

Safe C++ @ O'Reilly

2. Safe C++

Klaus Iglberger
April, 22nd, 2025

Content

1. Motivation
2. Undefined Behavior
3. Bounds Safety
4. Type Safety
5. Initialization Safety
6. Lifetime Safety
7. The Sharp Edges
8. Summary

2.1. Motivation

C++ has a Problem



C++ has a
Safety Problem





National Security Agency | Cybersecurity Information Sheet

Software Memory Safety

Executive summary

Modern society relies heavily on software-based automation, implicitly trusting developers to write software that operates in the expected way and cannot be compromised for malicious purposes. While developers often perform rigorous testing to prepare the logic in software for surprising conditions, exploitable software vulnerabilities are still frequently based on memory issues. Examples include overflowing a memory buffer and leveraging issues with how software allocates and de-allocates memory. Microsoft® revealed at a conference in 2019 that from 2006 to 2018 70 percent of their vulnerabilities were due to memory safety issues. [1] Google® also found a similar percentage of memory safety vulnerabilities over several years in Chrome®. [2] Malicious cyber actors can exploit these vulnerabilities for remote code

<https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3215760/>



<https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf>

2. Safe C++ - Motivation



EN
English

Search

Shaping Europe's digital future

Home > Library > Cyber Resilience Act

POLICY AND LEGISLATION | Publication 15 September 2022

Cyber Resilience Act

The proposal for a regulation on cybersecurity requirements for products with digital elements, known as the Cyber Resilience Act, bolsters cybersecurity rules to ensure more secure hardware and software products.

<https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act>

MOTHERBOARD
TECH BY VICE

The Internet Has a Huge C/C++ Problem and Developers Don't Want to Deal With It

What do Heartbleed, WannaCry, and million dollar iPhone bugs have in common?

AG

By [Alex Gaynor](#)

November 15, 2018, 1:39pm [Share](#) [Tweet](#) [Snap](#)

```
368 int lcd_create_map_value_to_empty()
369 {
370     memset(empty, 0, 8);
371     int i = 0;
372     int tmp;
```

<https://www.vice.com/en/article/a3mgxb/the-internet-has-a-huge-cc-problem-and-developers-dont-want-to-deal-with-it>

C++ has a
Safety Problem ?



Yes ...

... if you still program in classic C++!



How did we get there?



2. Safe C++ - Motivation



CppNow.org

Video Sponsorship Provided By

millennium
think-cell



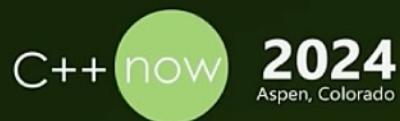
This Is C++

Jon Kalb



<https://www.youtube.com/watch?v=08gyuBC-MIE>

2. Safe C++ - Motivation



CppNow.org

Video Sponsorship Provided By

millennium
think-cell

The diagram consists of two rows of three rounded rectangular boxes each. The top row contains one box labeled 'safe'. The bottom row contains two boxes, the top one labeled 'fast' and the bottom one labeled 'safe'. To the right of this diagram is a large red circular 'prohibited' or 'no' symbol, overlaid on a smaller version of the same two-row diagram where the 'fast' box is at the top and the 'safe' box is at the bottom.

safe

fast

safe

This leads us to this observation that you can build safe on top of fast, but you can't build fast on top of safe. What this means is, that given an API that does no domain or other testing, one can build an interface on top of that API that adds any necessary [...] checking, making it safe.

Jon Kalb



<https://www.youtube.com/watch?v=08gyuBC-MIE>

Was this a bad
Strategy?



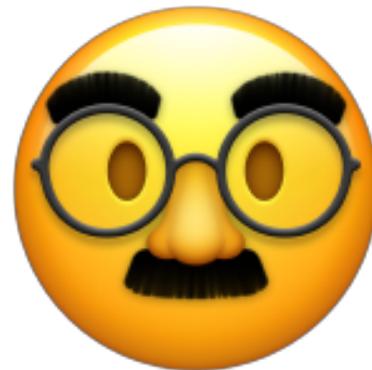
Is C++ Broken?



No!



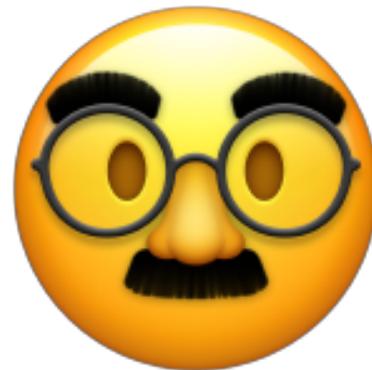
But we have to do our work



Yes, we have to continue to improve the language to make writing safe code simpler.



But there is an even more important task!



We have to communicate
better how to program in C++!



Terminology: Safety vs. Security



"The immediate problem is that it's Too Easy By Default™ to write security and safety vulnerabilities in C++ that would have been caught by stricter enforcement of known rules for Type, Bounds, Initialization, and Lifetime language safety"

(Herb Sutter)

Terminology: Safety vs. Security



"The immediate problem is that it's Too Easy By Default™ to write security and safety vulnerabilities in C++ that would have been caught by stricter enforcement of known rules for Type, Bounds, Initialization, and Lifetime language safety"

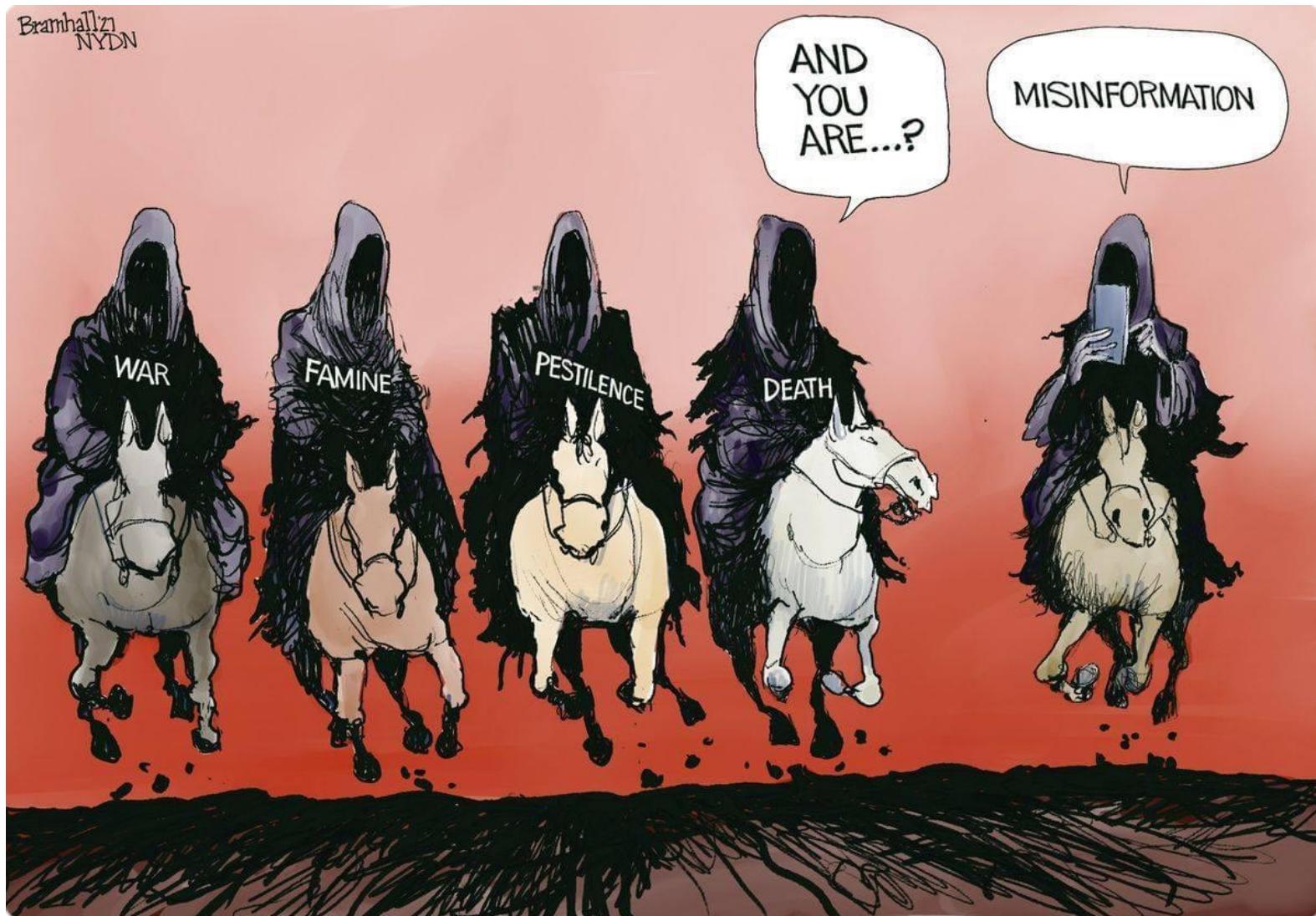
(Herb Sutter)

Bounds Safety Type Safety Initialization Safety Lifetime Safety



Bounds Safety
Type Safety
Initialization Safety
Lifetime Safety
Undefined Behavior

2. Safe C++ - Motivation





Bounds Safety
Type Safety
Initialization Safety
Lifetime Safety
Undefined Behavior

2.2. Undefined Behavior



Programming Task

Task (3_Concepts_and_STL/constexpr/UniquePtr_constexpr): What is wrong with the following given unique_ptr class template? Try to find all flaws and think about how you would test the class to detect these flaws.

```
template< typename T >
class unique_ptr
{
public:
    unique_ptr() {}
    unique_ptr( T* ptr ) : ptr_{ptr} {}

    T& operator*() const { return *ptr_; }
    T* get() const { return ptr_; }

private:
    T* ptr_;
};
```



Pro Tip: Make sure you test
in both `constexpr` and non-
`constexpr` contexts

Copyright Jason Turner

@lefticus

emptycrate.com/idocpp

4

NDC { TechTown }

▶ ⏸ 🔍 7:03 / 59:52

▶ CC HD □ □ □

NDC
Conferences
Subscribe

<https://www.youtube.com/watch?v=MUOAovwQbFA>

2. Safe C++ - Undefined Behavior

A “must-do” since 2017!

cppcon | 2017
THE C++ CONFERENCE • BELLEVUE, WASHINGTON



BEN DEANE / bdeane@blizzard.com / @ben_deane

JASON TURNER / jason@emptycrate.com / @lefticus

CPPCON / MONDAY 25TH SEPTEMBER 2017

1 / 106

- And this is Ben Deane,



constexpr
ALL the Things!

