C++ Code Smells

Jason Turner

- Co-host of CppCast https://cppcast.com
- Host of C++ Weekly https://www.youtube.com/c/JasonTurner-lefticus
- Projects
  - https://chaiscript.com
  - https://cppbestpractices.com
  - https://github.com/lefticus/cpp_box
  - https://coloradoplusplus.info
- Microsoft MVP for C++ 2015-present

Jason Turner

Independent and available for training or contracting

- https://articles.emptycrate.com/idocpp

Check out the “North Denver Metro C++ Meetup,” we’ve been meeting consistently since November 2016!

About my Talks

- Move to the front!
- Please interrupt and ask questions
- This is approximately how my training days look

Upcoming Events

- CppCon - Sept 21, 2019 - Applied `constexpr` - Doing More At Compile-Time

C++ Best Practices

C++ Best Practices

C++ Best Practices

- C++ Core Guidelines: 496 (Herb Sutter, Bjarne Stroustrup, et al)

C++ Best Practices

- C++ Core Guidelines: 496 (Herb Sutter, Bjarne Stroustrup, et al)
- Effective Modern C++: 42 (Scott Meyers)

C++ Best Practices

- C++ Core Guidelines: 496 (Herb Sutter, Bjarne Stroustrup, et al)
- Effective Modern C++: 42 (Scott Meyers)
- C++ Best Practices: 109 (Me, et al)

C++ Best Practices

- C++ Core Guidelines: 496 (Herb Sutter, Bjarne Stroustrup, et al)
- Effective Modern C++: 42 (Scott Meyers)
- C++ Best Practices: 109 (Me, et al)
- C++ Coding Standards: 101 (Herb Sutter, Andrei Alexandrescu)

C++ Best Practices

- C++ Core Guidelines: 496 (Herb Sutter, Bjarne Stroustrup, et al)
- Effective Modern C++: 42 (Scott Meyers)
- C++ Best Practices: 109 (Me, et al)
- C++ Coding Standards: 101 (Herb Sutter, Andrei Alexandrescu)

Just from these 4 items: 748 best practices!

C++ Best Practices

This has a lot of questions…

C++ Best Practices

This has a lot of questions…

- How many are unique?

C++ Best Practices

This has a lot of questions…

- How many are unique?
- How many are important?

C++ Best Practices

This has a lot of questions…

- How many are unique?
- How many are important?
- Which ones can tools tell us about?

C++ Best Practices

This has a lot of questions…

- How many are unique?
- How many are important?
- Which ones can tools tell us about?

> *We don’t have to teach things all compilers warn on*

Herb Sutter (CppCon 2018)

Code Smells

Code Smells

Code Smells

- Is it possible to swap these around and look for “smells” instead?

Code Smells

- Is it possible to swap these around and look for “smells” instead?
- Do the smells help us reduce the set of best practices?

Code Smells

- Is it possible to swap these around and look for “smells” instead?
- Do the smells help us reduce the set of best practices?

I asked Twitter for their favorite C++ Code Smells

What Do We Think?

```cpp
#include <string>

void do_work()
{
  std::string str;
  // do some stuff
  str = "Hello World";
  // work with str
}
```

https://godbolt.org/z/7zl9t_

What Do We Think?

```cpp
#include <string>

void do_work()
{
  std::string str; /// construction
  // do some stuff
  str = "Hello World"; /// assignment
  // work with str
}
```

https://godbolt.org/z/baIKi2

Construction Separate From Assignment - Ben Deane

```cpp
#include <string>

void do_work()
{
  // do some stuff
  const std::string str = "Hello World";
  // work with str
}
```

https://godbolt.org/z/mQw9HG

Construction vs Assignment - Ben Deane

What Do We Think?

```cpp
#include <string>

void get_value(std::string &out_param);

int main()
{
  std::string value;
  get_value(value);
  // use value
}
```

https://godbolt.org/z/egT7ec

Out Variables - Ólafur Waage

```cpp
#include <string>

std::string get_value();

int main()
{

  const auto value = get_value();
  // use value
}
```

https://godbolt.org/z/eL1fLw

Out Variables - Ólafur Waage

```cpp
#include <string>

std::string get_value();

int main()
{
  /// How many parameters does this function take?
  const auto value = get_value();
  // use value
}
```

https://godbolt.org/z/4LqxYi

Out Variables - Ólafur Waage

Construction / Assignment / Out Variables

See also this article from Sean Parent:

https://stlab.cc/tips/stop-using-out-arguments.html

What Do We Think?

```cpp
#include <vector>

void process_more(const std::vector<double> &);

void process_data(const std::vector<double> &values) {
  bool in_range = true;
  for (const auto &v : values) {
    if (v < 5.0 || v > 100.0) {
      in_range = false;
      break;
    }
  }

  if (in_range) {
    process_more(values);
  }
}
```

https://godbolt.org/z/PXsqPk

Raw Loops - Sean Parent

```cpp
#include <vector>
#include <algorithm>

void process_more(const std::vector<double> &);

void process_data(const std::vector<double> &values) {
  const auto in_range = [](const double d) {
    return d >= 5.0 && d <= 100.0;
  };

  // this now reads as a sentence
  const bool all_in_range = all_of(begin(values), end(values), in_range);

  if (all_in_range) {
    process_more(values);
  }
}
```

https://godbolt.org/z/JtbNhg

Raw loops don’t express intent, but algorithms can.

