C++ Pitfalls and Sharp Edges to Avoid

Amir Kirsh

Cpp North 2025

(picture by Amir Kirsh, no AI involved)

About me

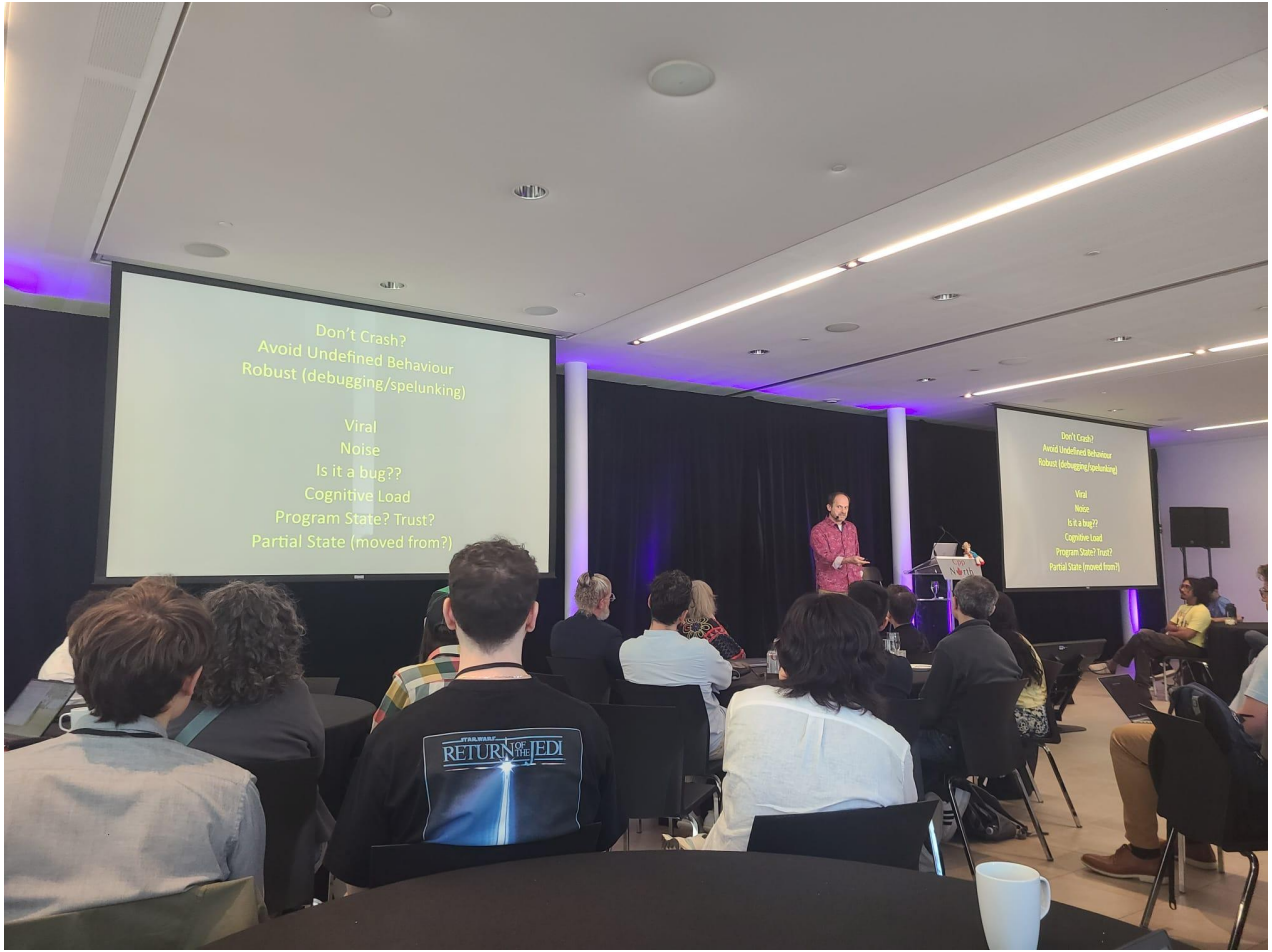
Lecturer
Academic College of Tel-Aviv-Yaffo
Tel-Aviv University

Co-Organizer of the **CoreCpp** conference and meetup group

Trainer and Advisor
(C++, but not only)

https://www.linkedin.com/posts/rainergrimm_cpp-cplusplus-als-activity-7341052629470339072-D1Tp/

Should we check for null?


Don't Crash?
Avoid Undefined Behaviour
Robust (debugging/spelunking)
Viral
Noise
Is it a bug??
Cognitive Load
Program State? Trust?
Partial State (moved from?)
RETURN OF THE JEDI

Should we check for null?

It depends …

Should we check for null?

~~It depends ...~~

YES !

```
void process(int* ptr)
    pre(ptr != nullptr)  // a check!
{
    // Safe to dereference
    *ptr += 1;
}
```

Bugs,


DON'T TEST ME!
Segfault again?!
<TEST_CASE>
TEST_CASE();
REQUIRE(REQT&ttest'');
REQUIER(REQI5);
}
fix later?');
comnfaut ageheen');
}
Segmentation fault
undefined behavior
desecution fault
double free?
int main()
Advanced C++ Testing

Bugs, where do they come from?

A real story of a Bug in Production – 02-Jan-2011

Thursday, January 13, 2011

Freaking behavior of a small little C/C++ bug

Oh boy.
Read till the end the event and its root cause. Important morals follow below.

We run systems that on high capacity events handle thousands of transactions per second. One of the most heavy-traffic periods is New-Year's-Eve, the 31st of December, were most of our systems are under heavy stress around the world, stress that tends to difuse to our support teams. Structured and strict preparations usually make us pass this heavy-traffic day properly in most, if not all sites. Which happily was the case also this year.

Shockingly, on January 2nd we had a crash in two sites.

