What Is Good C++?

Jason Turner

- Co-host of CppCast http://cppcast.com
- Host of C++ Weekly https://www.youtube.com/c/JasonTurner-lefticus
- Co-creator of ChaiScript http://chaiscript.com
- Curator of http://cppbestpractices.com
- Microsoft MVP for C++ 2015-present

Jason Turner

Independent and available for training or contracting

- http://articles.emptycrate.com/idocpp
- http://articles.emptycrate.com/training.html
- Next training: 2 days of best practices @ CppCon 2017

Why Do You Use C++ In Your Organization?

Why Do You Use C++ In Your Organization?

- Performance?
- Correctness?
- Compile time?!

Why I use C++

This summation of integers:

```
1  unsigned int sum(const unsigned int range)
2  {
3    unsigned int val = 0;
4    for (unsigned int i = 1; i < range + 1; ++i)
5    {
6      val += i;
7    }
8    return val;
9  }
```

Why I use C++

Compiles to (with clang):

```
sum(unsigned int):  # @sum(unsigned int)
        lea     ecx, [rdi + 1]
        xor     eax, eax
        cmp     ecx, 2
        jb      .LBB0_2
        lea     eax, [rdi - 1]
        lea     ecx, [rdi - 2]
        imul    rcx, rax
        shr     rcx
        lea     eax, [rcx + 2*rdi - 1]
.LBB0_2:                # %for.cond.cleanup
        ret
```

What Compilers Do You Use?

- clang?
- gcc?
- msvc?
- something else?

What Standards Do You Use?

- Pre-C++98
- C++98
- C++03
- C++11
- C++14
- C++17
- C++20?

On C++

C++: an octopus made by nailing extra legs onto a dog.

Steve Taylor

When your hammer is C++, everything begins to look like a thumb.

Steve Haflich (December 1994)

> *Writing in C or C++ is like running a chain saw with all the safety guards removed,*

Bob Gray

> *C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off*

Bjarne Stroustrup (~1986)

Within C++, there is a much smaller and cleaner language struggling to get out

Bjarne Stroustrup (The Design and Evolution of C++)

Is This Good C++?

Writing Good C++

For many years C++ stagnated, but since 2011 we have seen rapid changes with major changes every 3 years.

How do we keep up with these changes and learn to recognize (and write) good C++?

Is This Good C++?

```
1  void assign_value(int *ptr) {
2    *ptr = 42;
3  }
```

Is This Good C++?

```
1  void assign_value(int *ptr) {
2    *ptr = 42;
3  }
4
5  int main() {
6    assign_value(nullptr);
7  }
```

Is This Good C++?

```
1  void assign_value(int *ptr) {
2    assert(ptr);
3    *ptr = 42;
4  }
5
6  int main() {
7    assign_value(nullptr);
8  }
```

Is This Good C++?

```
1  void assign_value(int *ptr) {
2    assert(ptr); /// what about release builds?
3    *ptr = 42;
4  }
5
6  int main() {
7    assign_value(nullptr);
8  }
```

Is This Good C++?

```
1  void assign_value(int *ptr) {
2    if (ptr) {
3      *ptr = 42;
4    } else {
5      // throw error?
6    }
7  }
8
9  int main() {
10   assign_value(nullptr);
11 }
```

Is This Good C++?

```
1  void assign_value(int &ptr) {
2    ptr = 42;
3  }
4
5  int main() {
6    assign_value(nullptr); /// compilation error
7  }
```

Is This Good C++? No.

- Bad - dereference of `nullptr`

Is This Good C++?

```
1  #include <cstdlib>
2
3  int main(const int argc, const char *[])
4  {
5    if (argc > 2)
6      std::cout << "Error: Not Enough Args\n";
7      return EXIT_FAILURE;
8
9    return EXIT_SUCCESS;
10 }
```

Is This Good C++?

```
1  #include <cstdlib>
2
3  int main(const int argc, const char *[])
4  {
5    if (argc > 2) {
6      std::cout << "Error: Not Enough Args\n";
7      return EXIT_FAILURE;
8    }
9
10   return EXIT_SUCCESS;
11 }
```

Is This Good C++? No.