Raw Loops - Sean Parent (Not via Twitter Tho)

What Do We Think?

```cpp
double Data::total_area()
{
  int value = 0;

  // step 1: pipe area
  for (int i = 0; i < pipes.size(); ++i) {
    value += pipes[i].radius * pipes[i].radius * M_PI;
  }

  // step 2: hose area
  for (int i = 0; i < hose.size(); ++i) {
    value += hose[i].radius * pipes[i].radius * M_PI;
  }

  // and many more

  return value;
}
```

https://godbolt.org/z/X30YWl

What Do We Think?

```cpp
double Data::total_area()
{
  int value = 0;

  // step 1: pipe area
  for (int i = 0; i < pipes.size(); ++i) {
    value += pipes[i].radius * pipes[i].radius * M_PI;
  }

  // step 2: hose area
  for (int i = 0; i < hose.size(); ++i) {
    value += hose[i].radius * pipes[i].radius * M_PI; ///
  }

  // and many more

  return value;
}
```

https://godbolt.org/z/4ifcyl

Multi-Step Functions

Instead decompose steps into functions and/or lambdas.

```cpp
constexpr double area(const double r) { return r * r * M_PI; }

double Data::total_area()
{
  const auto accumulate_area = [](const auto lhs, const auto rhs) {
    return lhs + area(rhs);
  }

  const auto total_area = [&](const auto &container) {
    return std::accumulate(begin(container), end(container), 0.0, accumulate_area);
  };

  return total_area(pipes) + total_area(hoses) /* + other things */;
}
```

https://godbolt.org/z/AXbKzX

Multi-Step Functions

Are comments necessary in this code?

```cpp
constexpr double area(const double r) { return r * r * M_PI; }

double Data::total_area()
{
  const auto accumulate_area = [](const auto lhs, const auto rhs) {
    return lhs + area(rhs);
  }

  const auto total_area = [&](const auto &container) {
    return std::accumulate(begin(container), end(container), 0.0, accumulate_area);
  };

  return total_area(pipes) + total_area(hoses) /* + other things */;
}
```

https://godbolt.org/z/AXbKzX

Multi-Step Functions - Björn Fahller, Tony Van Eerd & Peter Sommerlad

What Do We Think?

```cpp
struct Data {
  int x;
  int y;

  bool operator==(Data &rhs) {
    return x == rhs.x && y == rhs.y;
  }
};
```

https://godbolt.org/z/KTSfn1

Non-Cannonical Operators

What is the issue with this code?

```cpp
struct Data {
  int x;
  int y;

  bool operator==(const Data &rhs) const { ///
    return x == rhs.x && y == rhs.y;
  }
};
```

https://godbolt.org/z/swKlu5

Non-Cannonical Operators

Conversions

From earlier… what do we see?

```cpp
double Data::total_area()
{
  int value = 0;

  // step 1: pipe area
  for (int i = 0; i < pipes.size(); ++i) {
    value += pipes[i].radius * pipes[i].radius * M_PI;
  }

  // step 2: hose area
  for (int i = 0; i < hose.size(); ++i) {
    value += hose[i].radius * hose[i].radius * M_PI;
  }

  // and many more

  return value;
}
```

https://godbolt.org/z/5csxkf

Conversions

Conversions in at least 1 place, probably 3, loss of data.

```cpp
double Data::total_area()
{
  int value = 0;

  // step 1: pipe area
  for (int i = 0; i < pipes.size(); ++i) {
    value += pipes[i].radius * pipes[i].radius * M_PI; ///
  }

  // step 2: hose area
  for (int i = 0; i < hose.size(); ++i) {
    value += hose[i].radius * hose[i].radius * M_PI; ///
  }

  // and many more

  return value; ///
}
```

https://godbolt.org/z/-7HFjp

What Do We Think?

```cpp
#include <string>

void use_string(const std::string &s);

std::string get_string();

int main()
{
  const std::string str = get_string();
  use_string(str.c_str());
}
```

https://godbolt.org/z/Du2E-i

Code With Implicit Constructors

```cpp
#include <string>

void use_string(const std::string &s);

std::string get_string();

int main()
{
  const std::string str = get_string();
  use_string(str.c_str()); /// string->c_str->string
}
```

https://godbolt.org/z/QVamMx

Always exists in code that’s been refactored over a long period.

