

New Trend

- What is the new trend?
- Why did it replace the original?

Original Code

- What is the original solution?
- How elegant is it?
- What are the problems with it?

New Code

- What is the new solution?
- How elegant is it?
- What are the problems with it?

Analysis

- Pros and Cons of the original trend
- Pros and Cons of the new trend
- Comparison of the trends

Link to Slides

Newer Isn't Always Better : Investigating Legacy Design Trends and Their Modern Replacements

Not Released Yet

- Index For Loops->Range Based For Loops
- Global Variables->Singletons->Dependency Injection
- Virtual Functions->CRTP->Deducing This

Link to Slides

import CMake; //Mastering C++ Modules

Not Released Yet

Bill Hoffman, Kitware

- Modules introduced in C++20 Standard
- CMake Support vs ChatGPT Anecdote
- New Module Support in CMake 3.28
  - CXX_MODULES_BMI - new in version 3.28
  - CXX_SCAN_FOR_MODULES
  - Only works if CMAKE_CXX_STANDARD is 20+
  - Needs a “new enough” compiler

Link to Slides

Irksome C++

Walter E. Brown

Consistency in spelling counts, too

- By convention, the std library uses underscores to set off prepositions that occur within a std library name; *e.g.*:
  - ✓ By: `valueless_by_exception`, `chunk_by_view`.
  - ✓ From: `derived_from`, `constructible_from`, `shared_from_this`.
  - ✓ In/out: `in_out_result`.
  - ✓ Of: `out_of_range`, `alignment_of`, `is_base_of`.
  - ✓ To: `to_underlying`, `to_chars`, `to_string`, `convertible_to`.
  - ✓ With: `common_with`, `swappable_with`, `totally_ordered_with`.
  - ✗ … And then we have `addressof`, `tolower`, `toupper`!
- ✗ Every such inconsistency is irksome: more to learn/remember/teach.

Link to Slides

Not Released Yet

Irksome C++

Walter E. Brown

boolishness ☺

- I dearly wish C++ had never classified `bool` as an integer type.
  - Did anyone ever have a good reason to want the likes of either `true + false` or `42 * true` to be valid expressions?
- Here's some code I recently found in the wild (reformatted to fit here):

```
return dividend / divisor
     + (abs_remainder >= abs_half_divisor
       + (divisor % T{2} || quotient_sign < 0)
       ) * quotient_sign;
```

  - Note the arithmetic conflation of `int`s, `bool`s-as-`int`s, and `int`s-as-`bool`s?
  - How long does it take even a seasoned programmer to grok what's going on?

Link to Slides

Not Released Yet

CppAcademy

Modern C++: When Efficiency Matters
Andreas Fertig

- Format
  - 2 days, 8 hours each
  - Small Class: 6 people
  - Interactive
- Topics
  - Modern C++: Uniform Initialization, noexcept, static vs inline, constexpr, Lambdas, templates, and more!
  - https://cppinsights.io/ and https://godbolt.org/

CppAcademy

CppInsights

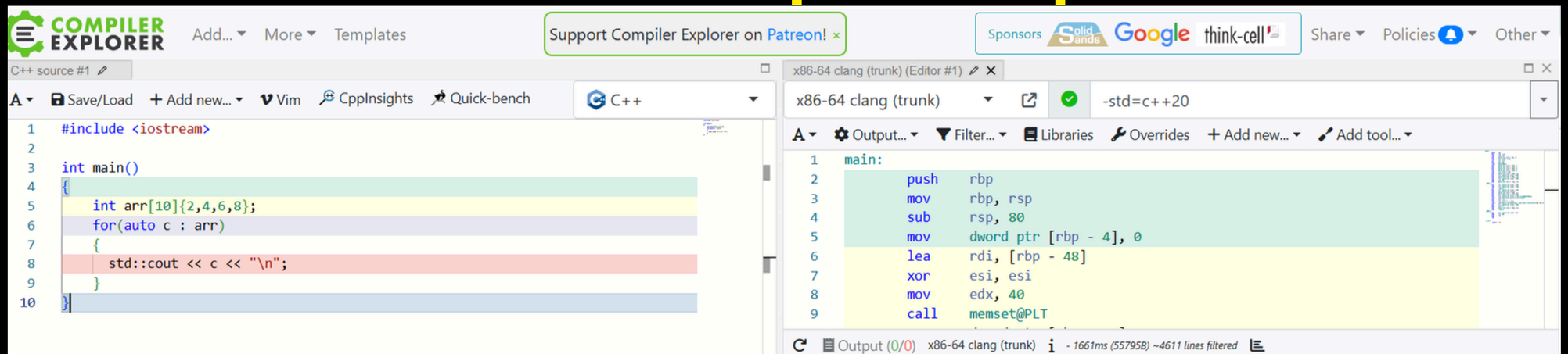
Source:

```cpp
#include <iostream>

int main()
{
    int arr[10]{2,4,6,8};
    for(auto c : arr)
    {
      std::cout << c << "\n";
    }
}
```

Insight:

```cpp
#include <iostream>

int main()
{
  int arr[10] = {2, 4, 6, 8, 0, 0, 0, 0, 0, 0};
  {
    int (&__range1)[10] = arr;
    for(int * __begin1 = __range1, *__end1 = __range1 + 10L; __begin1 != __end1; ++__begin1) {
      int c = *__begin1;
      std::operator<<(std::cout.operator<<(c), "\n");
    }

  }
  return 0;
}
```

Godbolt/Compiler Explorer



Highly Recommended Talks

- Back to Basics: Function Call Resolution by Ben Saks
  - Early Access Link
- Fast and Small C++: When Efficiency Matters by Andreas Fertig
  - Early Access Link
- 10 Problems Large Companies Have With Managing C++ Dependencies and How to Solve them By Augustin Popa
  - Early Access Link
- Implementing Reflection Using the New C++20 Tooling Opportunity: Modules by Maiko Steeman
- The Beman Project: Bringing Standard Libraries to the Next Level by David Sankel
- LLVM's Realtime Safety Revolution: Tools for Modern Mission Critical Systems by David Trevelyan and Chris Apple
  - Early Access Link

References

Slides from All Talks: https://github.com/CppCon/CppCon2024

Other Trip Reports:

- https://devblogs.microsoft.com/cppblog/microsoft-c-team-at-cppcon-2024-trip-report/?ref=dailydev
- https://medium.com/@sakshivrm17/cppcon-2024-trip-report-ba26e6844eb7

General References:

- https://isocpp.org/faq
- https://en.cppreference.com/w/cpp/links

Compiler Support for C++ Versions: https://en.cppreference.com/w/cpp/compiler_support

Proposals: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/

Thank you for listening!

My Notes from the Conference at My Github, Username = **edjineer**