Analyzing the crash led to a timer that instead of re-scheduling itself for every 5 seconds, keeps snapping abruptly in periods of milliseconds.

While still analyzing the case, reproducing it in our labs, the problem vanished as suddenly as it appeared, on the end of the same day. January 3rd, 00:00, systems went back to behave nicely.

https://softwareanimals.blogspot.com/2011/01/freaking-behavior-of-small-little-cc.html

Essence and accident in software engineering

Picture by Amir Kirsh: https://www.infoq.com/articles/No-Silver-Bullet-Summary -- OOPSLA 2005, Montreal

“No Silver Bullet - Essence and Accident in Software Engineering” by Fred Brooks, 1986

Bugs related to “Essence” vs. “Accident”

	“Essential” Bugs	“Accidental” Bugs
Nature	Conceptual	Technical
Source	Misunderstanding, Misconception, Omission – In the Problem Domain	Faults in Implementation or Environment
Error Type	Logical Errors	Implementation Errors

“Essential” Bugs

“Essential” Bugs

- **Requirement misunderstandings / Missing requirements:** faulty requirements; wrong assumptions; improper error handling;
- **Complicated state management:** wrong flow chosen due to complicated state (coding error, usually due to bad design); changing system state when shouldn’t;
- **Faulty conceptual models:** implementing a model that doesn't actually reflect how the system should behave;
- ...

Essence is complicated...

- Understanding and foreseeing all possible inputs.
- And combination of inputs.
- Defining system behavior for all possible states.
- Defining fail-safe.
- Setting a safe path from bad state to fail-safe.
- …

🔑 Handling the Essence

- **Clear and actionable requirements**
  - Aligned with actual user / system needs
  - Understandable, consistent and complete
- **A good model (architecture) that fits the requirements**
  - Fitting, maintainable, future-ready
- **Effective validation, testing and traceability**
  - Reliable, comprehensive, trustworthy

“Accidental” Bugs

“Accidental” Bugs (1)

- **Typos:** using the wrong var; calling the wrong function; using () instead of [ ] or vice versa; wrong order of arguments; use of = instead of == or vice versa;
- **Semantic misuse:** initializing with {} instead of () or vice versa, not realizing the difference; using “static” on a variable without following the implications; using “auto” on a variable without following the implications; being unaware to “short-circuit” conditions behavior; relying on order of arguments evaluation in function call;

“Accidental” Bugs (2)

- **Conversions:** bad *implicit* and *explicit* conversions;
- **Overflow / Underflow:** both for signed (UB) and unsigned (is it planned?);
- **Floating-point errors:** cumulative rounding errors; precision loss; comparison of floating numbers;
- **Ownership and data management:** mutating data assuming wrongly single ownership; updating a copy when the original should be updated; (byval / byref mistakes).

“Accidental” Bugs (3)

- **Using uninitialized data:** uninitialized variables, pointers, objects;
- **Using dead resources:** dangling pointers and references; invalidated iterators; accessing freed memory; null dereference; moved-from object;
- **Off-by-one errors:** the classic loop boundaries mistake;
- **Out-of-bounds access:** by index; pointer arithmetic;
- **Wrong item accessed:** by index; pointer arithmetic; iterator;

“Accidental” Bugs (4)

- **Resource leaks:** memory, files, connections, handles of any kind;
- **Threading and concurrency:** data races; deadlocks;
- **Bad API use:** ignoring API’s contract (documented or not); sending wrong measurement units; flipping arguments; ignoring error cases / exceptions / return value / output parameters;
- **Critical inefficiencies:** redundant copying; repeated unnecessary work in a loop; unnecessary use of expensive APIs

Wow, so many potential bugs...

There are “Safety Nets”

The “Safety Nets” (1)

⚠️ **Compiler warnings**

always solve them, they are stronger than any best practice!
(*note:* `-Wall` is not all)

The “Safety Nets” (1)

⚠️ **Compiler warnings**

always solve them, they are stronger than any best practice!
(*note:* `-Wall` is not all)

Is there a warning flag for that:

```
int main() {
    int* ptr = nullptr;
    return *ptr + 1;
}
```

`-Wall -Wextra`
do not warn on that!

The “Safety Nets” (1)

⚠️ **Compiler warnings**

always solve them, they are stronger than any best practice!
(*note:* `-Wall` is not all)

Is there a warning flag for that:

```
int main() {
    int* ptr = nullptr;
    return *ptr + 1;
}
```

`-Wall -Wextra`
do not warn on that!

Answer: `-Wnull-dereference`

See: Why do compilers not warn about null dereferencing - Stack Overflow
https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html

The “Safety Nets” (1)

⚠️ **Compiler warnings** - recommended set of flags:

```
-Wall -Wextra -Werror -Wpedantic -Wrestrict -Wcast-qual -Wshadow
-Wnon-virtual-dtor -Wunused -Wduplicated-cond -Wcast-align
-Wnull-dereference -Wdouble-promotion -Wfloat-equal -Wuseless-cast
-Wsign-conversion -Wconversion -Wlogical-op
```

Choose more if you want:

https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html

You may want to omit the following warnings (consider each one separately) to avoid false positives:

```
-Wno-comment -Wno-stringop-overflow -Wno-array-bounds
```

The “Safety Nets” (2)

🔬 **Static code analysis tools:**

Use them, they help you conform with best practices and to avoid bugs

See for example: https://rules.sonarsource.com/cpp/RSPEC-5912

🕵️ **Runtime Sanitizers:** address, ub, threading

The “Safety Nets” (3)

📋 **Testing: Unit Tests, Component, Integration & System Testing**

Remember that 100% branch coverage is not enough if tests are “naive”

Consider using mutation testing and fuzz testing.