Code With Conversions

Who can tell me what `std::move` is?

Code With Conversions

Who can tell me what `std::move` is?

An unconditional cast to an r-value reference of the given type.

Code With Conversions

Who can tell me what `std::move` is?

An unconditional cast to an r-value reference of the given type.

```cpp
std::string s;
std::move(s); /// unconditional cast to `std::string &&`
```

What Do We Think?

```cpp
#include <string>

std::string get_value()
{
  std::string s = "Hello There World";
  return std::move(s);
}
```

https://godbolt.org/z/caqj1e

Code With Conversions

```cpp
#include <string>

std::string get_value()
{
  std::string s = "Hello There World";
  return std::move(s); ///
}
```

https://godbolt.org/z/4peiWk

Pessimizing “return by move” prevents move elision.

Code With Conversions

```cpp
#include <string>

std::string get_value()
{
  std::string s = "Hello There World";
  return std::move(s); ///
}
```

https://godbolt.org/z/4peiWk

Pessimizing “return by move” prevents move elision.

`std::move` is another type of conversion that is a code smell.

What Do We Think?

```cpp
int main()
{
  const int i = 4;
  const_cast<int &>(i) = 13;
  return i; /// what is returned?
}
```

https://godbolt.org/z/LylmN3

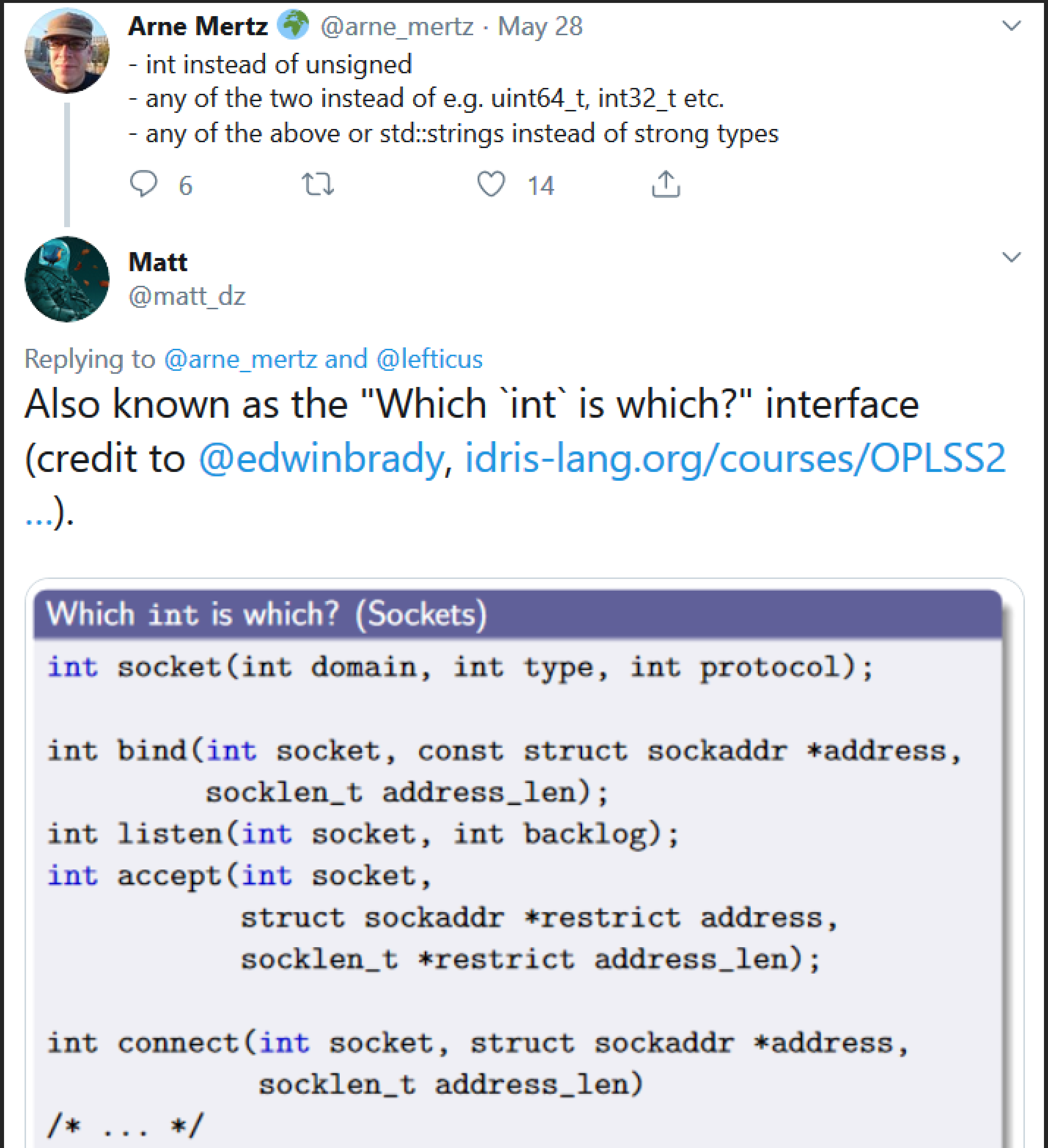
Casting Away `const`

```
int main()
{
  const int i = 4;
  const_cast<int &>(i) = 13;
  return i; /// 4 returned
}
```

https://godbolt.org/z/MIi_T1

Modifying a `const` object during its lifetime is UB. `const_cast` is another explicit conversion that is a code smell.