- Bad - missing braces led to confusing indentation and code layout

Is This Good C++?

```
1  int main()
2  {
3    // print counting down from 100 to 0
4    for (unsigned int i = 100; i >= 0; --i)
5    {
6      std::cout << i << '\n';
7    }
8  }
```

Is This Good C++?

```
1  int main()
2  {
3    // print counting down from 100 to 0
4    for (int i = 100; i >= 0; --i) /// a solution?
5    {
6      std::cout << i << '\n';
7    }
8  }
```

Is This Good C++? No.

- Bad - infinite loop caused by unsigned integer math

Is This Good C++?

On the topic of loops

```
1  void print_vector(std::vector<int> t_vec) {
2    for (std::vector<int>::const_iterator itr = t_vec.begin();
3         itr != t_vec.end();
4         ++itr) {
5      std::cout << *itr;
6    }
7  }
8
9  int main() {
10   print_vector({1,2,3,4});
11 }
```

Is This Good C++?

In C++11 this should be

```
1  void print_vector(std::vector<int> t_vec) {
2    for (const auto &v : t_vec) {
3      std::cout << v;
4    }
5  }
6
7  int main() {
8    print_vector({1,2,3,4});
9  }
```

Is This Good C++?

And also?

```
1  void print_vector(const std::vector<int> &t_vec) { ///
2    for (const auto &v : t_vec) {
3      std::cout << v;
4    }
5  }
6
7  int main() {
8    print_vector({1,2,3,4});
9  }
```

Is This Good C++? No.

- Bad - copy of expensive vector
- Not-good - overly verbose loop construct that is prone to typos

Is This Good C++?

```
1  std::int8_t shift_8_times(std::int8_t val)
2  {
3    val <<= 8;
4    return val;
5  }
```

Is This Good C++?

```
1  std::int8_t shift_8_times(std::int8_t val)
2  {
3    val <<= 8; /// undefined behavior
4    return val;
5  }
```

Is This Good C++? No.

- Bad - invoking C++ undefined behavior
- Potentially Worse - invoking CPU undefined behavior

Is This Good C++?

```
1  struct Data {
2    int x; int y;
3    std::string name;
4    ~Data() {}
5  };
6  int main() {
7    std::vector<Data> vec;
8    const Data d{1,2,"Object"};
9    vec.push_back(std::move(d));
10 }
```

Is This Good C++?

```
1  struct Data {
2    int x; int y;
3    std::string name;
4    ~Data() {} /// silently disabled moving
5  };
6  int main() {
7    std::vector<Data> vec;
8    const Data d{1,2,"Object"};
9    vec.push_back(std::move(d));
10 }
```

Is This Good C++?

```
1  struct Data {
2    int x; int y;
3    std::string name;
4  };
5  int main() {
6    std::vector<Data> vec;
7    const Data d{1,2,"Object"};
8    vec.push_back(std::move(d)); /// cannot move const obj
9  }
```

Is This Good C++?

```
1  struct Data {
2    int x; int y;
3    std::string name;
4  };
5  int main() {
6    std::vector<Data> vec;
7    Data d{1,2,"Object"};
8    vec.push_back(std::move(d));
9  }
```

Is This Good C++?

```
1  struct Data {
2    int x; int y;
3    std::string name;
4  };
5  int main() {
6    std::vector<Data> vec;
7    Data d{1,2,"Object"};
8    vec.push_back(std::move(d));
9    d.x = 15;
10 }
```

Is This Good C++?

```
1  struct Data {
2    int x; int y;
3    std::string name;
4  };
5  int main() {
6    std::vector<Data> vec;
7    Data d{1,2,"Object"};
8    vec.push_back(std::move(d));
9    d.x = 15; /// using moved-from object
10 }
```

Is This Good C++?

```
1  struct Data {
2    int x; int y;
3    std::string name;
4  };
5  int main() {
6    std::vector<Data> vec;
7    vec.push_back({1,2,"Object"});
8  }
```

Is This Good C++? No.