The “Safety Nets” (4)

📘 Best practices

https://isocpp.github.io/CppCoreGuidelines
https://isocpp.org/wiki/faq/coding-standards
https://google.github.io/styleguide/cppguide.html

and other (sometimes contradicting...) resources

🔍 Code Review

Make sure your code reviews include actual showstopper decisions when needed.

🤖 AI-Reviews

👥 Pair-Programming

The “Safety Nets” (5)

🚦 **Requirements ⇔ Testing ⇔ Code – Coverage Management**

To make sure all requirements are handled and there is no excess code that doesn’t serve any requirement.

🛡️ **Protective Programming**

Use asserts and checks (e.g. contracts) for preconditions and postconditions. (For critical checks asserts are not enough!)

🔁 **Continuous Integration Process**

Automatically build and test code on every change, catching errors early in the development process.

When the safety nets work...

Code Example

What’s the bug in this piece of code:

```
bool done = false;
// ...
if (done == true); {
  std::cout << "Done!\n";
}
```

https://compiler-explorer.com/z/3Ejc7znE8

Code Example

Safety Nets (1)

What's the bug in this piece of code:

```
bool done = false;
// ...
if (done == true); {
  std::cout << "Done!\n";
}
```

https://compiler-explorer.com/z/3Ejc7znE8

Compiler warnings (-Wall -Wextra)

```
warning: suggest braces around empty body in an 'if' statement
[-Wempty-body]
   5 |     if (done == true); {
     |                       ^
warning: this 'if' clause does not guard...
[-Wmisleading-indentation]
   5 |     if (done == true); {
     |     ^~
note: ...this statement, but the latter is misleadingly indented as
if it were guarded by the 'if'
   5 |     if (done == true); {
     |
```

Code Example

Safety Nets (2)

What's the bug in this piece of code:

```
bool done = false;
// ...
if (done == true); {
  std::cout << "Done!\n";
}
```

https://compiler-explorer.com/z/3Ejc7znE8

Static Code Analysis (Clang-Tidy)

```
warning: if statement has empty body
[clang-diagnostic-empty-body]
   5 |     if (done == true); {
     |                       ^
note: put the semicolon on a separate line to silence this
warning
1 warning generated.
```

Semicolon production memory leak in Java code

Thursday, August 19, 2010

Semi-colon and java.lang.OutOfMemoryError

; ; ; ; ; ; ; ; ; ; ; ; ; ;

I want to share with you a crash at customer site caused by java.lang.OutOfMemoryError.

Here is the original code:

```
if (synchRemove(lobj.getSeqNum()) != null);
    timeoutedList.add(lobj);
```

Can you see the problem?

(Well it's much easier after the relevant lines of code are isolated. In reality it took a few days and nights to get to these lines, remem
customer environment where not all relevant info is easily available for the development team. The OOM doesn't necessarily occur at

Moral:

1. Small semicolon can cause big troubles
2. It's hard to see everything in code review. A trouble-making redundant semicolon can skip the eyes of the reviewer
3. Load test may find such cases (but may still miss them, if the relevant scenario was not created)
4. Good unit tests may also help
5. Most Coding Guidelines require curly brackets for any block, even containing only one line. This could possibly reveal the error

https://softwareanimals.blogspot.com/2010/08/semi-colon-and-javalangoutofmemoryerror.html

Another Code Example

Another Code Example

What's the bug in this piece of code:

```
class Point {
  int x_, y_;
public:
 Point(int x = 0, int y = 0)
    : x_(x), y_(y) {}
};

int main() {
    Point p = (1, 2);
}
```

https://compiler-explorer.com/z/xqKsbdsT1

Another Code Example

What's the bug in this piece of code:

```
class Point {
  int x_, y_;
public:
 Point(int x = 0, int y = 0)
    : x_(x), y_(y) {}
};

int main() {
    Point p = (1, 2);
}
```

https://compiler-explorer.com/z/xqKsbdsT1

Safety Nets (1)

Compiler warnings (-Wall -Wextra)

```
warning: left operand of comma operator has no effect
[-Wunused-value]
  11 |     Point p = (1, 2);
     |               ^
warning: variable 'p' set but not used [-Wunused-but-set-variable]
  11 |     Point p = (1, 2);
     |           ^
```

Another Code Example

Safety Nets (2)

What's the bug in this piece of code:

```
class Point {
  int x_, y_;
public:
 Point(int x = 0, int y = 0)
    : x_(x), y_(y) {}
};

int main() {
    Point p = (1, 2);
}
```

https://compiler-explorer.com/z/xqKsbdsT1

Static Code Analysis (Clang-Tidy)

```
<source>:11:16: warning: left operand of comma operator has
no effect [clang-diagnostic-unused-value]
  11 |     Point p = (1, 2);
     |                ^
1 warning generated.
```

But… the safety nets do not always work

But… the safety nets do not always work

⇒ Code may look fine, or actually be fine (but not in our case) thus **not issuing compiler or static code analysis warnings**.

⇒ Some compiler and static code analysis warnings are **suppressed/disabled** to avoid too many false positives.

⇒ **Code may cause UB** (or unspecified behavior) which is not always caught in testing (not all cases of UB are caught by UB sanitizers).

⇒ **Not all issues are detected in testing**.