Meeting C++ 2024

think-cell

Woven
by TOYOTA

Adobe



my favorite data structures

{ Hana Dusíková, hana@woven.toyota }

▶ My favorite data structures - Hana Dusíková



<https://www.youtube.com/watch?v=jAIFLEK3jiw>

Programming Task

Task (3_Concepts_and_STL/constexpr/Count): Take a look at the given count() function. What is the expected result for the calls in the main() function?

```
std::uint64_t count( std::uint64_t size )
{
    std::uint64_t count{};

    for( int i=0; size-i >= 0; ++i ) {
        ++count;
    }

    return count;
}
```

Programming Task

Task (3_Concepts_and_STL/constexpr/IntegralShift): Take a look at the given shift() function. What is the expected result for the calls in the main() function?

```
std::uint64_t shift( std::uint64_t count )
{
    return 1 << (count % 64);
}
```

Guidelines

Core Guideline ES.100: Don't mix signed and unsigned arithmetic

Core Guideline ES.101: Use unsigned types for bit manipulation

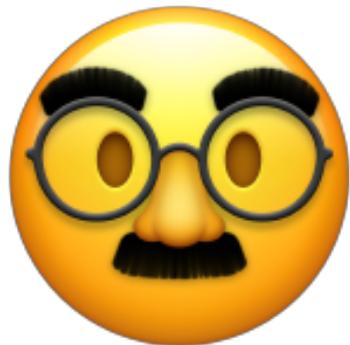
Core Guideline ES.102: Use signed types for arithmetic

2. Safe C++ - Undefined Behavior



I cannot add `constexpr` in front of every function.
So much of our functionality is not `constexpr` yet
and the compiler would constantly complain ...

2. Safe C++ - Undefined Behavior



It's all about the structure of your code. It's all about the design.

Programming Task

Task (3_Concepts_and_STL/constexpr/Ranges_constexpr): What is wrong with the following given `print_countries()` function (think about changes in the input)? Try to find the flaw and think about how you would test the function to detect this flaw.

2. Safe C++ - Undefined Behavior



Brian Ruth

Thinking Functionally in C++

Why are these categories important?

- **Actions**

- Allow input to programs that is unknown when the program was written
- Performing an action has consequences
- Affect how a program executes

- **Calculations**

- Reliable, a calculation always produces the same resulting data when given the same input data.
- Encapsulated, has no effect outside of itself
- Thread safe, since it is entirely self contained, no ordering or locking is necessary

- **Data**

- Fundamental building block
- Immutable, data does not change
- Transparent, you can look at data and see what it is

Video Sponsorship Provided By:

think-cell



▶ ▶ 🔍 8:38 / 50:24 · Identifying Code functionally >

<https://www.youtube.com/watch?v=6-WH9Hnec1M>

2. Safe C++ - Undefined Behavior



Cppcon
The C++ Conference

20
23



October 01 - 06

+

23



Brian Ruth

Thinking Functionally in C++

Isolate the Actions

```
std::vector<Employee> FilterEmployees(const std::vector<Employee> employees,
                                         FilterFunction filter) {
    std::vector<Employee> filteredEmployees;
    std::copy_if(begin(employees), end(employees),
                 std::back_inserter(filteredEmployees), filter);
    return filteredEmployees;
}

bool IsDayInRange(const Day day, const DateRange range) {
    return day >= dateRange.firstDay && day <= dateRange.lastDay;
}

int main() {
    const auto dateRange = GetThisWeekDateRange();
    const auto allEmployees = GetCurrentEmployees();
    const auto birthdayFilter = [dateRange](auto& employee) {
        return IsDayInRange(employee.birthday, dateRange); };

    const auto birthdayEmployees = FilterEmployees(allEmployees, birthdayFilter);
    ...
}
```

Video Sponsorship Provided By:

think-cell

▶ ▶ 🔍 17:23 / 50:24 • Implementation >

▶ CC ⚙ □ □ □

<https://www.youtube.com/watch?v=6-WH9Hnec1M>

Guidelines

Guideline: Use `constexpr` whenever possible.

Guideline: Use `constexpr` in combination with compile time tests to avoid any undefined behavior.

Guideline: Separate your functionality into non-`constexpr` code that doesn't have to be tested (input, output, data, ...) and `constexpr` code that can be tested at compile time.

2.3. Bounds Safety



Programming Task

Task (2_Safe_Cpp/RangesRefactoring_Countries):

Step 1: Understand the code of the `main()` function: what does the final output print?

Step 2: Considering the general case, the initial code contains a bug. For which input does the given solution not work?

Step 3: Improve readability by choosing better names for the variables.

Step 4: Refactor the `main()` function from an imperative to a declarative style by means of C++20 ranges.

Bounds Safety Example

```
struct Country
{
    std::string name{};           // Name of the country
    std::string capital{};        // Name of the capital
    unsigned int area{};          // Area in square kilometer
    float residents{};            // Number of residents in millions
};

struct Continent
{
    std::string name{};           // Name of the continent
    std::vector<Country> countries{}; // Countries of the continent
};

void print_five_biggest_countries()
{
    std::vector<Continent> continents = get_continents_with_countries();

    std::vector<Country> countries{};

    for( auto const& continent : continents ) {
        for( auto const& country : continent.countries ) {
            if( country.residents > 1000000000 )
                countries.push_back( country );
        }
    }

    std::sort( countries.begin(), countries.end(),
              [ ]( Country const& a, Country const& b ) {
                  return a.residents > b.residents;
              } );

    for( auto const& country : countries ) {
        std::cout << country.name << " (" << country.residents << "M) ";
    }
}
```

Bounds Safety Example

```
struct Country
{
    std::string name{};      // Name of the country
    std::string capital{};   // Name of the capital
    unsigned int area{};     // Area in square kilometer
    float residents{};       // Number of residents in millions
};

struct Continent
{
    std::string name{};          // Name of the continent
    std::vector<Country> countries{}; // Countries of the continent
};

void print_five_biggest_countries()
{
    std::vector<Continent> continents = get_continents_with_countries();

    std::vector<Country> countries{};

    for( auto const& continent : continents ) {
        for( auto const& country : continent.countries ) {
            if( country.residents > 1000000000 )
                countries.push_back( country );
        }
    }

    std::sort( countries.begin(), countries.end(),
              [ ]( Country const& a, Country const& b ) {
                  return a.residents > b.residents;
              } );

    for( auto const& country : countries ) {
        if( country.residents > 1000000000 )
            print( country.name );
    }
}
```

Bounds Safety Example

```
struct Country
{
    std::string name{};           // Name of the country
    std::string capital{};        // Name of the capital
    unsigned int area{};          // Area in square kilometer
    float residents{};            // Number of residents in millions
};

struct Continent
{
    std::string name{};           // Name of the continent
    std::vector<Country> countries{}; // Countries of the continent
};

void print_five_biggest_countries()
{
    std::vector<Continent> continents = get_continents_with_countries();

    std::vector<Country> countries{};

    for( auto const& continent : continents ) {
        for( auto const& country : continent.countries ) {
            if( country.residents > countries.back().residents )
                countries.push_back( country );
        }
        if( countries.size() == 5 )
            break;
    }
}
```

Bounds Safety Example

```
void print_five_biggest_countries()
{
    std::vector<Continent> continents = get_continents_with_countries();

    std::vector<Country> countries{};

    for( auto const& continent : continents ) {
        for( auto const& country : continent.countries )
        {
            auto pos = begin(countries);

            while( pos != end(countries) && pos->area > country.area )
                ++pos;

            countries.insert( pos, country );
        }
    }

    std::vector<Country> five_biggest_countries{ begin(countries), begin(countries)+5U };

    for( size_t i=1U; i<5U; ++i )
    {
        auto& country1 = five_biggest_countries[i-1U];
        for( size_t j=i; j<5U; ++j ) {
            auto& country2 = five_biggest_countries[j];
            if( country1.residents < country2.residents ) {
                Country tmp{country1};
                country1 = country2;
                country2 = tmp;
            }
        }
    }
}
```

Bounds Safety Example

```
void print_five_biggest_countries()
{
    std::vector<Continent> continents = get_continents_with_countries();

    std::vector<Country> countries{};

    for( auto const& continent : continents ) {
        for( auto const& country : continent.countries )
        {
            auto pos = begin(countries);

            while( pos != end(countries) && pos->area > country.area )
                ++pos;

            countries.insert( pos, country );
        }
    }

    std::vector<Country> five_biggest_countries{ begin(countries), begin(countries)+5U };

    for( size_t i=1U; i<5U; ++i )
    {
        auto& country1 = five_biggest_countries[i-1U];
        for( size_t j=i; j<5U; ++j ) {
            auto& country2 = five_biggest_countries[j];
            if( country1.residents < country2.residents ) {
                Country tmp{country1};
                country1 = country2;
                country2 = tmp;
            }
        }
    }
}
```

Bounds Safety Example

```
void print_five_biggest_countries()
{
    std::vector<Continent> continents = get_continents_with_countries();

    std::vector<Country> countries{};

    for( auto const& continent : continents ) {
        for( auto const& country : continent.countries )
        {
            auto pos = begin(countries);

            while( pos != end(countries) && pos->area > country.area )
                ++pos;

            countries.insert( pos, country );
        }
    }

    std::vector<Country> five_biggest_countries{ begin(countries), begin(countries)+5U };

    for( size_t i=1U; i<5U; ++i )
    {
        auto& country1 = five_biggest_countries[i-1U];
        for( size_t j=i; j<5U; ++j ) {
            auto& country2 = five_biggest_countries[j];
            if( country1.residents < country2.residents ) {
                Country tmp{country1};
                country1 = country2;
                country2 = tmp;
            }
        }
    }
}
```



Bounds Safety Example

```
    countries.insert( pos, country );
}

std::vector<Country> five_biggest_countries{ begin(countries), begin(countries)+5U };

for( size_t i=1U; i<5U; ++i )
{
    auto& country1 = five_biggest_countries[i-1U];
    for( size_t j=i; j<5U; ++j ) {
        auto& country2 = five_biggest_countries[j];
        if( country1.residents < country2.residents ) {
            Country tmp{ country1 };
            country1 = country2;
            country2 = tmp;
        }
    }
}

for( auto const& country : five_biggest_countries ) {
    std::cout << country << '\n';
}
```



Bounds Safety Example

```
    countries.insert( pos, country );
}

}

std::vector<Country> five_biggest_countries{ begin(countries), begin(countries)+5U };

for( size_t i=1U; i<5U; ++i )
{
    auto& country1 = five_biggest_countries[i-1U];
    for( size_t j=i; j<5U; ++j ) {
        auto& country2 = five_biggest_countries[j];
        if( country1.residents < country2.residents ) {
            Country tmp{ country1 };
            country1 = country2;
            country2 = tmp;
        }
    }
}

for( auto const& country : five_biggest_countries ) {
    std::cout << country << '\n';
}
}
```

Bounds Safety Example

```

        countries.insert( pos, country );
    }
}

std::vector<Country> five_biggest_countries{ begin(countries), begin(countries)+5U };

for( size_t i=1U; i<5U; ++i )
{
    auto& country1 = five_biggest_countries[i-1U];
    for( size_t j=i; j<5U; ++j ) {
        auto& country2 = five_biggest_countries[j];
        if( country1.residents < country2.residents ) {
            Country tmp{ country1 };
            country1 = country2;
            country2 = tmp;
        }
    }
}

for( auto const& country : five_biggest_countries ) {
    std::cout << country << '\n';
}

```



The UB smiley

Bounds Safety Example

```
    countries.insert( pos, country );
}

}

assert( countries.size() >= 5U ); // The minimum number of countries must be 5!

std::vector<Country> five_biggest_countries{ begin(countries), begin(countries)+5U };

for( size_t i=1U; i<5U; ++i )
{
    auto& country1 = five_biggest_countries[i-1U];
    for( size_t j=i; j<5U; ++j ) {
        auto& country2 = five_biggest_countries[j];
        if( country1.residents < country2.residents ) {
            Country tmp{ country1 };
            country1 = country2;
            country2 = tmp;
        }
    }
}

for( auto const& country : five_biggest_countries ) {
    std::cout << country << '\n';
}
```

Programming Task

Task (2_Safe_Cpp/RangesRefactoring_Birthday):

Step 1: Understand the inner workings of the `select_birthday_children()` function: what does the function return?