Weak Typing - or “Which `int` is which?”



Arne Mertz @arne_mertz · May 28

- int instead of unsigned
- any of the two instead of e.g. uint64_t, int32_t etc.
- any of the above or std::strings instead of strong types

Matt
@matt_dz

Replying to @arne_mertz and @lefticus

Also known as the "Which `int` is which?" interface (credit to @edwinbrady, idris-lang.org/courses/OPLSS2 ...).

Which int is which? (Sockets)

```
int socket(int domain, int type, int protocol);

int bind(int socket, const struct sockaddr *address,
         socklen_t address_len);
int listen(int socket, int backlog);
int accept(int socket,
           struct sockaddr *restrict address,
           socklen_t *restrict address_len);

int connect(int socket, struct sockaddr *address,
            socklen_t address_len)
/* ... */
```

Code With Conversions - implicit/explicit/casts (many people)

Warnings

Can our compilers warn us on this code?

```cpp
double Data::total_area()
{
  int value = 0;

  // step 1: pipe area
  for (int i = 0; i < pipes.size(); ++i) {
    value += pipes[i].radius * pipes[i].radius * M_PI; ///
  }

  // step 2: hose area
  for (int i = 0; i < hose.size(); ++i) {
    value += hose[i].radius * pipes[i].radius * M_PI; ///
  }

  // and many more

  return value; ///
}
```

https://godbolt.org/z/nmu738

Warnings

What about here?

```
#include <string>

std::string get_value()
{
  std::string s = "Hello There World";
  return std::move(s);
}
```

https://godbolt.org/z/caqj1e

Code With Warnings - Björn Fahller, Dimitar Mirchev

What Do We Think?

What are the implications of using a `static` variable?

```
#include <string>

void log_error(std::string const &location,
               std::string const &desc);

void do_things(bool const error)
{
  static std::string const FunctionName{"do_things"};

  if (error) {
    log_error(FunctionName, "Error Occured!");
  }
}
```

https://godbolt.org/z/vNItBg

What Do We Think?

What are the implications of using a `static` variable?

```cpp
#include <string>

void log_error(std::string const &location,
               std::string const &desc);

void do_things(bool const error)
{
  static std::string const FunctionName{"do_things"};

  if (error) {
    log_error(FunctionName, "Error Occured!");
  }
}
```

https://godbolt.org/z/vNItBg

Each time the variable is accessed it must be checked to see if it’s been initialized.

`static const`

Compare to this

```cpp
#include <string_view>

void log_error(std::string_view const &location,
               std::string_view const &description);

void do_things(bool const error)
{
  constexpr static std::string_view FunctionName{"do_things"};

  if (error) {
    log_error(FunctionName, "Error Occured!");
  }
}
```

https://godbolt.org/z/QsPDhn

`static const`

Compare to this

```cpp
#include <string_view>

void log_error(std::string_view const &location,
               std::string_view const &description);

void do_things(bool const error)
{
  constexpr static std::string_view FunctionName{"do_things"};

  if (error) {
    log_error(FunctionName, "Error Occured!");
  }
}
```

https://godbolt.org/z/QsPDhn

`static const` is a code smell that should probably should be `constexpr`.

`static const`

What Do We Think?

```cpp
// Data.hpp
extern int const Value;
```

```cpp
// Data.cpp
#include <Data.hpp>
int const Value = 5;
```

```cpp
// Value.cpp
#include <Data.hpp>
int getValue() {
  return Value;
}
```

https://godbolt.org/z/0oUBDO

What Do We Think?

Or simplified:

```
extern int const Value;

int getValue() {
  return Value; ///
}
```

https://godbolt.org/z/H1buHN

extern `const`

It’s like we’re telling the compiler:

> *I have some really important information for you, but I’m not going to tell you what it is.*

This is also a code smell.

extern `const`

It’s like we’re telling the compiler:

> *I have some really important information for you, but I’m not going to tell you what it is.*

This is also a code smell.