- Bad - silent disabling of move operations
- Bad - silent reverting of move to copy of const object
- Bad - risk of using moved-from object

Use The Tools Available

Your compiler

- Modern compilers do complex analysis of source code
- Turn the warnings up as high as possible

Your compiler

- MSVC: `/W4` (`/Wall` not recommended)
- clang/GCC:
  - `-Wall` `-Wextra` `-Wshadow` `-Wnon-virtual-dtor`
  - `-Wold-style-cast` `-Wcast-align` `-Wunused`
  - `-Woverloaded-virtual` `—pedantic` `-Wsign-conversion`
  - `-Wmisleading-indentation`
- clang:
  - Consider `-Weverything` `-Wno-c++98-compat`

Static analyzers

- Perform an analysis of source code without actually executing the code
- Includes the sophisticated analysis and warnings that modern compilers perform

Static analyzers

- cppcheck
- MSVC `/analyze`
- clang
  - -format
  - -check
  - -tidy
- metrix++
- … and more

Runtime analyzers

- Sanitizers
  - Address
  - Thread
  - Memory
  - UndefinedBehavior
  - Leak
- Valgrind
  - Helgrind (Thread error detector)
  - Massif (Heap Profiler)
  - Memcheck
- Checked STL implementation / electric fence

Fuzzy Testing

- Attempts to generate new/novel tests for your code
- Works primarily with projects that accept input files
- ‘Breeds’ new inputs by analyzing code paths

Fuzzy Testers

- libfuzzer
- `-fsanitizer=fuzzer`
- AFL American Fuzzy Lop

`nullptr` dereference

```
1  void assign_value(int *ptr) {
2    *ptr = 42;
3  }
4
5  int main() {
6    assign_value(nullptr);
7  }
```

`nullptr` dereference

- Cppcheck: Possible null pointer dereference: ptr
- clang-tidy: Dereference of null pointer (loaded from variable'ptr')

Confusing Indentation

```
1  #include <cstdlib>
2
3  int main(const int argc, const char *[])
4  {
5    if (argc > 2)
6      std::cout << "Error: Not Enough Args\n";
7      return EXIT_FAILURE;
8
9    return EXIT_SUCCESS;
10 }
```

Confusing Indentation

- g++: this `if` clause does not guard...
- Cppcheck: Consecutive return, break, continue, goto or throw statements are unnecessary
- clang-tidy: statement should be inside braces

Unsigned / Signed Comparison

```
1  int main()
2  {
3    // print counting down from 100 to 0
4    for (unsigned int i = 100; i >= 0; --i)
5    {
6      std::cout << i << '\n';
7    }
8  }
```

Unsigned / Signed Comparison

- clang-tidy: comparison of unsigned expression >= 0 is always true
- g++: comparison of unsigned expression >= 0 is always true
- clang: comparison of unsigned expression >= 0 is always true
- Cppcheck: Unsigned variable `i` can't be negative so it is unnecessary to test it.

Outdated Loop Style

```
1  void print_vector(std::vector<int> t_vec) {
2    for (std::vector<int>::const_iterator itr = t_vec.begin();
3         itr != t_vec.end();
4         ++itr) {
5      std::cout << *itr;
6    }
7  }
8
9  int main() {
10   print_vector({1,2,3,4});
11 }
```

Outdated Loop Style

- clang-tidy: use range-based for loop instead
- clang-tidy: use auto when declaring iterators

Shifting > Size of Int

```
1  std::int8_t shift_8_times(std::int8_t val)
2  {
3    val <<= 8; /// undefined behavior
4    return val;
5  }
```

Shifting > Size of Int

- clang-tidy: shift count >= width of type
- clang: shift count >= width of type
- g++: conversion to 'int8_t {aka signed char}' from `int` may alter its value
- Cppcheck: function `shift_8_times` is never used.