Step 2: Refactor the function from an imperative to a declarative style by means of C++20 ranges.

Step 3: Compare the runtime performance of both versions (imperative and declarative).

Programming Task

Task (2_Safe_Cpp/RangesRefactoring_Animals):

Step 1: Understand the code of the main() function: what does the final output print?

Step 2: Improve readability by choosing better names for the variables.

Step 3: Refactor the main() function from an imperative to a declarative style by means of C++20 ranges.

Programming Task

Task (2_Safe_Cpp/RangesRefactoring_Recipes):

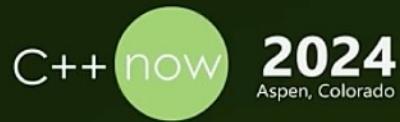
Step 1: Understand the code of the `main()` function: what does the final output print?

Step 2: Refactor the `main()` function from an imperative to a declarative style by means of C++20/23 ranges.

Programming Task

Task (2_Safe_Cpp/Erase): Consider the given `std::vector<int>`. Remove all odd elements from the vector as efficiently as possible.

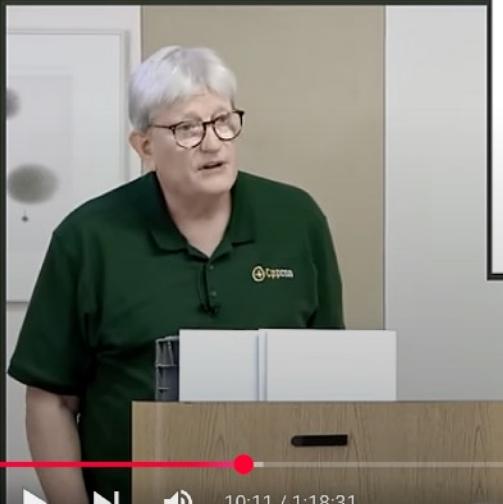
2. Safe C++ - Bounds Safety



CppNow.org

Video Sponsorship Provided By

millennium
think-cell



safe
fast



This Is C++

Jon Kalb



<https://www.youtube.com/watch?v=08gyuBC-MIE>

The Expert's Opinion



"If you want to improve code quality in your organization, I would say, take all your coding guidelines and replace them with the one goal. That's how important I think this one goal is:

No Raw Loops.

This will make the biggest change in code quality within your organization."

(Sean Parent, C++ Seasoning, Going Native 2013)

The Definition of Raw Loops

- A raw loop is any loop inside a function where the function serves purpose larger than the algorithm implemented by the loop.
- Range-based for loops for for-each and simple transforms
 - Use **auto const&** for for-each and **auto&** for transforms

```
for( auto const& elem : range ) f(elem); // for-each  
for( auto& elem : range ) e = f(elem); // simple transform
```

- Keep the body short

```
for( auto const& elem : range ) f(g(elem));  
for( auto const& elem : range ) { f(elem); g(elem); }  
for( auto& elem : range ) e = f(e) + g(e);
```

The Expert's Interpretation of Raw Loops



"9 times out of 10, a for-loop should either be the only code in a function, or the only code in the loop should be a function (or both)."
(Tony Van Eerd, @tvaneerd via Twitter)

The Expert's Opinion On The Cost of Code



"Each line of code costs a little. The more code you write, the higher the cost. The longer a line of code lives, the higher its cost. Clearly, unnecessary code needs to meet a timely demise before it bankrupts us."

(Pete Goodliffe, Becoming a Better Programmer)

The Expert's Opinion On Complexity



"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are by definition, not smart enough to debug it."

(Brian Kernighan)

I cannot use C++20 yet.
Thus I cannot use ranges.



2. Safe C++ - Bounds Safety

THE Ranges Library: ranges_v3

<https://github.com/ericniebler/range-v3>

The screenshot shows the GitHub repository page for 'range-v3' owned by 'ericniebler'. The repository is public and has 154 watchers, 449 forks, and a star rating of 4.2k. It features 38 branches and 22 tags. The 'About' section describes it as a Range library for C++14/17/20, basis for C++20's std::ranges. Key topics include c-plus-plus, proposal, iterator, and range. The 'Releases' section lists 19 releases, with the latest being 'Dude, Where's My Bored Ape?' from June 22, 2022. The 'Packages' section indicates no packages have been published. The 'Contributors' section shows 113 contributors, with 99 more listed below. The main area displays a list of recent commits from 'jicama'.

Commit	Message	Date
jicama	Fix for C++23 implicit move (#1838)	7e6f34b · 6 months ago
.github/workflows	Enable ASan in Windows CI	6 months ago
.vscode	constexpr algorithms (#1683)	3 years ago
cmake	Don't install Range-v3 when it is used as a subproject (#17...)	2 years ago
doc	release 0.12.0	3 years ago
example	add chunk_by which evaluate the predicate on adjacent ele...	4 years ago
include	eliminate apple-clang deprecation warnings about redeclar...	6 months ago
perf	build: range-v3 qualify cmake targets (#1557)	5 years ago
test	Fix for C++23 implicit move (#1838)	6 months ago
test_package	release 0.12.0	3 years ago
.buckconfig	* Added support for the Buck build system	9 years ago
.clang-format	use clang as a doxygen preprocessor and turn requires clau...	5 years ago
.gitattributes	better error messages from action misuses	11 years ago
.gitignore	Supporting Bzlmod	8 months ago
.gitmodules	use https authentication on gh-pages submodule	11 years ago
BUCK	merge from origin/master, fix merge conflicts	7 years ago
BUILD.bazel	Add include/std/* to the list of headers available through ba...	6 years ago
CMakeLists.txt	Don't install Range-v3 when it is used as a subproject (#17...)	2 years ago
LICENSE.md	release 0.12.0	2 years ago

Guidelines

Guideline: Manual loops are the biggest source of bounds safety issues. Using algorithms and C++20/23 ranges avoids this problem.

Guideline: Use algorithms to reduce the complexity of code.

Guideline: “No raw loops” (Sean Parent)

Guideline: Keep your code simple (KISS).

2.4. Type Safety



Guidelines

Core Guideline I.4: Make interfaces precisely and strongly typed

The Expert's Opinion



"Make your interfaces easy to use correctly and hard to use incorrectly."

(Scott Meyers, Effective C++)

Type Safety: An Example

```
void extrapolate_position(  
    double& x, double& y, double t,  
    double vx, double vy,  
    double ax=0.0, double ay=0.0 );  
  
int main()  
{  
    // ...  
    double x = 1.1;  
    double y = -2.3;  
    double time = 2.0;  
  
    extrapolate_position( x, y, time, -0.43, -0.11 );  
  
    extrapolate_position( x, y, -0.43, -0.11, time, 0.01 );  
    // ...  
}
```



It's sooo easy to swap values!



I cannot remember more than 5 parameters.



No info about expected units?



It returns via mutable references?

Type Safety: An Example

```
void extrapolate_position(
    double& x, double& y, double t,
    double vx, double vy,
    double ax=0.0, double ay=0.0 );

int main()
{
    // ...
    double x = 1.1;
    double y = -2.3;
    double time = 2.0;

    extrapolate_position( x, y, time, -0.43, -0.11 );
    extrapolate_position( x, y, -0.43, -0.11, time, 0.01 );
    // ...
}
```



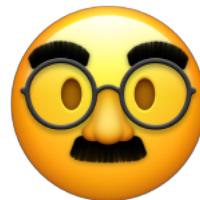
This is not up-to-date C++ ...

Type Safety: An Example

```
void extrapolate_position(
    double& x, double& y, double t,
    double vx, double vy,
    double ax=0.0, double ay=0.0 );

int main()
{
    // ...
    double x = 1.1;
    double y = -2.3;
    double time = 2.0;

    extrapolate_position( x, y, time, -0.43, -0.11 );
    extrapolate_position( x, y, -0.43, -0.11, time, 0.01 );
    // ...
}
```



... and it never was! ...

Type Safety: An Improved Example

```
Position extrapolate_position(  
    Position pos, Second time,  
    MeterPerSecond velocity,  
    MeterPerSquareSecond acceleration={} );  
  
int main()  
{  
    // ...  
    Position position{ 1.1, -2.3 };  
    Second time{ 2.0 };  
    MeterPerSecond velocity{ -0.43, -0.11 };  
  
    extrapolate_position( position, time, velocity );  
  
    extrapolate_position( position, time, velocity, Acceleration{0.01,0.0} );  
    // ...  
}
```



... Since C++98 we utilize strong types.

Guidelines

Core Guideline I.24: Avoid adjacent parameters of the same type when changing the argument order would change meaning

Guideline: Prefer strong types to integral and floating point parameters.



You're Not as Smart as
You Think You Are

- You are not as smart
as you think you are,

Presenter: Phil Nash

Play (k)

▶ ▶ ⏪ 0:00 / 6:52



Magic number 7 $(+/- 2)$

by George Miller, cognitive
psychologist in the '60s.



You're Not as
Smart as You
Think You Are

Guidelines

Core Guideline I.23: Keep the number of function arguments low

Guideline: Prefer a maximum of five function parameters.

Guidelines

Core Guideline I.4: Make interfaces precisely and strongly typed

Solution: Strong Types

Task (2_Safe_Cpp/Meter): Consider the Meter class that serve as a wrapper around integral values.

Step 1: What do we need to improve to really make this a strong type?

Step 2: Define user-defined literals for the Meter type.

Programming Task

Task (2_Safe_Cpp/Meter_Assembly): Compare the assembly output of the two functions `test()` (for instance by means of Compiler Explorer; godbolt.org). Is there any difference/overhead when using the `Meter` class template instead of the fundamental type?

Programming Task

Task (2_Safe_Cpp/StrongType): Refactor the following code by means of strong types and according user-defined literals.