Code Example

Code Example

What's wrong with this piece of code:

```
void foo(int i, int j) {
  std::cout << i << ' ' << j << std::endl;
}

int g = 0;

int f1() { return ++g; }
int f2() { return ++g; }

int main() {
  foo(f1(), f2());
}
```

https://compiler-explorer.com/z/Pxrn4evfP

Code Example

What's wrong with this piece of code:

```
void foo(int i, int j) {
  std::cout << i << ' ' << j << std::endl;
}

int g = 0;

int f1() { return ++g; }
int f2() { return ++g; }

int main() {
  foo(f1(), f2());
}
```

https://compiler-explorer.com/z/Pxrn4evfP

GCC

```
2 1
```

Clang

```
1 2
```

Code Example

What's wrong with this piece of code:

```
void foo(int i, int j) {
  std::cout << i << ' ' << j << std::endl;
}

int g = 0;

int f1() { return ++g; }
int f2() { return ++g; }

int main() {
  foo(f1(), f2());
}
```

https://compiler-explorer.com/z/Pxrn4evfP

Is it undefined behavior? No: Stackoverflow - Multiple unsequenced modifications following argument evaluation

Safety Nets

Compiler warnings (-Wall -Wextra)

None…

Clang-Tidy warnings

None…

What can go wrong here?

```
template<class Map, typename Key>
const typename Map::mapped_type& get_or_default(
    const Map& map,
    const Key& key,
    const typename Map::mapped_type& defaultVal
) {
    auto pos = map.find(key);
    return (pos != map.end() ?
            pos->second: defaultVal);
}
```

A the map can be empty

B dangling reference

C inefficiency

D code is too generic

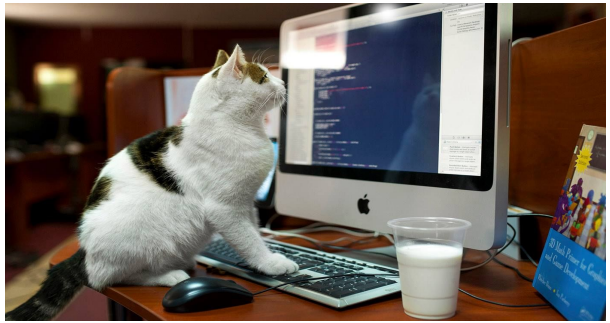


Image Source:
http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

What can go wrong here?

```
template<class Map, typename Key>
const typename Map::mapped_type& get_or_default(
    const Map& map,
    const Key& key,
    const typename Map::mapped_type& defaultVal
) {
    auto pos = map.find(key);
    return (pos != map.end() ?
            pos->second: defaultVal);
}
```

A the map can be empty

B dangling reference

C inefficiency

D code is too generic

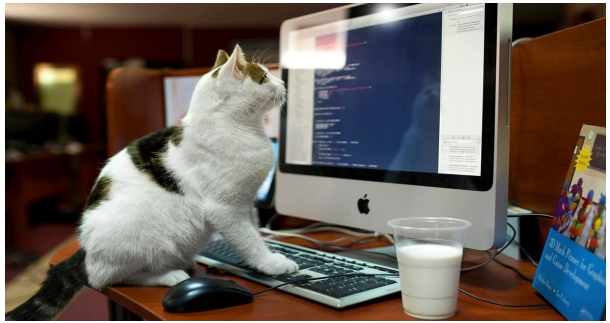


Image Source:
http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

...Beware of your return type!

```
template<class Map, typename Key>
const typename Map::mapped_type& get_or_default(
    const Map& map,
    const Key& key,
    const typename Map::mapped_type& defaultVal
) {
    auto pos = map.find(key);
    return (pos != map.end() ?
            pos->second: defaultVal);
}
```

```
const string& str = get_or_default(mymap, "pikotaro", "pineapple");
std::cout << str;
```

Note that ASAN locates the problem

Code presenting the problem:

http://coliru.stacked-crooked.com/a/e7983b00ebb59520

We can compile the code with ASAN sanitize flag

(see: https://github.com/google/sanitizers/wiki/AddressSanitizer -fsanitize=address)

which identifies the problem right ahead!

http://coliru.stacked-crooked.com/a/74d5b2e2d0876226

And now the problem is fixed!

http://coliru.stacked-crooked.com/a/d6c8516fe362aeae

...Beware of your return type!

The previous issue is taken from the famous CppCon 2017 talk by Louis Brandy:

“Curiously Recurring C++ Bugs at Facebook”

https://www.youtube.com/watch?v=lkgszkPnV8g&t=14m35s

There are a few additional interesting bugs there ^

In exploration of real-world issues

In exploration of real-world issues

C++ Code Issues - Sharing an Example

We are collecting real-world examples of C++ code issues, for research purposes. This includes cases that led to bugs, incorrect behavior, performance problems, or maintainability challenges.
The goal is to identify categories of issues and their mitigations, as well as areas that need more attention in code or tooling.

https://forms.gle/hcouFy1hPWcVnS8p6

What’s the problem with this code?

```
template<std::ranges::input_range Range>
requires std::is_arithmetic_v<std::ranges::range_value_t<Range>>
double average(Range&& r) {
  auto [sum, count] = std::accumulate(std::ranges::begin(r), std::ranges::end(r),
        std::pair<double, std::size_t>{0.0, 0},
        [](auto acc, const auto& value) {
            return std::pair{acc.first + value, acc.second + 1};
        }
  );
  if(count == 0) {
    std::invalid_argument("empty range");
  }
  return sum / count;
}
```

https://compiler-explorer.com/z/frnErKGdG

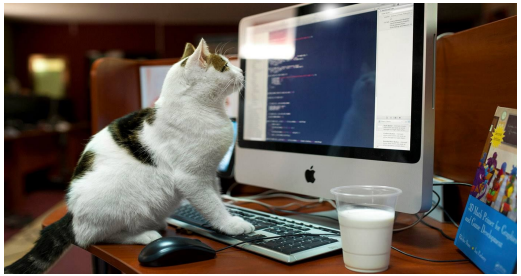


Image Source: http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

What's the problem with this code?

```
template<std::ranges::input_range Range>
requires std::is_arithmetic_v<std::ranges::range_value_t<Range>>
double average(Range&& r) {
  auto [sum, count] = std::accumulate(std::ranges::begin(r), std::ranges::end(r),
        std::pair<double, std::size_t>{0.0, 0},
        [](auto acc, const auto& value) {
            return std::pair{acc.first + value, acc.second + 1};
        }
  );
  if(count == 0) {
    std::invalid_argument("empty range");
  }
  return sum / count;
}
```