Inefficient / Disabled Moves

```
1  struct Data {
2    int x; int y;
3    std::string name;
4    ~Data() {}
5  };
6  int main() {
7    std::vector<Data> vec;
8    const Data d{1,2,"Object"};
9    vec.push_back(std::move(d));
10 }
```

Inefficient / Disabled Moves

- clang: definition of implicit copy constructor for `Data` is deprecated because it has a user-defined destructor
- clang-tidy: definition of implicit copy constructor for `Data` is deprecated because it has a user-defined destructor
- clang-tidy: class `Data` defines a non-default destructor but does not define a copy constructor, a copy assignment operator, a move constructor or a move assignment operator
- clang-tidy: use `= default` to define a trivial destructor
- clang-tidy: `std::move` of the `const` variable 'd' has no effect; remove `std::move()` or make the variable non-`const`

Use After Move

```
1  struct Data {
2    int x; int y;
3    std::string name;
4  };
5  int main() {
6    std::vector<Data> vec;
7    Data d{1,2,"Object"};
8    vec.push_back(std::move(d));
9    d.x = 15; /// using moved-from object
10 }
```

Use After Move

- clang-tidy: `d` is used after it was moved
- Cppcheck: struct member `Data::y` is never used.
- Cppcheck: Access of moved variable `d`.

Comparison of Constant Expressions

```
1  int main()
2  {
3    if (sizeof(int) < sizeof(long)) {
4      std::cout << "sizeof(int) < sizeof(long)\n";
5    }
6  }
```