Programming Task

Task (2_Safe_Cpp/StrongType_Assembly): Compare the assembly output of the two functions test() (for instance by means of Compiler Explorer; godbolt.org). Is there any difference/overhead when using the StrongType class template instead of the fundamental type?

But what about performance?
Aren't abstractions
slow?



2. Safe C++ - Type Safety

The screenshot shows the Compiler Explorer interface with two tabs open: 'C++ source #1' and 'x86-64 gcc 14.2 (Editor #1)'. The left tab displays the following C++ code:

```
147
148 //---- <Main.cpp.h> -----
149
150 //#include <StrongType.h>
151 #include <cstdlib>
152 #include <iostream>
153
154 using Second = StrongType<int,struct SecondTag>
155
156 int test( int second )
157 {
158     return second+7;
159 }
160
161 Second test( Second second )
162 {
163     return Second{ second.get()+7 };
164 }
165
166
```

The right tab shows the assembly output for the 'test' function:

```
x86-64 gcc 14.2
x86-64 gcc 14.2
A 1
A 2
A 3
A 4
A 5
A 6
```

The assembly output is mostly empty, with only the first few assembly lines visible. The status bar at the bottom indicates: Output (0/0) x86-64 gcc 14.2 - 9827ms (84490B) ~6031 lines filtered.



Add... More Templates

C++ source #1

A

C++

```
147
148 //---- <Main.cpp.h> -----
149
150 //#include <StrongType.h>
151 #include <cstdlib>
152 #include <iostream>
153
154 using Second = StrongType<int,struct SecondTag>
155
156 int test( int second )
157 {
158     return second+7;
159 }
160
161 Second test( Second second )
162 {
163     return Second{ second.get()+7 };
164 }
165
166
```

intel Solid Sands

think-cell

Share Policies Other

x86-64 gcc 14.2 (Editor #1)

x86-64 gcc 14.2

-std=c++20 -O3 -V

A

```
1 test(int):
2     lea    eax, [rdi+7]
3     ret
4 test(StrongType<int, SecondTag>):
5     lea    eax, [rdi+7]
6     ret
```

C Output (0/0) x86-64 gcc 14.2 i - 9827ms (84490B) ~6031 lines filtered

2. Safe C++ - Type Safety



CppNow.org

Video Sponsorship Provided By

millennium
think-cell

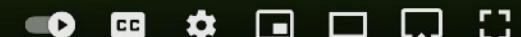


safe
fast



This Is C++

Jon Kalb



Do I have to write all of
this myself?



std::chrono Library

cppreference.com Create account Search

Page Discussion Standard revision: Diff View Edit History

C++ Date and time library

Date and time library

C++ includes support for two types of time manipulation:

- The [chrono library](#), a flexible collection of types that track time with varying degrees of precision (e.g., `std::chrono::time_point`).
- C-style date and time library (e.g., `std::time`).

Chrono library (since C++11)

The chrono library defines several main types as well as utility functions and common typedefs:

- [clocks](#)
- [time points](#)
- [durations](#)
- [calendar dates](#) (since C++20)
- [time zone information](#) (since C++20)

Clocks

A clock consists of a starting point (or epoch) and a tick rate. For example, a clock may have an epoch of January 1, 1970 and tick every second. C++ defines several clock types:

Defined in header `<chrono>`
Defined in namespace `std::chrono`

system_clock (C++11)	wall clock time from the system-wide realtime clock (class)
steady_clock (C++11)	monotonic clock that will never be adjusted (class)
high_resolution_clock (C++11)	the clock with the shortest tick period available (class)

<https://en.cppreference.com/w/cpp/chrono>

2. Safe C++ - Type Safety

Generic Strong Type Library 1

The screenshot shows the GitHub repository page for `rollbear/strong_type`. The repository is public and has 2 branches and 15 tags. It contains 277 commits from the author `rollbear`. The repository uses the Boost Software License 1.0. It has 420 stars, 33 forks, and 15 watchers. The repository was created on Jul 7, 2017, and last updated on Jul 7, 2020. The README file describes it as an additive strong typedef library for C++14/17/20 using the Boost Software License 1.0. It includes CI status, codecov coverage, and a 'Buy me a coffee' button. The repository has 14 contributors and supports C++ and CMake.

https://github.com/rollbear/strong_type

2. Safe C++ - Type Safety

Generic Strong Type Library 2

The screenshot shows the GitHub repository page for `foonathan/type_safe`. The repository is public and has 410 commits. The main branch is `main`, which has 2 branches and 6 tags. The repository has 1.5k stars, 118 forks, and 64 watchers. It uses C++ and Python languages. The repository is zero overhead utilities for preventing bugs at compile time. It includes releases such as v0.2.4 (Latest) and activity sections like Readme, MIT license, and Activity.

https://github.com/foonathan/type_safe

2. Safe C++ - Type Safety

Generic Strong Type Library 3

The screenshot shows the GitHub repository page for `joboccara/NamedType`. The repository has 26 issues, 8 pull requests, and 204 commits. It includes sections for About, Releases, Packages, Contributors, and Languages.

About

Implementation of strong types in C++

- Readme
- MIT license
- Activity
- 778 stars
- 43 watching
- 85 forks

Report repository

Releases 1

`NamedType` (Latest) on Feb 8, 2018

Packages

No packages published

Contributors 19

+ 5 contributors

Languages

C++ 99.4% CMake 0.6%

Basic usage

It central piece is the templated class `NamedType`, which can be used to declare a strong type with a `typedef`-like

<https://github.com/joboccara/NamedType>

Guidelines

Guideline: Prefer strong types to integral and floating point types.

2.5. Initialization Safety



Initialization of Variables

```
int i;
int array[100];
std::string s;
int* pi;

int func()
{
    int i;
    int array[100];
    std::string s;
    int* pi;

    // ...
}

struct Widget
{
    int i;
    int array[100];
    std::string s;
    int* pi;
};
```

Initialization of Static Variables

```
int i;                      // Initialized to 0
int array[100];              // Initialized to all 0s
std::string s;                // Initialized to an empty string
int* pi;                     // Initialized to nullptr

int func()
{
    int i;
    int array[100];
    std::string s;
    int* pi;

    // ...
}

struct Widget
{
    int i;
    int array[100];
    std::string s;
    int* pi;
};

};
```

Initialization of Local Variables

```
int i;                      // Initialized to 0
int array[100];              // Initialized to all 0s
std::string s;                // Initialized to an empty string
int* pi;                     // Initialized to nullptr

int func()
{
    int i;                  // Uninitialized
    int array[100];          // Uninitialized
    std::string s;            // Initialized to an empty string
    int* pi;                 // Uninitialized

    // ...
}

struct Widget
{
    int i;
    int array[100];
    std::string s;
    int* pi;
};
```

Initialization of Member Variables

```
int i;                      // Initialized to 0
int array[100];              // Initialized to all 0s
std::string s;                // Initialized to an empty string
int* pi;                     // Initialized to nullptr

int func()
{
    int i;                  // Uninitialized
    int array[100];          // Uninitialized
    std::string s;            // Initialized to an empty string
    int* pi;                 // Uninitialized

    // ...
}

struct Widget
{
    int i;                  // Uninitialised
    int array[100];          // Uninitialized
    std::string s;            // Initialized to an empty string
    int* pi;                 // Uninitialized
};
```

Value Initialization

```
int i{};                      // Initialized to 0
int array[100]{};              // Initialized to all 0s
std::string s{};                // Initialized to an empty string
int* pi{};                     // Initialized to nullptr

int func()
{
    int i{};                   // Initialized to 0
    int array[100]{};          // Initialized to all 0s
    std::string s{};            // Initialized to an empty string
    int* pi{};                  // Initialized to nullptr

    // ...
}

struct Widget
{
    int i{};                   // Initialized to 0
    int array[100]{};          // Initialized to all 0s
    std::string s{};            // Initialized to an empty string
    int* pi{};                  // Initialized to nullptr
};
```

Initialization and Performance

```
int i{};  
int array[100]{};  
std::string s{};  
int* pi{};  
  
int func()  
{  
    int i{};  
    int array[100]{};  
    std::string s{};  
    int* pi{};  
  
    // ...  
}  
  
struct Widget  
{  
    int i{};  
    int array[100]{};  
    std::string s{};  
    int* pi{};  
};
```



Initializing all integers in an array...
Isn't that too expensive?

Initialization and Performance

```
int i{};  
int array[100]{}; // No; compile-time initialization  
std::string s{};  
int* pi{};  
  
int func()  
{  
    int i{};  
    int array[100]{}; // Potentially; runtime initialization  
    std::string s{};  
    int* pi{};  
  
    // ...  
}  
  
struct Widget  
{  
    int i{};  
    int array[100]{}; // Potentially; runtime initialization  
    std::string s{};  
    int* pi{};  
};
```

Initialization and Performance

```
int i{};  
int array[100]{}; // Always compile-time initialized  
std::string s{};  
int* pi{};  
  
int func()  
{  
    int i{};  
    int array[100] // Not initialized; // Explicitly not initialized  
    std::string s{};  
    int* pi{};  
  
    // ...  
}  
  
struct Widget  
{  
    int i{};  
    int array[100] // Not initialized; // Explicitly not initialized  
    std::string s{};  
    int* pi{};  
};
```



Initialization and Performance

```
int i{};  
int array[100]{};                                // Always compile-time initialized  
std::string s{};  
int* pi{};  
  
int func()  
{  
    int i{};  
    int array[100] [[indeterminate]];      // C++26: Explicitly not initialized  
    std::string s{};  
    int* pi{};  
  
    // ...  
}  
  
struct Widget  
{  
    int i{};  
    int array[100] [[indeterminate]];      // C++26: Explicitly not initialized  
    std::string s{};  
    int* pi{};  
};
```

Guidelines

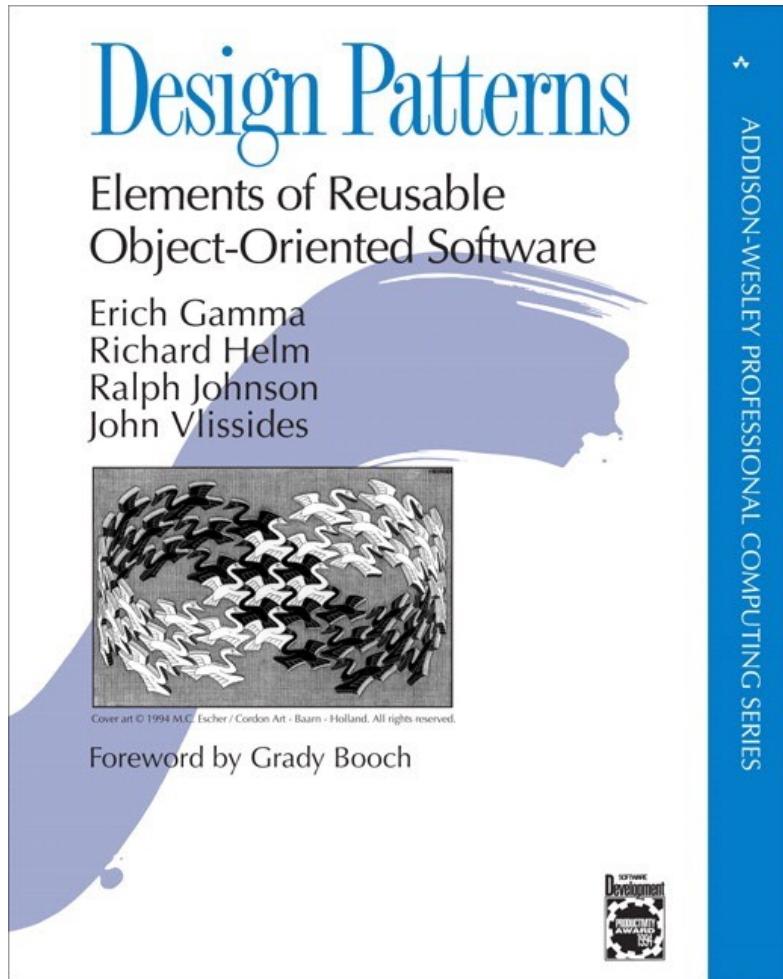
Core Guideline ES.20: Always initialise an object.