hint: problem is here

https://compiler-explorer.com/z/frnErKGdG

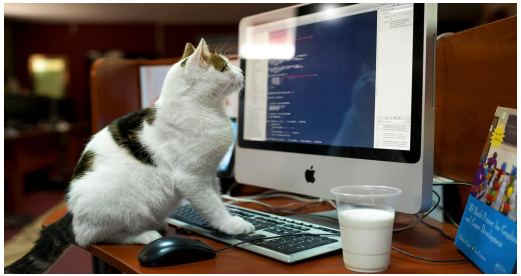


Image Source: http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

What’s the problem with this code?

```
template<std::ranges::input_range Range>
requires std::is_arithmetic_v<std::ranges::range_value_t<Range>>
double average(Range&& r) {
  auto [sum, count] = std::accumulate(std::ranges::begin(r), std::ranges::end(r),
        std::pair<double, std::size_t>{0.0, 0},
        [](auto acc, const auto& value) {
            return std::pair{acc.first + value, acc.second + 1};
        }
  );
  if(count == 0) {
    throw std::invalid_argument("empty range");
  }
  return sum / count;
}
```

missing the “throw” keyword

https://compiler-explorer.com/z/frnErKGdG

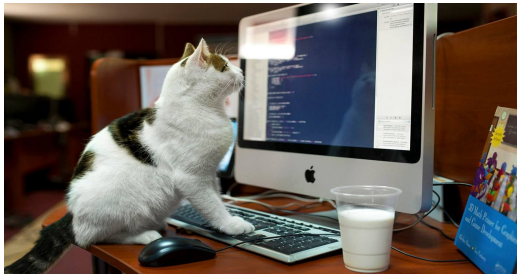


Image Source: http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

Forgetting a “throw”

- No compiler warning
- Static code analysis didn’t catch that
- There wasn’t a test case for the error scenario
- **Was caught in code review**

Both may warn in the future,
Other static analyzer may warn.

Mitigation:

- Fixing the code, adding throw
- Adding a test case for this flow

- aiming for 100% coverage we need this test case anyhow
- check that the expected exception is actually thrown

Contributor: Alex Cohn

What’s the problem with this code?

```
struct A {
  virtual void foo() const { std::cout << "const A\n"; }
  virtual void foo()       { std::cout << "A\n"; }
};

struct B: A {
    void foo() const override { std::cout << "const B\n"; }
};

int main() {
    B b;
    A* pa = &b;
    pa->foo();
}
```

https://compiler-explorer.com/z/7rrq11baj

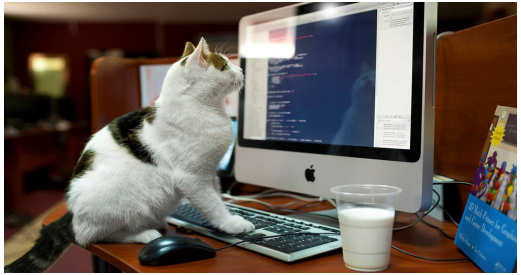


Image Source:
http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

Not overriding all function versions

- No compiler warning
- Static code analysis didn’t catch that
  (In the actual scenario, the static code analyzer was the one to suggest adding the const version...)
- There wasn’t a test case for the error scenario
- Was discovered **in production**!!

Both may warn in the future,
Other static analyzer may warn.

Mitigation:

- Added the non-const override version
- Added a test case for this flow

Other polymorphism issues

- Different default value in virtual functions (code example).
- Forgetting virtual destructor (code example).
- Forgetting adding virtual on base class destructor (code example).
- Calling virtual function from ctor or dtor without knowing the rules (code example).
- Object slicing, e.g. in assignment or comparison (code example).

What’s the problem with this code?

```
struct ChaosCrew {
  virtual ~ChaosCrew() {}
  std::shared_ptr<ChaosCrew> colleague;
};

struct CatInTheHat: public ChaosCrew {};
struct Thing1: public ChaosCrew {};
struct Thing2: public ChaosCrew {};

int main() {
  auto cat = std::make_shared<CatInTheHat>();
  auto thing1 = std::make_shared<Thing1>();
  auto thing2 = std::make_shared<Thing2>();
  cat->colleague = thing1;
  thing1->colleague = thing2;
  thing2->colleague = cat;
}
```

https://compiler-explorer.com/z/d4ErT4vnK

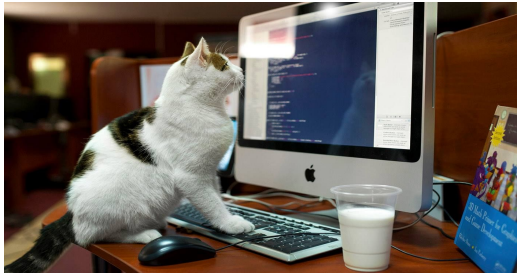


Image Source:
http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

Smart Pointers have their rules

Beware of shared_ptr ownership cycle:

- No compiler warning
- Address sanitizer may catch that

Mitigation:

- Break the cycle – use weak_ptr for at least one in the cycle

There are other pitfalls with smart pointers, most of them can be caught with address sanitizer, some may be caught with static analysis.