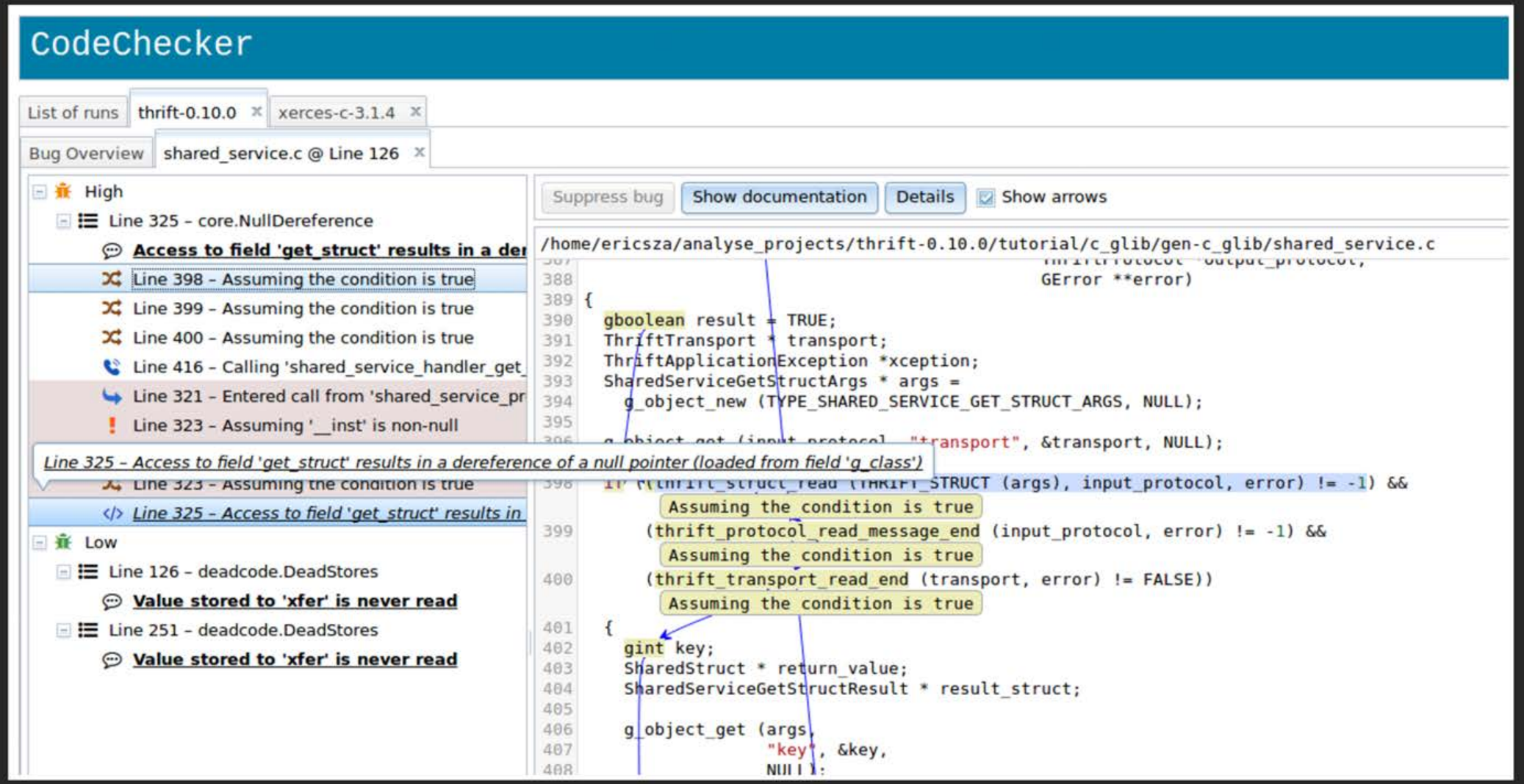
Comparison of Constant Expressions

- Upcoming update to Visual Studio will tell you to convert this to an `if constexpr` construct

Static Analysis

- Find (and fix) common problems
- Find (and upgrade) newer techniques that can be applied
- Report on misuse of newer C++ features

Ericsson CodeChecker



Sanitizers

Thoughts on this code?

```
1  std::size_t find_newline(const std::string &t_str,
2                           std::size_t start)
3  {
4    while (t_str[start] != '\n') {
5      ++start;
6    }
7    return start;
8  }
9
10 int main() {
11   return static_cast<int>(find_newline("Hello\nWorld", 2));
12 }
```

Sanitizers

What about this version?

```
1  std::size_t find_newline(const std::string &t_str,
2                           std::size_t start)
3  {
4    while (t_str[start] != '\n') {
5      ++start;
6    }
7    return start;
8  }
9
10 int main() {
11   return static_cast<int>(find_newline("Hello World", 2));
12 }
```

Sanitizers

Randomly may or may not crash

```
1  std::size_t find_newline(const std::string &t_str, std::size_t start)
2  {
3    while (t_str[start] != '\n') {
4      ++start;
5    }
6    return start;
7  }
8
9  int main() {
10   return static_cast<int>(find_newline("Hello World", 2));
11 }
```

Sanitizers

```
clang++ -fsanitizer=address parser.cpp
```

```
std::size_t find_newline(const std::string &t_str, std::size_t start)
{
  while (t_str[start] != '\n') {
    ++start;
  }
  return start;
}

int main() {
  return static_cast<int>(find_newline("Hello World", 2));
}
```

Sanitizers

- Causes a runtime error when a sanitizer rule has been violated
- Makes finding intermittent / random crashes much easier

Fuzzer

```
clang++ -fsanitizer=fuzzer,address parser.cpp TESTCORPUS
```

```
std::size_t find_newline(const std::string &t_str, std::size_t start)
{
  while (t_str[start] != '\n') {
    ++start;
  }
  return start;
}

extern "C" int LLVMFuzzerTestOneInput(const std::uint8_t *Data,
                                      size_t Size)
{
  find_newline(std::string(reinterpret_cast<const char *>(Data), Size),
               0);
  return 0;  // Non-zero return values are reserved for future use.
}
```

Fuzzer

- Automatically generates new inputs to your program
- When combined with a sanitizer (or two) crashes and generates a back trace
- Generated tests are saved in the output folder and can be fed back in to generate new tests
- Uses LLVM instrumentation to trace out new program executions

How Do We Recognize Good C++?

- Passes tests
- Passes tests with sanitizers
- Builds without warnings or errors
- Passes static analysis
- Passes fuzzing tests

Copyright Jason Turner

How Do We Become Better C++ Programmers?

We use the tools.

Links

- valgrind.org
- cppcheck.sourceforge.org
- clang.llvm.org/extra/clang-tidy/
- llvm.org/docs/LibFuzzer.html
- gcc.gnu.org/onlinedocs/gcc/Warning-Options.html
- clang.llvm.org/docs/DiagnosticsReference.html