Guideline: Prefer to be explicit when not initializing an object, preferably with `[[indeterminate]]`.

2.6. Lifetime Safety

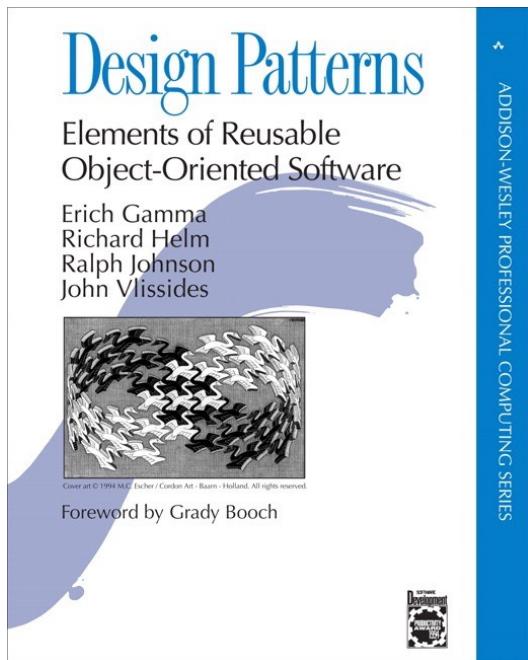


Terminology

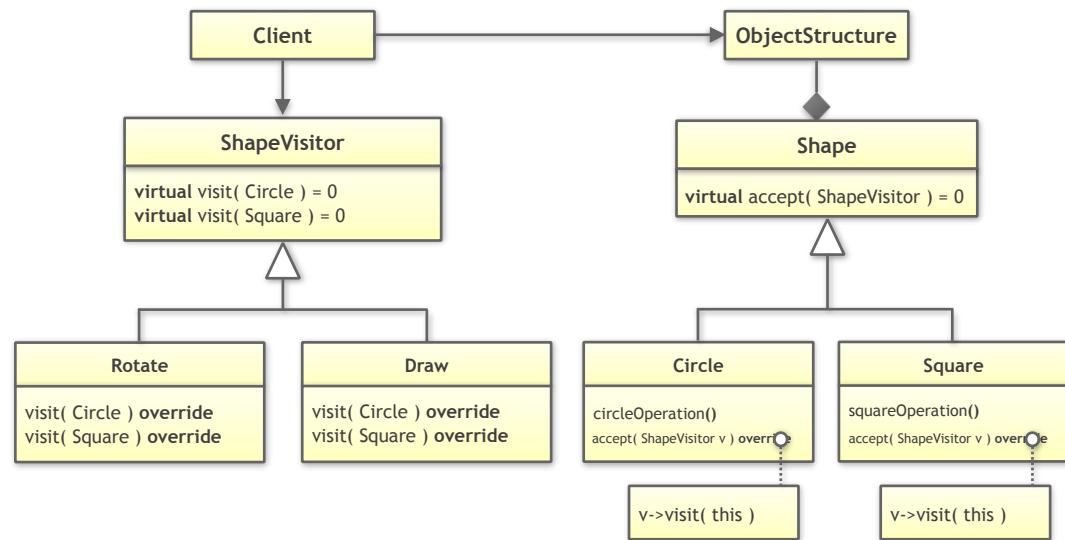


- The Gang of Four (GoF) book
- Published in 1994
- Source of 23 of the most commonly used design patterns
- Almost all design patterns are based on inheritance

2. Safe C++ - Lifetime Safety



One of these patterns is the Visitor design pattern



A Visitor-Based Solution

```
class Circle;
class Square;

class ShapeVisitor
{
public:
    virtual ~ShapeVisitor() = default;

    virtual void visit( Circle const& ) const = 0;
    virtual void visit( Square const& ) const = 0;
};

class Shape
{
public:
    Shape() = default;
    virtual ~Shape() = default;

    virtual void accept( ShapeVisitor const& ) = 0;
};

class Circle : public Shape
{
public:
    explicit Circle( double rad )
        : radius{ rad }
        , // ... Remaining data members
    {}
}
```

A Visitor-Based Solution

```
class Circle;
class Square;

class ShapeVisitor
{
public:
    virtual ~ShapeVisitor() = default;

    virtual void visit( Circle const& ) const = 0;
    virtual void visit( Square const& ) const = 0;
};

class Shape
{
public:
    Shape() = default;
    virtual ~Shape() = default;

    virtual void accept( ShapeVisitor const& ) = 0;
};

class Circle : public Shape
{
public:
    explicit Circle( double rad )
        : radius{ rad }
        , // ... Remaining data members
    {}
}
```

A Visitor-Based Solution

```
class Circle;
class Square;

class ShapeVisitor
{
public:
    virtual ~ShapeVisitor() = default;

    virtual void visit( Circle const& ) const = 0;
    virtual void visit( Square const& ) const = 0;
};

class Shape
{
public:
    Shape() = default;
    virtual ~Shape() = default;

    virtual void accept( ShapeVisitor const& ) = 0;
};

class Circle : public Shape
{
public:
    explicit Circle( double rad )
        : radius{ rad }
        , // ... Remaining data members
    {}
}
```

A Visitor-Based Solution

```
virtual void accept( ShapeVisitor const& ) = 0;  
};  
  
class Circle : public Shape  
{  
public:  
    explicit Circle( double rad )  
        : radius{ rad }  
        , // ... Remaining data members  
    {}  
  
    double getRadius() const noexcept;  
    // ... getCenter(), getRotation(), ...  
  
    void accept( ShapeVisitor const& ) override;  
  
    // ...  
  
private:  
    double radius;  
    // ... Remaining data members  
};  
  
class Square : public Shape  
{  
public:  
    explicit Square( double s )  
        : side{ s }  
        , // ... Remaining data members  
    {}  
};
```

A Visitor-Based Solution

```
// ... Remaining data members
};

class Square : public Shape
{
public:
    explicit Square( double s )
        : side{ s }
        , // ... Remaining data members
    {}

    double getSide() const noexcept;
    // ... getCenter(), getRotation(), ...

    void accept( ShapeVisitor const& ) override;

    // ...

private:
    double side;
    // ... Remaining data members
};

class Draw : public ShapeVisitor
{
public:
    void visit( Circle const& ) const override;
    void visit( Square const& ) const override;
};
```

A Visitor-Based Solution

```
double side;
// ... Remaining data members
};

class Draw : public ShapeVisitor
{
public:
    void visit( Circle const& ) const override;
    void visit( Square const& ) const override;
};

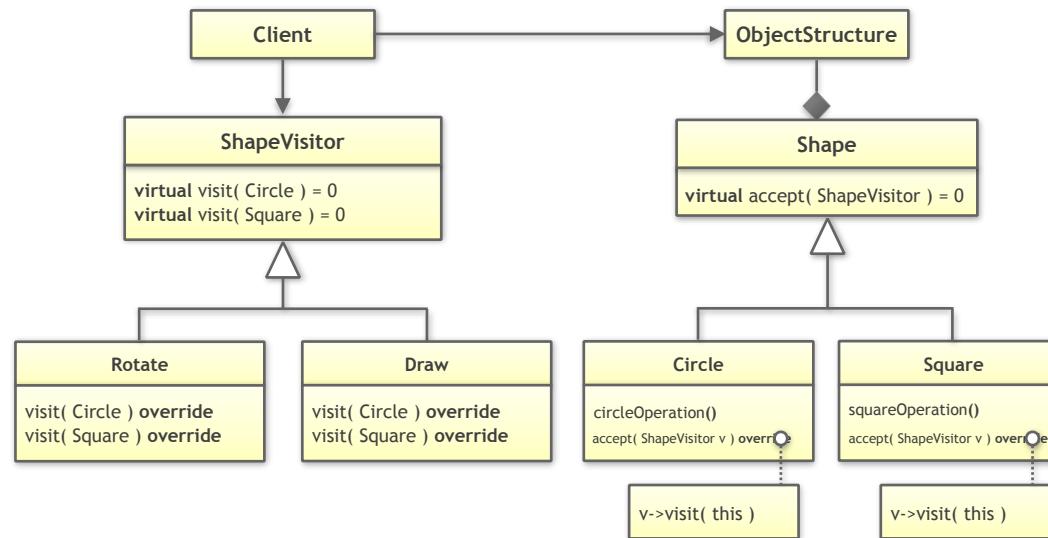
void drawAllShapes( std::vector<std::unique_ptr<Shape>> const& shapes )
{
    for( auto const& s : shapes )
    {
        s->accept( Draw{} )
    }
}

int main()
{
    using Shapes = std::vector<std::unique_ptr<Shape>>;
    // Creating some shapes
    Shapes shapes;
    shapes.emplace_back( std::make_unique<Circle>( 2.0 ) );
    shapes.emplace_back( std::make_unique<Square>( 1.5 ) );
}
```

A Visitor-Based Solution

```
void drawAllShapes( std::vector<std::unique_ptr<Shape>> const shapes )  
{  
    for( auto const& s : shapes )  
    {  
        s->accept( Draw{} )  
    }  
}  
  
int main()  
{  
    using Shapes = std::vector<std::unique_ptr<Shape>>;  
  
    // Creating some shapes  
    Shapes shapes;  
    shapes.emplace_back( std::make_unique<Circle>( 2.0 ) );  
    shapes.emplace_back( std::make_unique<Square>( 1.5 ) );  
    shapes.emplace_back( std::make_unique<Circle>( 4.2 ) );  
  
    // Drawing all shapes  
    drawAllShapes( shapes );  
}
```

2. Safe C++ - Lifetime Safety



The classic style of programming has many disadvantages:

- ⌚ We have a **two inheritance hierarchies** (intrusive)
- ⌚ Performance is reduced due to **two virtual function calls** per operation
- ⌚ Promotes **dynamic memory allocation**
- ⌚ Performance is reduced due to **many small, manual allocations**
- ⌚ We usually **manage lifetimes explicitly** (`std::unique_ptr`)
- ⌚ Performance is affected due to **many pointers** (indirections)
- ⌚ Danger of **lifetime-related bugs**

2. Safe C++ - Lifetime Safety

The image is a screenshot from a video conference. In the top left corner, the Cppcon 2022 logo is displayed, featuring a stylized 'C' icon, the text 'Cppcon 2022 | The C++ Conference', and the date 'September 12th-16th'. To the right of the logo is a large, semi-transparent image of a white fish swimming in blue water. A speech bubble originates from a portrait of Dave Abrahams on the left, containing his spoken words. Below the portrait, the name 'Dave Abrahams' is displayed. At the bottom of the screen, there is a navigation bar with video controls (play, pause, volume, etc.) and a progress bar indicating the video is at 1:19 of 1:00:45. The video is titled 'Problem >'. On the right side of the video frame, there is a small 'think-cell' logo.