What’s the better option?

extern `const`

`constexpr`

```cpp
// Data.hpp
constexpr int Value = 5;

int getValue() {
  return Value;
}
```

https://godbolt.org/z/-5Z1DH

extern `const`

What Do We Think?

```cpp
void use_int()
{
  int *i = new int(5);
  delete i;
}
```

https://godbolt.org/z/qU7C8A

Raw `new` and `delete`

```cpp
#include <memory>

void use_int()
{
  auto i = std::make_unique<int>(5);
}
```

https://godbolt.org/z/OODlM2

Of course this is wasteful and the heap should be avoided if possible.

Raw `new` and `delete`

Code Smells

Code Smells

- Constructions Separate from Assignment

Code Smells

- Constructions Separate from Assignment
- Out Variables

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops
- Multi-Step Functions

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops
- Multi-Step Functions
- Non-Canonical Operators

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops
- Multi-Step Functions
- Non-Canonical Operators
- Code With Conversions

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops
- Multi-Step Functions
- Non-Canonical Operators
- Code With Conversions
- Casting Away `const`

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops
- Multi-Step Functions
- Non-Canonical Operators
- Code With Conversions
- Casting Away `const`
- Code With Warnings

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops
- Multi-Step Functions
- Non-Canonical Operators
- Code With Conversions
- Casting Away `const`
- Code With Warnings
- `static const`

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops
- Multi-Step Functions
- Non-Canonical Operators
- Code With Conversions
- Casting Away `const`
- Code With Warnings
- `static const`
- `extern const`

Code Smells

- Constructions Separate from Assignment
- Out Variables
- Raw Loops
- Multi-Step Functions
- Non-Canonical Operators
- Code With Conversions
- Casting Away `const`
- Code With Warnings
- `static const`
- `extern const`
- Raw `new` and `delete`

Some Refactoring Code Reviews

Let’s Update This Code Sample

```cpp
#include <iostream>
using namespace std;

int main()
{
  int length;
  string greet1 = "Hello";
  string greet2 = ", world!";
  string greet3 = greet1 + greet2;

  length = greet3.size();
}
```

https://godbolt.org/z/hJEDYV

Let’s Update This Code Sample

```cpp
#include <iostream>
using namespace std;

int main()
{
  const string greet1 = "Hello";
  const string greet2 = ", world!";
  const string greet3 = greet1 + greet2;

  const auto length = greet3.size();
}
```

https://godbolt.org/z/eSwuNb

Let’s Update This Code Sample #2

```cpp
#include <iostream>

int main()
{
  int i, n, fact = 1;

  std::cout << "Enter a whole number: ";
  std::cin >> n;

  for (i = 1; i <= n; ++i) {
    fact *= i;
  }

  std::cout << "\nFactorial of " << n << " = " << fact << std::endl;
  return 0;
}
```

https://godbolt.org/z/Q2D71b

Let's Update This Code Sample #2

```
#include <iostream>

template<typename Type> Type read_input() {
  Type obj;
  std::cin >> obj;
  return obj;
}
constexpr int factorial(int value) {
  int result = 1;
  while (value > 0) {
    result *= value;
    --value;
  }
  return result;
}

int main() {
  std::cout << "Enter a whole number: ";

  const auto n = read_input<int>();
  const auto fact = factorial(n);

  std::cout << "\nFactorial of " << n << " = " << fact << '\n';
  return EXIT_SUCCESS;
}
```

https://godbolt.org/z/IfdGtn

Conclusions

There Is One Thing That Keeps Coming Up

…That Hasn't Been Explicitly Mentioned

Review

Construction Separate From Assignment

```cpp
void do_work() {
  std::string str;
  str = "Hello World";
}
```

https://godbolt.org/z/sEyCts

vs

```cpp
void do_work() {
  const std::string str = "Hello World";
}
```

Out Variables

```cpp
void get_value(std::string &out_param);

int main() {
  std::string value;
  get_value(value);
}
```

https://godbolt.org/z/doe0q8

vs.

```cpp
std::string get_value();

int main() {
  const auto value = get_value();
}
```

https://godbolt.org/z/Sb44wG

Raw Loops

```cpp
void process_data(const std::vector<double> &values) {
  bool in_range = true;
  for (const auto &v : values) {
    if (v < 5.0 || v > 100.0) {
      in_range = false;
      break;
    }
  }
  if (in_range) {
    process_more(values);
  }
}
```