What’s the problem with this code?

```
template<typename T>
void writeNumberToBinaryFile(std::ofstream& file, const char* input, bool isBinary) {
  static_assert(std::is_trivially_copyable_v<T>, "T must be trivially copyable");
  const char* output;
  if (isBinary) output = input;
  else {  // Parse input as text, then write the value as binary
    std::istringstream iss(input);
    T value;
    iss >> value;
    if (iss.fail()) throw std::invalid_argument("Failed to parse input as number");
    output = reinterpret_cast<const char*>(&value);
  }
  file.write(output, sizeof(T));
  if (!file) {
    throw std::runtime_error("Failed to write binary data");
  }
}
```

https://compiler-explorer.com/z/Psa6755b3

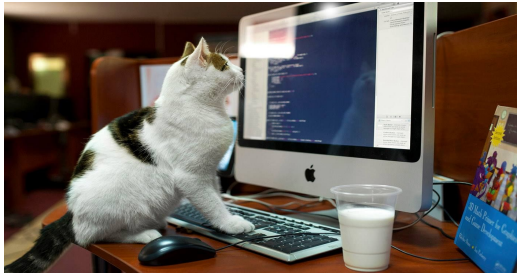


Image Source: http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

Using a dead local variable

- No compiler warning
- Address sanitizer would catch that
- May become a security breach
- **Was actually caught by static analyzer**

May warn in the future

Mitigation:

- Move the local variable up

Not the exact same issue, but resembles:

- Shadowing a local variable (code example)
- An even more bizarre example

What’s the problem with this code?

```
class Widget {
    std::string s = "hi";
public:
    const std::string& getString() const { return s; }
};

void func(const std::string& str, Widget w) {
    std::cout << str << std::endl;
    std::cout << w.getString() << std::endl;
}

int main() {
    Widget w;
    func(w.getString(), std::move(w));
}
```

https://compiler-explorer.com/z/eGhKansjb

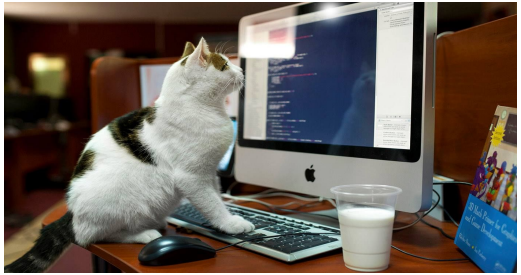


Image Source: http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

And now?

```
class Widget {
    std::string s = "hi";
public:
    std::string getString() const { return s; }
};

void func(const std::string& str, Widget w) {
    std::cout << str << std::endl;
    std::cout << w.getString() << std::endl;
}

int main() {
    Widget w;
    func(w.getString(), std::move(w));
}
```

https://compiler-explorer.com/z/37Ms3r5cj

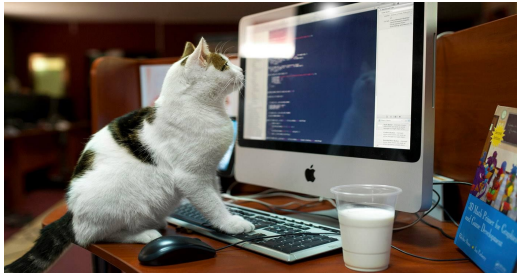


Image Source: http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

Using a moved-from object

- No compiler warning
- Static analyzer didn’t catch it
- Address sanitizer wouldn’t catch it
- **Was caught in testing**

Both may warn in the future,
Other static analyzer may warn.

Mitigation:

- Testing, code review (look for static analyzer that can catch it?)

Same issue as invalidated iterators, dangling pointers and references

Contributor: Tal Jerome

What can be the problem with this code?

```
template<typename Base>
using Registry = std::vector<std::function<unique_ptr<Base>()>>;
Registry<A> registry;
template<typename T> auto registerFactory() {
    // assume single thread
    static auto registration = registry.insert(registry.end(), [](){
        return std::make_unique<T>();
    });
    return registration;
}

int main() {
    registerFactory<A>(); registerFactory<B>();
    // C was assumed to be registered but is not
    registerFactory<C>();
}
```

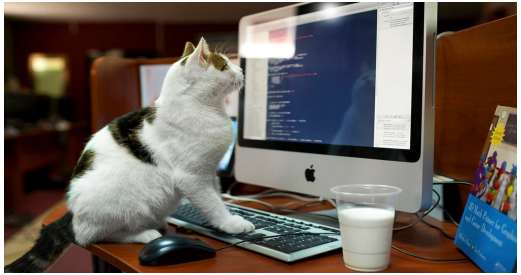


Image Source: http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

What can be the problem with this code?

```
template<typename Base>
using Registry = std::vector<std::function<unique_ptr<Base>()>>;
Registry<A> registry;
template<typename T> auto registerFactory() {
    // assume single thread
    static auto registration = registry.insert(registry.end(), [](){
        return std::make_unique<T>();
    });
    return registration;
}

int main() {
    registerFactory<A>(); registerFactory<B>();
    // C was assumed to be registered but is not
    registerFactory<C>();
}
```

C was an alias (using/typedef) of B

https://compiler-explorer.com/z/14nbGfPxM

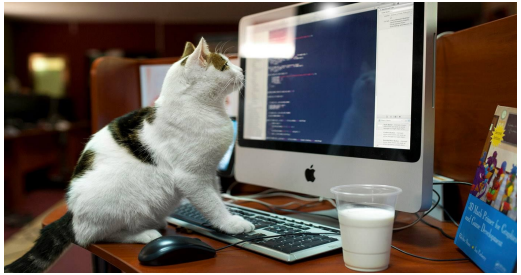


Image Source: http://www.magicindie.com/magicblog/wp-content/uploads/2013/12/cat_programmer.jpg

Assuming Wrong Behaviour

- No compiler warning
- Can’t be catched by compiler nor by static analyzer or dynamic analyzer

Tools cannot read our thoughts...