People, we have a problem, and it's one we may not really like to face. In fact, we maybe can't even see it, because we are swimming in it. It's like water to the fish. I'm talking about reference semantics.

Dave Abrahams

Values: Safety, Regularity, Independence, and the Future of Programming

1:19 / 1:00:45 • Problem >

Video Sponsorship Provided By:

think-cell

<https://www.youtube.com/watch?v=QthAU-t3PQ4>

The Solution: Value Semantics





A Value-Based Solution: std::variant

```
class Circle
{
public:
    explicit Circle( double rad )
        : radius{ rad }
        , // ... Remaining data members
    {}

    double getRadius() const noexcept;
    // ... getCenter(), getRotation(), ...

private:
    double radius;
    // ... Remaining data members
};

class Square
{
public:
    explicit Square( double s )
        : side{ s }
        , // ... Remaining data members
    {}

    double getSide() const noexcept;
    // ... getCenter(), getRotation(), ...

private:
    double side;
```



A Value-Based Solution: std::variant

```
class Circle
{
public:
    explicit Circle( double rad )
        : radius{ rad }
        , // ... Remaining data members
    {}

    double getRadius() const noexcept;
    // ... getCenter(), getRotation(), ...

private:
    double radius;
    // ... Remaining data members
};

class Square
{
public:
    explicit Square( double s )
        : side{ s }
        , // ... Remaining data members
    {}

    double getSide() const noexcept;
    // ... getCenter(), getRotation(), ...

private:
    double side;
```



A Value-Based Solution: std::variant

```
private:  
    double radius;  
    // ... Remaining data members  
};  
  
class Square  
{  
public:  
    explicit Square( double s )  
        : side{ s }  
        , // ... Remaining data members  
    {}  
  
    double getSide() const noexcept;  
    // ... getCenter(), getRotation(), ...  
  
private:  
    double side;  
    // ... Remaining data members  
};  
  
using Shape = std::variant<Circle,Square>;  
  
class GLDrawer  
{  
public:  
    void operator()( Circle const& ) const;  
    void operator()( Square const& ) const;
```



A Value-Based Solution: std::variant

```
private:  
    double side;  
    // ... Remaining data members  
};  
  
using Shape = std::variant<Circle,Square>;  
  
class GLDrawer  
{  
public:  
    void operator()( Circle const& ) const;  
    void operator()( Square const& ) const;  
    // ...  
};  
  
void drawAllShapes( std::vector<Shape> const& shapes )  
{  
    for( auto const& s : shapes )  
    {  
        std::visit( GLDrawer{}, s );  
    }  
}  
  
int main()  
{  
    using Shapes = std::vector<Shape>;
```



A Value-Based Solution: std::variant

```
private:  
    double side;  
    // ... Remaining data members  
};  
  
using Shape = std::variant<Circle,Square>;  
  
class GLDrawer  
{  
public:  
    void operator()( Circle const& ) const;  
    void operator()( Square const& ) const;  
    // ...  
};  
  
void drawAllShapes( std::vector<Shape> const& shapes )  
{  
    for( auto const& s : shapes )  
    {  
        std::visit( GLDrawer{}, s );  
    }  
}  
  
int main()  
{  
    using Shapes = std::vector<Shape>;
```



A Value-Based Solution: std::variant

```
private:  
    double side;  
    // ... Remaining data members  
};  
  
using Shape = std::variant<Circle,Square>;  
  
class GLDrawer  
{  
public:  
    void operator()( Circle const& ) const;  
    void operator()( Square const& ) const;  
    // ...  
};  
  
void drawAllShapes( std::vector<Shape> const& shapes )  
{  
    for( auto const& s : shapes )  
    {  
        std::visit( GLDrawer{}, s );  
    }  
}  
  
int main()  
{  
    using Shapes = std::vector<Shape>;
```



A Value-Based Solution: std::variant

```
void drawAllShapes( std::vector<Shape> const& shapes )
{
    for( auto const& s : shapes )
    {
        std::visit( GLDrawer{}, s );
    }
}

int main()
{
    using Shapes = std::vector<Shape>;

    // Creating some shapes
    Shapes shapes;
    shapes.emplace_back( Circle{ 2.0 } );
    shapes.emplace_back( Square{ 1.5 } );
    shapes.emplace_back( Circle{ 4.2 } );

    // Drawing all shapes
    drawAllShapes( shapes );
}
```



From Pointers/References to Values

Task (2_Safe_Cpp/Visitor_Refactoring):

Step 1: Implement the `area()` operations as a classic visitor. Hint: the area of a circle is `radius*radius*M_PI`, the area of a square is `side*side`.

Step 2: Refactor the classic Visitor solution by a value semantics based solution. Note that the general behavior should remain unchanged.

Step 3: Switch from one to another graphics library. Discuss the feasibility of the change: how easy is the change? How many pieces of code on which level of the architecture have to be touched?

Step 4: Add the new feature to serialize shapes by means of the FastSerialization library. Observe how well the new code can be integrated and how many new dependencies are created.

Step 5: Discuss the advantages of the value semantics based solution in comparison to the classic solution.

From Pointers/References to Values

Task (2_Safe_Cpp/Strategy_Refactoring):

Step 1: Refactor the classic Strategy solution by a value semantics based solution. Note that the general behavior should remain unchanged.

Step 2: Switch from one to another graphics library. Discuss the feasibility of the change: how easy is the change? How many pieces of code on which level of the architecture have to be touched?

Step 3: Add an `area()` operations for all shapes. How easy is the change? How many pieces of code on which level of the architecture have to be touched? Hint: the area of a circle is `radius*radius*M_PI`, the area of a square is `side*side`.

Step 4: Add the new feature to serialize shapes by means of the FastSerialization library. Observe how well the new code can be integrated and how many new dependencies are created.



Breaking Dependencies

The Visitor Design Pattern

KLAUS IGLBERGER



 Includes paid promotion >



Breaking Dependencies: Type Erasure - A Design Analysis

KLAUS IGLBERGER



Cppcon
The C++ Conference

0:01 / 1:01:33

20
21



October 24-29



<https://www.youtube.com/watch?v=4eeESJQk-mw>

From Pointers/References to Values



Task (2_Safe_CppToInt): Compare the five different solutions to convert a string into an integer and think about the pros and cons of every solution.



From Pointers/References to Values

```
#include <charconv>
#include <optional>
#include <string_view>

std::optional<int> to_int( std::string_view sv )
{
    std::optional<int> oi{};
    int i{};

    auto const result =
        std::from_chars( sv.data(), sv.data() + sv.size(), i );

    if( result.ec != std::errc::invalid_argument ) {
        oi = i;
    }

    return oi;
}
```

Guidelines

Guideline: Prefer value semantics to reference semantics.

Guideline: Try to reduce the use of pointers.

2.7. The Sharp Edges

2. Safe C++ - The Sharp Edges



But sometimes I have to write dangerous code:
low-level details, memory management, pointer
arithmetic, ... How do I deal with these situations?

2. Safe C++ - The Sharp Edges



Yes, once in a while you'll have to deal with such details. Best practice is to do encapsulate those details as best as possible.

Encapsulate the dangerous/tricky parts

- **Encapsulate** everything dangerous/tricky into separate entities
- This is what other programming languages force you to do
- Write **specific tests** for these parts
- Do more **reviews** or even **code inspections**
- Use **pair programming** for these parts
- And perhaps above all else: **use the available tools**
 - IDEs
 - Static code analysis
 - Sanitizers
 - Fuzzers
 - ...

IDEs

- CLion (JetBrains)
- MS Visual Studio (Microsoft)
- ...

2. Safe C++ - The Sharp Edges

Keynote: How To Address 7 Major C++ Pain Points with Tools - Anastasiia Kazakova - NDC TechTown 2024



Rule 2: refactor with AI

```
-  
numbers(const std::array<int, 9> &arr) {  
    int i = 0;  
    int count = 0;  
    for (i = 0; i < arr.size(); i++) {  
        if (arr[i] % 2 != 0) {  
            count++;  
        }  
    }  
  
    std::cout << "Array with numbers: ";  
    for (int j = 0; j < arr.size(); j++) {  
        std::cout << arr[j] << " ";  
    }  
    std::cout << std::endl;  
  
    std::cout << "Count of odd numbers: " << count;  
}  
  
std::array<int, 9> arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};  
int count = num_of_odd_numbers(arr);  
std::cout << count;
```

AI Assistant

Absolutely, you can use the `std::count_if` algorithm from the Standard Template Library (STL) in combination with a lambda function for this task. The `std::count_if` takes a range and a predicate, and counts the number of elements in the range for which the predicate returns `true`.

Here is your refactored code:

```
C++  
#include <array>  
#include <iostream>  
#include <algorithm>  
  
int  
< num_of_odd_numbers(const std::array<int, 9> &arr) {  
    return std::count_if(arr.begin(), arr.end(), [] (int i) {  
        return i % 2 != 0;  
    });  
}
```

Show Diff

Share your feedback

Waiting...

57

Static Code Analysis

- Clang-Tidy (LLVM)
- CppCheck (Daniel Marjamaki)
- ...

2. Safe C++ - The Sharp Edges

Extra Clang Tools 21.0.0git documentation

CLANG-TIDY

« [Extra Clang Tools 21.0.0git \(In-Progress\) Release Notes](#) :: [Contents](#) :: [Clang-Tidy Checks](#) »

Clang-Tidy

Contents

- [Clang-Tidy](#)
 - [Using Clang-Tidy](#)
 - [Suppressing Undesired Diagnostics](#)

See also:

- [List of Clang-Tidy Checks](#)
- [Clang-tidy IDE/Editor Integrations](#)
- [Getting Involved](#)
- [External Clang-Tidy Examples](#)

`clang-tidy` is a clang-based C++ “linter” tool. Its purpose is to provide an extensible framework for diagnosing and fixing typical programming errors, like style violations, interface misuse, or bugs that can be deduced via static analysis. `clang-tidy` is modular and provides a convenient interface for writing new checks.