https://godbolt.org/z/ugp5Sz

vs

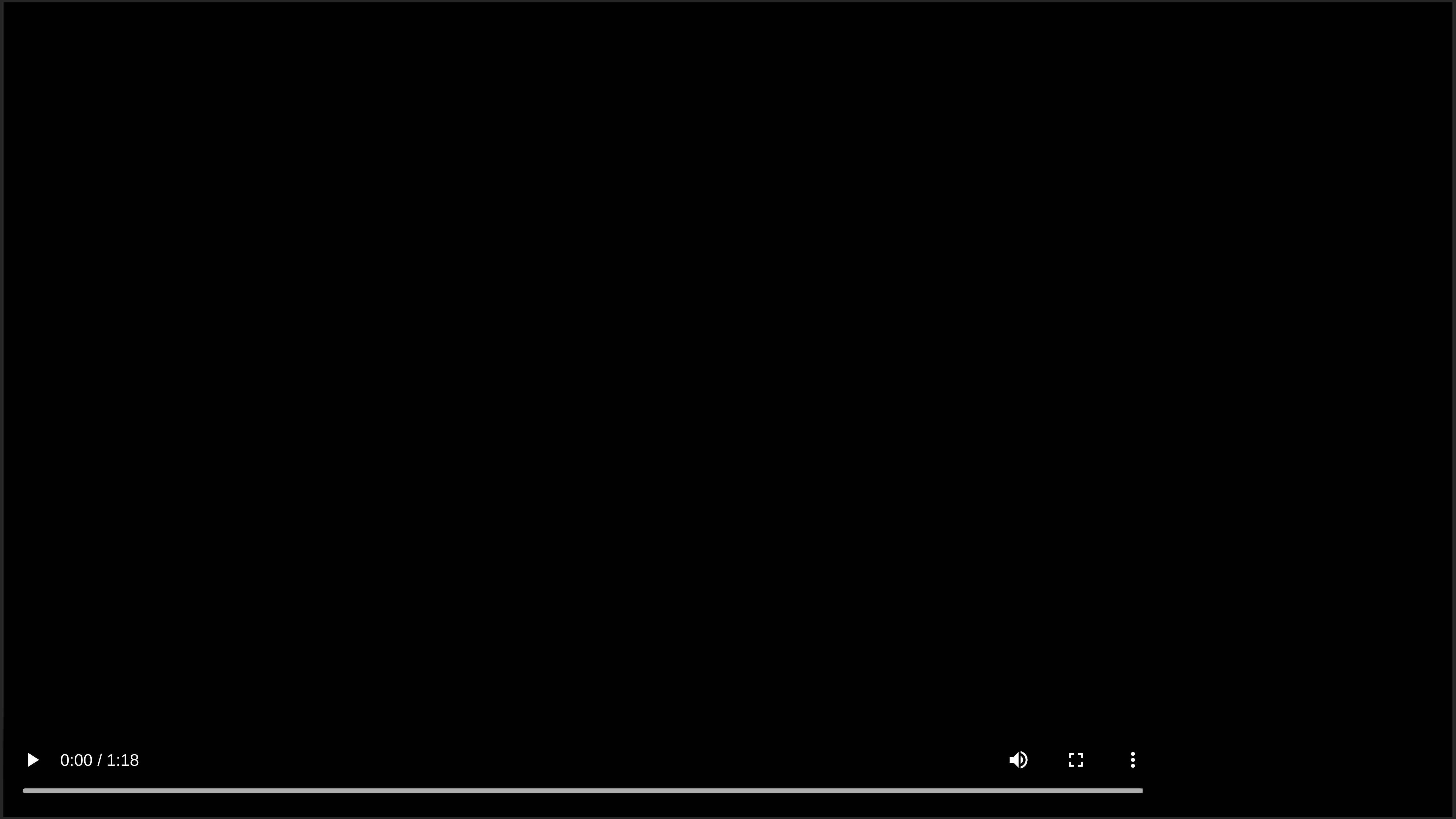
```cpp
void process_data(const std::vector<double> &values) {
  const auto in_range = [](const double d) {
    return d >= 5.0 && d <= 100.0;
  };
  const bool all_in_range = all_of(begin(values), end(values), in_range);
  if (all_in_range) {
    process_more(values);
  }
}
```

https://godbolt.org/z/SHzs6h

What Kept Coming Up?

`const`

It’s not like this is the first time `const` has been mentioned at a conference


0:00 / 1:18

`const`

- Any lack of `const` is a code smell
- `const` forces us into more organized code
- `const` prevents common errors
- `const` encourages more use of algorithms

Do You `const` Value Parameters?

```cpp
#include <cstdio>

void hello_world(int count)
{
  for (int i = 0; i < count; ++count) {
    puts("Hello World");
  }
}
```

https://godbolt.org/z/xQ6MfK

Do You `const` Value Parameters?

```cpp
#include <cstdio>

void hello_world(int count)
{
  for (int i = 0; i < count; ++count) { ///
    puts("Hello World");
  }
}
```

https://godbolt.org/z/WB6XyB

This is a real error I’ve made before.