Mitigation:

- Testing, code review
- Keep it simple - being too innovative in code might be risky

Inspired by a case reported by Ronen Friedman
https://github.com/ceph/ceph/pull/62070

What’s the problem with this code? (1)

```
Direction Algorithm2::objDirection(Obj* obj) {
  int x_diff = obj->getRow() - other_obj->getRow();
  int y_diff = obj->getCol() - other_obj->getCol();
  if (abs(y_diff) > board_height / 2) {
    y_diff = (y_diff > 0) ?
    y_diff - board_height : y_diff + board_height;
  }
  if (std::abs(x_diff) > board_width / 2) {
    x_diff = (x_diff > 0) ?
    x_diff - board_width : x_diff + board_width;
  }
  if (y_diff < 0 && x_diff == 0) {
    return Direction::U; // Up
  } else if (y_diff < 0 && x_diff < 0) {
    return Direction::UR;
  } else if (y_diff == 0 && x_diff < 0) {
    // cont’ on other side ⇒
    return Direction::R; // Left
  } else if (y_diff > 0 && x_diff < 0) {
    return Direction::DR; // Down-Right
  } else if (y_diff > 0 && x_diff == 0) {
    return Direction::D; // Down
  } else if (y_diff > 0 && x_diff > 0) {
    return Direction::DL; // Down-Left
  } else if (y_diff == 0 && x_diff > 0) {
    return Direction::L; // Right
  } else if (y_diff < 0 && x_diff > 0) {
    return Direction::UL; // Up-Right
  }
  return Direction::U;
}
```

What’s the problem with this code? (2)

```
Action Algorithm2::changeDirection(Direction dest) {
  int curr_dir = (int)obj->getDirection();
  int dest_dir = (int)dest;
  int change_dir = curr_dir - dest_dir;
  if (change_dir == 0) {
    return Action::ROTATE_EIGHT_LEFT;
  } else if (change_dir == -1 || change_dir == 7) {
    return Action::ROTATE_EIGHT_RIGHT;
  } else if (change_dir == 1 || change_dir == -7) {
    return Action::ROTATE_EIGHT_LEFT;
  } else if (change_dir == -2 || change_dir == 6) {
    return Action::ROTATE_QUARTER_RIGHT;
  } else if (change_dir == 2 || change_dir == -6) {
    // cont’ on other side ⇒
    return Action::ROTATE_QUARTER_LEFT;
  } else if (change_dir == -3 || change_dir == 5) {
    return Action::ROTATE_QUARTER_RIGHT;
  } else if (change_dir == 3 || change_dir == -5) {
    return Action::ROTATE_QUARTER_LEFT;
  } else {
    return Action::ROTATE_QUARTER_RIGHT;
  }
}
```

Unmaintainable code leads to bugs

- Magic Numbers
- Bad use of enums
- Unclear, overcomplicated, logic

Needs refactoring
CppCon 2017: Mikhail Matrosov "Refactor or die"

Mitigation:

- Unit Testing => Refactoring

What’s the problem with this code?

```
if(!action.isCancel()) {
 if(queue.activeRequests().containsSimilar(action)) return -1; // already processed, see spe
 if(!is_end_of_day) {
   if(approver.isManager()) {
     if(approver.level >= action.approvingLevel()) {
       int prev_rejections = actionsDAO.getPrevRejections(action, request_date);
       if(prev_rejections < MAX_PREV_REJECTIONS)
         queue.addRequest(action, prev_rejections, request_date);
     } else {
       if(prev_rejections < MAX_ESC_PREV_REJECTIONS) {
          User manager = userDAO.getManager(approver);
          approver.escalate(manager, action, prev_rejections, request_date);
       } else {
         // reject request
       }
     } // handle end of day - add action to next day queue, handle cancel request ...
```

State Hell

- Too many nested if levels
- Code is too complicated
- State management becomes a mess

Needs refactoring
CppCon 2017: Mikhail Matrosov "Refactor or die"

Mitigation:

- Unit Testing => Refactoring

...Refactor - Turn “state hell” into State Machine

```
if(!action.isCancel()) {
 if(queue.activeRequests().containsSimilar(action)) return -1; // already processed, see spe
 if(!is_end_of_day) {
   if(approver.isManager()) {
     if(approver.level >= action.approvingLevel()) {
       int prev_rejections = actionsDAO.getPrevRejections(action, request_date);
       if(prev_rejections < MAX_PREV_REJECTIONS)
         queue.addRequest(action, prev_rejections, request_date);
     } else {
       if(prev_rejections < MAX_ESC_PREV_REJECTIONS) {
          User manager = userDAO.getManager(approver);
          approver.escalate(manager, action, prev_rejections, request_date);
       } else {
         // reject request
       }
     } // handle end of day - add action to next day queue, handle cancel request ...
```

...State Machine for managing complex state

A better way to manage complicated states

- Different states - different behavior
- State is is mapped according to the different behavior we wish to model
- Improved communication between Product and Development
- A clear context
    - Better tracing
    - Easier extensibility, adding new behaviors is less bug prone

Other significant issues

Other significant issues (1)

🔌 **API issues**

- API not working as documented.
- Assuming API behavior which is not documented.
- Bad arguments sent:
  - Wrong order
  - Type mismatch leading to undesired casting
  - Wrong measurement units (yes the classic one)
- Not handling error case, ignoring return value.

Other significant issues (1) - Mitigations

🔌 **API issues**

- API not working as documented.
- Assuming API behavior which is not documented.
- Bad arguments sent:
  - Wrong order
  - Type mismatch leading to undesired casting
  - Wrong measurement units (yes the classic one)
- Not handling error case, ignoring return value.

Protective programming: validate that results are reasonable. Integration testing.