Using Clang-Tidy

`clang-tidy` is a [LibTooling](#)-based tool, and it’s easier to work with if you set up a compile command database for your project (for an example of how to do this, see [How To Setup Tooling For LLVM](#)). You can also specify compilation options on the command line after `--`:

```
$ clang-tidy test.cpp -- -Imy_project/include -DMY_DEFINES ...
```

If there are too many options or source files to specify on the command line, you can store them in a parameter file, and use `clang-tidy` with that parameters file:

```
$ clang-tidy @parameters_file
```

`clang-tidy` has its own checks and can also run Clang Static Analyzer checks. Each check has a name and the checks to run can be chosen using the `-checks=` option, which specifies a comma-separated list of positive and negative (prefixed with `-`) globs. Positive globs add subsets of checks, and negative globs remove

2. Safe C++ - The Sharp Edges

- [Clang-Tidy](#)
 - [Using Clang-Tidy](#)
 - [Suppressing Undesired Diagnostics](#)

See also:

- [List of Clang-Tidy Checks](#)
- [Clang-tidy IDE/Editor Integrations](#)
- [Getting Involved](#)
- [External Clang-Tidy Examples](#)

`clang-tidy` is a clang-based C++ “linter” tool. Its purpose is to provide an extensible framework for diagnosing and fixing typical programming errors, like style violations, interface misuse, or bugs that can be deduced via static analysis. `clang-tidy` is modular and provides a convenient interface for writing new checks.

Using Clang-Tidy

`clang-tidy` is a [LibTooling](#)-based tool, and it's easier to work with if you set up a compile command database for your project (for an example of how to do this, see [How To Setup Tooling For LLVM](#)). You can also specify compilation options on the command line after `--`:

```
$ clang-tidy test.cpp -- -Imy_project/include -DMY_DEFINES ...
```

If there are too many options or source files to specify on the command line, you can store them in a parameter file, and use `clang-tidy` with that parameters file:

```
$ clang-tidy @parameters_file
```

`clang-tidy` has its own checks and can also run Clang Static Analyzer checks. Each check has a name and the checks to run can be chosen using the `-checks=` option, which specifies a comma-separated list of positive and negative (prefixed with `-`) globs. Positive globs add subsets of checks, and negative globs remove them. For example,

```
$ clang-tidy test.cpp -checks=-*,clang-analyzer-*,-clang-analyzer-cplusplus*
```

2. Safe C++ - The Sharp Edges

Cppcheck

A tool for static C/C++ code analysis

[Home](#) | [Wiki](#) | [Forum](#) | [Issues](#) | [Developer Info](#) | [Online Demo](#) | [Project page](#)

[Download](#) [Features](#) [News](#) [Documentation](#) [Support](#) [Contribute](#)

Cppcheck is a [static analysis tool](#) for C/C++ code. It provides [unique code analysis](#) to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. The goal is to have very few false positives. Cppcheck is designed to be able to analyze your C/C++ code even if it has non-standard syntax (common in embedded projects).

Cppcheck is available both as open-source (this page) and as **Cppcheck Premium** with extended functionality and support. Please visit www.cppcheck.com for more information and purchase options for the commercial version.

Download

Cppcheck 2.17 (open source)

Platform
Windows 64-bit (No XP support)
Source code (.zip)
Source code (.tar.gz)

File
[Installer](#)
[Archive](#)
[Archive](#)

Packages

Cppcheck can also be installed from various package managers; however, you might get an outdated version then.

2. Safe C++ - The Sharp Edges

Packages

Cppcheck can also be installed from various package managers; however, you might get an outdated version then.

Debian:

```
sudo apt-get install cppcheck
```

Fedora:

```
sudo yum install cppcheck
```

Mac:

```
brew install cppcheck
```

Features

Unique code analysis that detect various kinds of bugs in your code.

Both command line interface and graphical user interface are available.

Cppcheck has a strong focus on detecting undefined behaviour.

Unique analysis

2. Safe C++ - The Sharp Edges

Unique analysis

Using several static analysis tools can be a good idea. There are unique features in each tool. This has been established in many studies.

So what is unique in Cppcheck.

Cppcheck uses unsound flow sensitive analysis. Several other analyzers use path sensitive analysis based on abstract interpretation, that is also great however that has both advantages and disadvantages. In theory by definition, it is better with path sensitive analysis than flow sensitive analysis. But in practice, it means Cppcheck will detect bugs that the other tools do not detect.

In Cppcheck the data flow analysis is not only "forward" but "bi-directional". Most analyzers will diagnose this:

```
void foo(int x)
{
    int buf[10];
    if (x == 1000)
        buf[x] = 0; // <- ERROR
}
```

Most tools can determine that the array index will be 1000 and there will be overflow.

Cppcheck will also diagnose this:

```
void foo(int x)
{
    int buf[10];
    buf[x] = 0; // <- ERROR
    if (x == 1000) {}
}
```

2. Safe C++ - The Sharp Edges

Cppcheck will also diagnose this:

```
void foo(int x)
{
    int buf[10];
    buf[x] = 0; // <- ERROR
    if (x == 1000) {}
}
```

Undefined behaviour

- Dead pointers
- Division by zero
- Integer overflows
- Invalid bit shift operands
- Invalid conversions
- Invalid usage of STL
- Memory management
- Null pointer dereferences
- Out of bounds checking
- Uninitialized variables
- Writing const data

Security

The most common types of security vulnerabilities in 2017 (CVE count) was:

Category	Amount	Detected by Cppcheck
Buffer Errors	2530	A few
Improper Access Control	1366	A few (unintended backdoors)
Information Leak	1426	A few (unintended backdoors)
Permissions, Privileges, and Access Control	1196	A few (unintended backdoors)

Programming Task

Task (2_Safe_Cpp/CppCheck): Use CppCheck to detect the following bugs in the example program: division-by-0, use of uninitialized variables, out-of-bounds access, and use-after-free..

Sanitizers

- Part of every modern C++ compiler
- Address Sanitizer (ASan)
 - Use after free/return/scope
 - Heap/Stack/Global buffer overflow
 - Initialization order bugs
 - Memory leaks
- Memory Sanitizer (MSan)
 - detection of uninitialised memory reads
- Thread Sanitizer (TSan)
 - data race detected
- Undefined Behavior Sanitizer (UBSan)
- ...

Programming Task

Task (3_Concepts_and_STL/constexpr/AddressSanitizer): Use AddressSanitizer to detect the out-of-bounds access in the following demo code.

2. Safe C++ - The Sharp Edges

The screenshot shows a GitHub repository interface for the 'sanitizers' project. The top navigation bar includes links for Code, Issues (618), Pull requests (1), Actions, Projects, Wiki (highlighted with a red underline), Security (1), and Insights. The search bar contains the placeholder 'Type / to search'. On the right side, there are icons for issues, pull requests, and other repository functions.

AddressSanitizer

chefmax edited this page on May 15, 2019 · [29 revisions](#)

Introduction

[AddressSanitizer](#) (aka ASan) is a memory error detector for C/C++. It finds:

- [Use after free](#) (dangling pointer dereference)
- [Heap buffer overflow](#)
- [Stack buffer overflow](#)
- [Global buffer overflow](#)
- [Use after return](#)
- [Use after scope](#)
- [Initialization order bugs](#)
- [Memory leaks](#)

This tool is very fast. The average slowdown of the instrumented program is ~2x (see [AddressSanitizerPerformanceNumbers](#)).

The tool consists of a compiler instrumentation module (currently, an LLVM pass) and a run-time library which replaces the `malloc` function.

The tool works on x86, ARM, MIPS (both 32- and 64-bit versions of all architectures), PowerPC64. The supported operation systems are Linux, Darwin (OS X and iOS Simulator), FreeBSD, Android:

OS	x86	x86_64	ARM	ARM64	MIPS	MIPS64	PowerPC	PowerPC64
Linux	yes	yes			yes	yes	yes	yes
OS X	yes	yes						
iOS Simulator	yes	yes						

The sidebar on the right contains a navigation menu with the following sections and links:

- Pages (78)
 - Find a page...
 - Home
 - AddressSanitizer
 - Introduction
 - Getting AddressSanitizer
 - Using AddressSanitizer
 - Interaction with other tools
 - gdb
 - ulimit -v
 - Flags
 - Call stack
 - Incompatibility
 - Turning off instrumentation
 - FAQ
 - Talks and papers
 - Comments?
 - AddressSanitizerAlgorithm
 - AddressSanitizerAndDebugger
 - AddressSanitizerAndroidPlatform
 - AddressSanitizerAsDso

Address Sanitizer

```
    countries.insert( pos, country );
}
}

std::vector<Country> five_biggest_countries{ begin(countries), begin(countries)+5U };

for( size_t i=1U; i<5U; ++i )
{
    auto& country1 = five_biggest_countries[i-1U];
    for( size_t j=i; j<5U; ++j ) {
        auto& country2 = five_biggest_countries[j];
        if( country1.residents < country2.residents ) {
            Country tmp{ country1 };
            country1 = country2;
            country2 = tmp;
        }
    }
}

for( auto const& country : five_biggest_countries ) {
    std::cout << country << '\n';
}
```



Let's assume that there are only 4 countries instead of a minimum of 5

Address Sanitizer

```
=====
==88511==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x611000000150 at pc 0x0001082bfe14 bp
READ of size 8 at 0x611000000150 thread T0
#0 0x0001082bfe13 in __asan_memcpy+0xf23 (libclang_rt.asan_osx_dynamic.dylib:x86_64h+0xd7e13)
#1 0x0001078dea58 in Country::Country(Country const&)+0x58 (AddressSanitizer:x86_64+0x100011a58)
#2 0x0001078cf3c in Country::Country(Country const&)+0x1c (AddressSanitizer:x86_64+0x100002d3c)
#3 0x0001078d7798 in Country* std::__1::__construct_at
```

0x611000000150 is located 48 bytes after 224-byte region [0x611000000040, 0x611000000120)
allocated by thread T0 here:

```
#0 0x0001082d6a6d in _Znwm+0x7d (libclang_rt.asan_osx_dynamic.dylib:x86_64h+0xeea6d)
#1 0x0001078d1ae4 in void* std::__1::__libcpp_operator_new
```

Address Sanitizer

SUMMARY: AddressSanitizer: heap-buffer-overflow (AddressSanitizer:x86_64+0x100011a58) in Country::Country

Shadow bytes around the buggy address:

```
0x610fffffe80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
0x610fffffff00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
0x610fffffff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
0x611000000000: fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00  
0x611000000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
=>0x611000000100: 00 00 00 00 fa fa fa fa fa fa[fa]fa fa fa fa fa  
0x611000000180: fd  
0x611000000200: fd fa  
0x611000000280: fa  
0x611000000300: fa  
0x611000000380: fa fa
```

Shadow byte legend (one shadow byte represents 8 application bytes):

Addressable: 00

Partially addressable: 01 02 03 04 05 06 07

Heap left redzone: fa

Freed heap region: fd

Stack left redzone: f1

Stack mid redzone: f2

Stack right redzone: f3

Stack after return: f5

Stack use after scope: f8

Global redzone: f9

Global init order: f6

Poisoned by user: f7

Container overflow: fc

Array cookie: ac

Intra object redzone: bb

ASan internal: fe

Left alloca redzone: ca

Right alloca redzone: ch

Fuzzers

- AFL++
- Google/fuzztest
- ...