Do You `const` Value Parameters?

```cpp
#include <cstdio>

void hello_world(const int count) /// fails to compile
{
  for (int i = 0; i < count; ++count) {
    puts("Hello World");
  }
}
```

https://godbolt.org/z/y65vs4

Do You `const` Temporary Values?

```cpp
#include <string>

void consume_string(std::string);

int main()
{
  const std::string str = "Hello World";
  consume_string(std::move(str));
}
```

https://godbolt.org/z/P-cn5P

Do You `const` Temporary Values?

```cpp
#include <string>

void consume_string(std::string);

int main()
{
  const std::string str = "Hello World";
  consume_string(std::move(str)); /// silently reverts to copy
}
```

https://godbolt.org/z/f2S-JJ

Do You `const` Temporary Values?

```cpp
#include <string>

void consume_string(std::string);

int main()
{
  std::string str = "Hello World"; /// :( not const
  consume_string(std::move(str));
}
```

https://godbolt.org/z/nZZYub

How do we resolve this problem?

Do You `const` Temporary Values?

```cpp
#include <string>

void consume_string(std::string);

int main()
{
  consume_string("Hello World");
}
```

https://godbolt.org/z/NgwJ0a

How do we resolve this problem?

Do You `const` Temporary Values?

```cpp
#include <string>

void consume_string(std::string);

int main()
{
  consume_string("Hello World"); /// avoid temporary
}
```

https://godbolt.org/z/Gb5atn

Do You `const` Temporary Values?

```cpp
#include <string>

void consume_string(std::string);
std::string get_string();

int main()
{
  consume_string(get_string()); /// write a function!
}
```

https://godbolt.org/z/OvSh39

Do You `const` Returned Objects?

```cpp
#include <string>

std::string get_string()
{
  const std::string value{"Hello World"};
  return value; /// is this OK?
}
```

https://godbolt.org/z/6nr0bT

Do You `const` Returned Objects?

```cpp
#include <string>

std::string get_string(const bool hello)
{
  const std::string value{"Hello"};
  const std::string value2{"World"};

  if (hello) {
    return value; /// is this OK?
  } else {
    return value2;
  }
}
```

https://godbolt.org/z/9F1l2o

Do You `const` Returned Objects?

```cpp
#include <string>

std::string get_string(const bool hello)
{
  const std::string value{"Hello"};
  const std::string value2{"World"};

  if (hello) {
    return value; /// is this OK?
  } else {
    return value2;
  }
}
```

https://godbolt.org/z/9F1l2o

NRVO likely does not apply here.

Do You `const` Returned Objects?

```cpp
#include <string>

std::string get_string(const bool hello)
{
  const std::string value{"Hello"};
  const std::string value2{"World"};

  if (hello) {
    return std::move(value); /// equiv
  } else {
    return std::move(value2); /// equiv
  }
}
```

https://godbolt.org/z/164dsc

Without NRVO, this is implicitly a `move`. How do we resolve this problem?

Do You `const` Returned Objects?

```cpp
#include <string>

std::string get_string(const bool hello)
{
  if (hello) {
    const std::string value{"Hello"};
    return value;
  } else {
    const std::string value2{"World"};
    return value2;
  }
}
```

https://godbolt.org/z/MFDs_M

Do You `const` Returned Objects?

```cpp
#include <string>

std::string get_string(const bool hello)
{
  if (hello) {
    return "Hello";
  } else {
    return "World";
  }
}
```

https://godbolt.org/z/JvqoUu

Avoiding the temporary so we don’t have to worry about it gives us the optimal solution.

`std::move` With Returned Values

I found many examples like this in LLVM while preparing for this talk:

```cpp
auto do_things() {
  std::unique_ptr<LTOModule> Ret(new LTOModule(std::move(M), Buffer, target));
  Ret->parseSymbols();
  Ret->parseMetadata();
  return std::move(Ret);
}
```

https://godbolt.org/z/mKl4fo

`std::move` With Returned Values

I found many examples like this in LLVM while preparing for this talk:

```cpp
auto do_things() {
  std::unique_ptr<LTOModule> Ret(new LTOModule(std::move(M), Buffer, target));
  Ret->parseSymbols();
  Ret->parseMetadata();
  return std::move(Ret);
}
```

https://godbolt.org/z/mKl4fo

This does generate a warning…

`std::move` With Returned Values

I found many examples like this in LLVM while preparing for this talk:

```
auto do_things() {
  std::unique_ptr<LTOModule> Ret(new LTOModule(std::move(M), Buffer, target));
  Ret->parseSymbols();
  Ret->parseMetadata();
  return std::move(Ret);
}                                                           https://godbolt.org/z/mKl4fo
```

This does generate a warning…

“Redundant move in return statement”

Do You `const` Value Return Types?

Do You `const` Value Return Types?

```cpp
#include <string>

const std::string get_value() {
  return "Hello There World!";
}
```

https://godbolt.org/z/rE_vYp

Do You `const` Value Return Types?

```cpp
#include <string>

std::string get_value() {
  return "Hello There World!";
}

int main()
{
  get_value() += "Weird...";
}
```

https://godbolt.org/z/Si8VcC

Do you want to forbid this?