"RTFM"

Move to Strong Types

Add unit tests that mock external APIs for all relevant error cases

TESTING

Other significant issues (2) - Mitigations

🗃️ **Bad memory access, invalidated iterators etc.:**

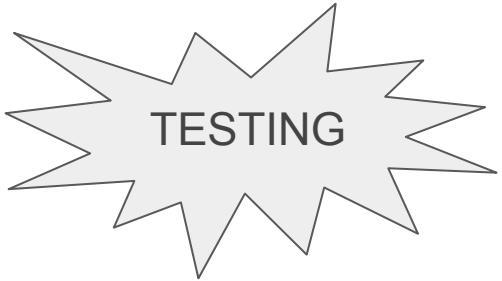
- Using vector iterator retrieved before push_back
- Using vector iterator retrieved before erasing a previous element
- Using unordered_map iterator retrieved before insert
- Accessing vector beyond size limits but within capacity,
  not caught by address sanitizer (old values still there)

Other significant issues (2) - Mitigations

Bad memory access, invalidated iterators etc.:

- Using vector iterator retrieved before push_back
- Using vector iterator retrieved before erasing a previous element
- Using unordered_map iterator retrieved before insert
- Accessing vector beyond size limits but within capacity, not caught by address sanitizer (old values still there)

Code Review,
Best Practices,
Sanitizers,
Fuzz Testing



Other significant issues (3)

- **The language is becoming more complicated**

Other significant issues (3)

The language is becoming more complicated

Most C++ programmers are not like the members of the committee
"average programmer," [...] is seriously underrepresented on the committee.

DIRECTION FOR ISO C++, P2000R4, 15-10-2022,
H. Hinnant, R. Orr, B. Stroustrup, D. Vandevoorde, M. Wong

This "biases the committee towards language lawyering, advanced features, and implementation issues, rather than directly addressing the needs of the mass of C++ developers [...] ". — [...] —
Our expert imbalance also results in over-complicated solutions that require advanced proficiencies for simple tasks. Consider the hoops one needs to jump through to make `std::print` work with a custom type when compared to the old stream operators. —— C++ Should Be C++, P3023R, *David Sankel*, 31-10-2023

C++ Should Be C++ - David Sankel - C++Now 2024 – 56m43s

Other significant issues (3)

- **The language is becoming more complicated**

std::for_each

Defined in header <algorithm>

```
template< class InputIt, class UnaryFunc >
UnaryFunc for_each( InputIt first, InputIt last, UnaryFunc f );
```

(1) (constexpr since C++20)

std::ranges::for_each, std::ranges::for_each_result

Defined in header <algorithm>

Call signature

```
template< std::input_iterator I, std::sentinel_for<I> S, class Proj = std::identity,
          std::indirectly_unary_invocable<std::projected<I, Proj>> Fun >
constexpr for_each_result<I, Fun>
    for_each( I first, S last, Fun f, Proj proj = {} );
```

(1) (since C++20)

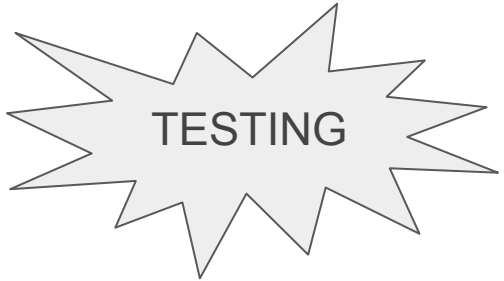
Other significant issues (3) - Mitigations

🕸 The language is becoming more complicated

Yes, but:

- Frameworks may be complicated - BUT you should keep user code simple.
- Don't be tempted by the latest cool feature on the block, use new features only if they provide real value and serve their intended purpose.
- Don’t commit code that you cannot explain.

Keep coherent and simple design



Focus on code readability and maintainability

Summary

Summary (1)

Programming is Tricky, C++ might be even more

🕳️ **Mind the Traps:** be aware of the pitfalls, such as UB, dangling references, slicing, moved-from objects, and more…

👁️ **Be Explicit:** Prefer clarity over cleverness, keep your code simple

- ambiguity and complexity invite bugs

🔒 **Prefer Safe Constructs:** RAII, smart pointers, strong typing, constexpr, small functions, stateless vs stateful.

🔁 **Review & Refactor:** Tech debt and legacy code are bug magnets!

Summary (2)

Make sure to use the right tools 🔧

⚠️ Compiler warnings (at least: -Wall -Wextra -Werror)

🔬 Static analysis tools (enable warnings and try to fix issues)

🕵️ Runtime sanitizers (ASan, UBSan, TSan)

🔍 Code reviews with real blocking power

Summary (3)

Testing, testing, testing

📋 Unit Testing

📊 Real meaningful coverage (aim for 100% branch coverage!)

🎲 Fuzz Testing

🧩 Integration and System Testing

Additional Resources

Curiously Recurring C++ Bugs at Facebook – Louis Brandy – CppCon 2017

Typical Type Typos – Amir Kirsh – ACCU 2021

So You Thought C++ Was Weird? Meet Enums – Roth Michaels – CppCon 2021
(Lightning Talk)

Acknowledgements

Thanks to:
Alex Kushnir, Natanel Hofshi, Udi Lavi, Alex Cohn, Tal Jerome, Eliran Malki, Ronen Friedman and other anonymous contributors, for providing code examples and insights.

Contributions Appreciated

C++ Code Issues - Sharing an Example

We are collecting real-world examples of C++ code issues, for research purposes. This includes cases that led to bugs, incorrect behavior, performance problems, or maintainability challenges.
The goal is to identify categories of issues and their mitigations, as well as areas that need more attention in code or tooling.

https://forms.gle/hcouFy1hPWcVnS8p6

Any questions before we conclude?

Bye

Photo by Howie R on Unsplash

Amir Kirsh
kirshamir@gmail.com

Additional Dark Corners

Is it undefined behavior or illegal, when I use a name that is both already declared outside of the class and later declared inside the class? - Stack Overflow See code: https://compiler-explorer.com/z/qz768f7sh

Leak of cyclic shared_ptrs: https://compiler-explorer.com/z/d4ErT4vnK

Is passing a C++ object into its own constructor legal?

Alignment and Punning: https://blog.hiebl.cc/posts/practical-type-punning-in-cpp/

Slicing: Slicing in the standard library, A constexpr virtual CRTP comparison - C++Online2025