2. Safe C++ - The Sharp Edges

AFLplusplus / AFLplusplus

Type to search

Code Issues 28 Pull requests Discussions Actions Security Insights

AFLplusplus Public

Sponsor Watch 84 Fork 1.1k Star 5.5k

stable 34 Branches 36 Tags Go to file Add file Code

vanhauser-thc Merge pull request #2320 from AFLplusplus/dev f590973 · last week 7,313 Commits

Commit	Message	Time
.github	feat: re-enable arm64 docker containers. Use GH arm runn...	3 weeks ago
benchmark	Update COMPARISON.md	6 months ago
coresight_mode	various changes	3 years ago
custom_mutators	Fix various spelling errors (#2293)	last month
dictionaries	Add JSON Schema dictionary	5 months ago
docs	fix doc	last week
frida_mode	Add iOS cross-compilation support	2 weeks ago
include	Add fflush(stdout); before abort call	last week
instrumentation	Update LTO documentation to reference LLVM 19 in all exa...	2 weeks ago
nyx_mode	Fix various spelling errors (#2293)	last month
qemu_mode	update qemuafl	last month
src	fix We need at least one valid input seed that does not cras...	2 weeks ago
test	Fix various spelling errors (#2293)	last month
testcases	Add docs content overview	4 years ago
unicorn_mode	unicornafl example: fix incorrect comment (#2315)	2 weeks ago
utils	Merge branch 'dev' into ios	2 weeks ago

About

The fuzzer afl++ is afl with community patches, qemu 5.1 upgrade, collision-free coverage, enhanced lf-intel & redqueen, AFLfast++ power schedules, MOpt mutators, unicorn_mode, and a lot more!

[aflplus.plus](#)

testing security instrumentation
qemu fuzzing fuzz-testing afl
afl-fuzz fuzzer unicorn-emulator
afl-fuzzer afl-gcc fuzzer-afl
afl-compiler unicorn-mode

Readme

Apache-2.0 license

Cite this repository

Activity

Custom properties

5.5k stars

84 watching

1.1k forks

Report repository

Releases 36

v4.31c Latest last month

+ 35 releases

2. Safe C++ - The Sharp Edges

google / fuzztest

Type to search

Code Issues Pull requests Discussions Actions Projects Security Insights

fuzztest Public Watch 18 Fork 81 Star 799

main 58 Branches 3 Tags Go to file Add file <> Code

fniksic and copybara-github Replace C++20 branch predictor hints with C++17-compliant hints 2c14292 · 2 days ago 1,206 Commits

	.github/workflows	Increase crash reporting limit when running with FuzzTest.	last month
	bazel	Simplified setup_configs.sh to run properly with MODULE.bazel	2 months ago
	build_defs	Reference @google_com_fuzztest inside the cc_fuzztest_g...	8 months ago
	centipede	Replace C++20 branch predictor hints with C++17-complia...	2 days ago
	cmake	Bump up the version of Protobuf and fix a few copyright he...	6 months ago
	codelab	Fix CMakeLists.txt in the codelab.	4 months ago
	common	Replace C++20 branch predictor hints with C++17-complia...	2 days ago
	doc	Fix FlatMap example for VectorOf	last month
	domain_tests	Fix a few instances of passing no arguments to a variadic m...	3 weeks ago
	e2e_tests	Distinguish between undetected and failing custom mutator...	2 weeks ago
	fuzztest	Tear down fixture in the controller mode as early as possible.	3 days ago
	grammar_codegen	Add bzlmod support for fuzztest	7 months ago
	tools	Adding support for the InGrammar domain to allow inserting...	last year
	.bazelrc	Enable -Wc++17-compat-pedantic to see incompatibilities ...	2 months ago
	.clang-format	Internal change	3 years ago
	CMakeLists.txt	Pull out the Centipede defs library into a new common pack...	9 months ago

About

No description, website, or topics provided.

Readme

View license

Code of conduct

Security policy

Activity

Custom properties

799 stars

18 watching

81 forks

Report repository

Releases 2

FuzzTest 2024-10-25 Latest
3 weeks ago

+ 1 release

Packages

No packages published

Contributors 43

Guidelines

Guideline: Use the available tools to help to detect problems.

Guideline: It is considered unprofessional today to not use static code analysis, sanitizers, a test framework and fuzzers.

2.8. Summary

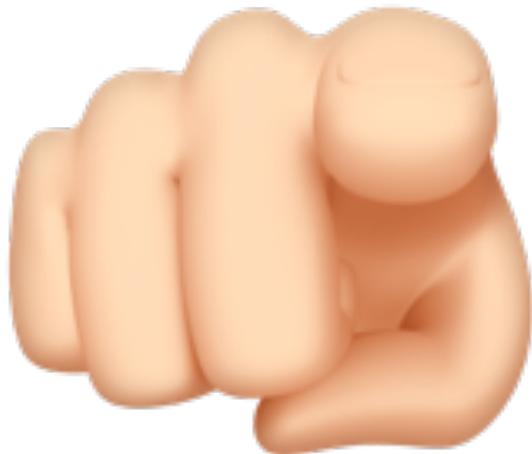
What can we do?



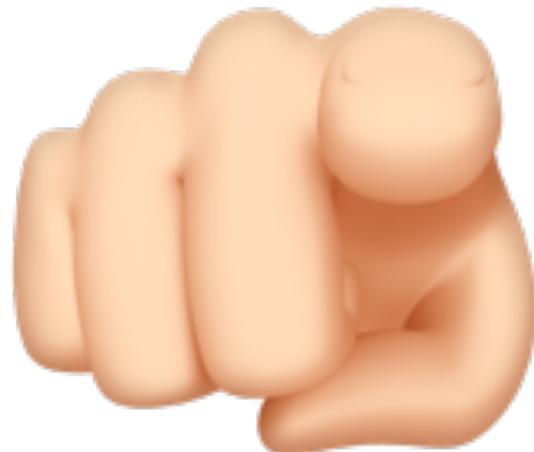
I can do my part as
C++ trainer/consultant



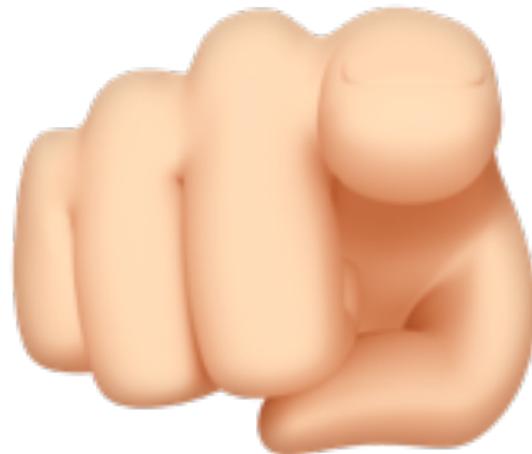
But we also need YOU!



Spread the word that working in
classic C++ is like grovelling in the dirt



Tell people to stand up, to become proud C++ developers that know about the best practices and use the tools



2. Safe C++ - Summary

“With our best practices and tools we can easily remove the vast majority of safety issues.”

(Klaus Iglberger)



2. Safe C++ - Summary



*"If there were 90-98% fewer C++ type/bounds/
initialization/lifetime vulnerabilities we
wouldn't be having this discussion."*

(Herb Sutter)

Summary

- ➊ Programming in C++ isn't safe by default, we have do some homework
- ➋ We have great solutions to eliminate most common safety problems
 - ➌ Most bounds safety issues (> 95%) can be resolved by not using indices/ iterators (algorithms, ranges, ...)
 - ➌ 100% type safety by strong types and constraints (C++20 concepts)
 - ➌ Almost all lifetime safety issues (~ 99%) can be resolved by using value semantics
 - ➌ No undefined behavior by constexpr and sanitisers
- ➌ These are not new solutions, by now they are mainstream – use them!
- ➌ Use the idiomatic rules and tools of C++
 - ➌ You cannot complain about safety issues if you don't use these tools
- ➌ It's a people problem: Too few people use the idiomatic C++ of today
- ➌ Help to spread the information how C++ should be used today

References (Safety)

- JF Bastien: Safety and Security: The Future of C++. C++Now 2023 (<https://www.youtube.com/watch?v=Gh79wcGJdTg>)
- Timur Doumler: “C++ and Safety”. C++ on Sea 2023 (<https://www.youtube.com/watch?v=imtpoc9jtOE>)
- Sean Parent: “Safety in C++: All the Safeties!”. C++ on Sea 2023 (<https://www.youtube.com/watch?v=BaUv9sgLCPc>)
- Louis Dionne, “Security in C++ - Hardening Techniques From the Trenches”. C++Now 2024 (<https://www.youtube.com/watch?v=t7EJTO0-reg>)

References (Strong Types)

- Howard Hinnant: Design Rationale for the <chrono> Library. Meeting C++ 2019 (<https://www.youtube.com/watch?v=adSAN282YIw>)
- Jonathan Boccara: Strongly typed constructors. Fluent{C++} (<https://www.fluentcpp.com/2016/12/05/named-constructors/>)
- Jonathan Müller: Tutorial: Emulating strong/opaque typedefs in C++. foonathan::blog() (<https://foonathan.net/2016/10/strong-typedefs/>)
- Alex Dathskovsky: To Int or to Uint, This is the Question. CppCon 2024 (<https://www.youtube.com/watch?v=pnaZ0x9Mmm0>)

References (Tools)

- CLion: <https://www.jetbrains.com/clion/>
- MS Visual Studio Code: <https://code.visualstudio.com/>
- Clang Tidy: <https://clang.llvm.org/extra/clang-tidy/>
- CppCheck: <https://cppcheck.sourceforge.io/>
- AddressSanitizer: <https://github.com/google/sanitizers/wiki/AddressSanitizer>
- AFL++: <https://aflplus.plus/>
- Google/fuzztest: <https://github.com/google/fuzztest>

email: klaus.iglberger@cpp-training.com

LinkedIn: [linkedin.com/in/klaus-iglberger](https://www.linkedin.com/in/klaus-iglberger)

Xing: [xing.com/profile/Klaus_Iglberger/cv](https://www.xing.com/profile/Klaus_Iglberger/cv)