Do You `const` Value Return Types?

```cpp
#include <string>

const std::string get_value() { ///
  return "Hello There World!";
}

int main()
{
  get_value() += "Weird..."; // fails to compile...
}
```

https://godbolt.org/z/Asn9Qm

Do You `const` Value Return Types?

```cpp
#include <string>

int get_value() { ///
  return 5;
}

int main()
{
  get_value() += 10; /// not allowed on built in types
}
```

https://godbolt.org/z/EKKsYB

Do You `const` Value Return Types?

```cpp
#include <string>

std::string get_value() {
  return "Hello There World";
}

int main()
{
  std::string s;
  s = get_value(); /// move-assignment
}
```

https://godbolt.org/z/5i3_li

Do You `const` Value Return Types?

```cpp
#include <string>

const std::string get_value() { ///
  return "Hello There World";
}

int main()
{
  std::string s;
  s = get_value(); /// copy-assignment
                   /// (you cannot move from `const`)
}
```

https://godbolt.org/z/xaTgFb

Do You `const` Value Return Types?

- You don’t want to `const` value return types, it breaks move operations
- Of course we could likely have rewritten this code so the issue didn’t even come up…

Do You `const` Value Return Types?

- You don't want to `const` value return types, it breaks move operations
- Of course we could likely have rewritten this code so the issue didn't even come up…

```cpp
#include <string>

const std::string get_value() { /// (but still not a good idea)
  return "Hello There World";
}

int main()
{
  // not copy or move!
  const auto s = get_value();
}
```

https://godbolt.org/z/5Wl9K3

We Find 3 Smells:

1. Missing and Ignored Compiler Warnings

Special checks for many of these things.

1. Missing and Ignored Compiler Warnings

Special checks for many of these things.

- cppcheck can help you reduce variable scope

1. Missing and Ignored Compiler Warnings

Special checks for many of these things.

- cppcheck can help you reduce variable scope
- "variable can be `const` from various tools

1. Missing and Ignored Compiler Warnings

Special checks for many of these things.

- cppcheck can help you reduce variable scope
- "variable can be `const` from various tools
- C++ Core Guidelines checks reduce raw pointer / memory usage

1. Missing and Ignored Compiler Warnings

Special checks for many of these things.

- cppcheck can help you reduce variable scope
- "variable can be `const` from various tools
- C++ Core Guidelines checks reduce raw pointer / memory usage
- “Pessimizing `move`” warnings

1. Missing and Ignored Compiler Warnings

Special checks for many of these things.

- cppcheck can help you reduce variable scope
- "variable can be `const` from various tools
- C++ Core Guidelines checks reduce raw pointer / memory usage
- “Pessimizing `move`” warnings
- `const` return values in clang-tidy

2. Missing `const` and `constexpr`, Misplaced `const`

- Why isn't that value or member function `const`?
- If it's known at compile time it should be `constexpr` or an `enum`.

This forces us into more efficient and more organized code, utilizing:

- `std::array`
- `<algorithm>`
- `<numeric>`

> *"east `const`? west `const`? I don't care, just use `const`!"*

I'm not a AAA fan, but it does push us in the same direction as `const`.

3. Weak Types And Casts

Unfortunately the C++ standard library does not help us here:

`string`, `filesystem::path`, `const char *`, and `string_view` have many conversions with `optional`, `variant`, and `shared_ptr` contributing to the issues with non-`explicit` constructors.

- Use stronger typing
- See #1 and #2, they catch some
- For the rest
  - read the code
  - use `auto`
  - Use the correct types to avoid casting
  - avoid named temporaries to avoid `std::move`

Bonus Code Review

```cpp
#include <vector>
#include <limits>

int range(std::vector<int> &values)
{
  int min = std::numeric_limits<int>::max();
  int max = std::numeric_limits<int>::min();

  for (int i = 0; i < values.size(); ++i) {
    if (values[i] < min) {
      min = values[i];
    }
    if (values[i] > max) {
      max = values[i];
    }
  }

  return max - min;
}
```

https://godbolt.org/z/WbJty0

Bonus Code Review

Bonus Code Review

```cpp
#include <algorithm>

template<typename Itr>
auto range(const Itr begin, const Itr end)
{
  const auto [min_elem, max_elem] =
    std::minmax_element(begin, end);

  return *max_elem - *min_elem;
}
```

https://godbolt.org/z/BmUPbf

Jason Turner

- Co-host of CppCast https://cppcast.com
- Host of C++ Weekly https://www.youtube.com/c/JasonTurner-lefticus
- Projects
  - https://chaiscript.com
  - https://cppbestpractices.com
  - https://github.com/lefticus/cpp_box
  - https://coloradoplusplus.info
- Microsoft MVP for C++ 2015-present

Jason Turner

Independent and available for training or contracting

- https://articles.emptycrate.com/idocpp

Check out the “North Denver Metro C++ Meetup,” we’ve been meeting consistently since November 2016!