The Power of Compile-Time Resources

An Overview of some tools, libraries, techniques, a real-world use case

… and hopefully some inspiration.

Jason Turner

- Host of C++ Weekly https://www.youtube.com/c/lefticus1
- Author
  - C++ Best Practices
  - OpCode, Copy and Reference, Object Lifetime Puzzlers
  - https://amzn.to/3xWh8Ox
  - https://leanpub.com/u/jason_turner
- Developer
  - https://cppbestpractices.com
  - https://github.com/cpp-best-practices
- Microsoft MVP for C++ 2015-present

Jason Turner

Independent and available for training and code reviews

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

About my Talks

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

The Power of Compile Time Resources

Backstory

- Rich Code For Tiny Computers - power of the optimizer and 0 cost abstractions (2016)
- `constexpr` All The Things - a proof of concept compile-time JSON parser (2017)
- Applied Best Practices - oops, I accidentally made a `constexpr` capable ARM emulator (proving that anything is possible at compile time) (2018)

Getting Practical

- C++Weekly Ep233 - `constexpr` map vs `std::map` (2020)
- Your New Mental Model For `constexpr` - pushing the limits of how much can be known at compile-time (2021)
- C++Weekly Ep313 - The `constexpr` Problem That Took Me 5 Years To Fix! - reducing the pain of making compile-time resources (2022)
- C++Weekly Ep319 - A JSON to C++ Converter - Let’s just convert JSON straight into C++ compile-time resources (2022)

(I now have 28 videos in my personal `constexpr` playlist - https://bit.ly/jasonturner-constexpr-playlist)

What’s Next?

What’s Next?

- Power of Compile Time Resources ⟵ *You are here*

What compile-time ideas can we apply on a large scale in a real world existing project?

What’s Next?

- Power of Compile Time Resources ⟵ *You are here*

What compile-time ideas can we apply on a large scale in a real world existing project?

And what will happen?

“`constexpr` All The Things”

- Compile-time parsing of JSON files
- Compile-time access to parsed JSON data

This has some problems

- It’s slow to compile
- It’s hard to get right
- Embedding is annoying (no `std::embed` yet :( )

Do You have JSON files Known at compile time?

Do You have JSON files Known at compile time?

- Configuration files that do not change at run time?

Do You have JSON files Known at compile time?

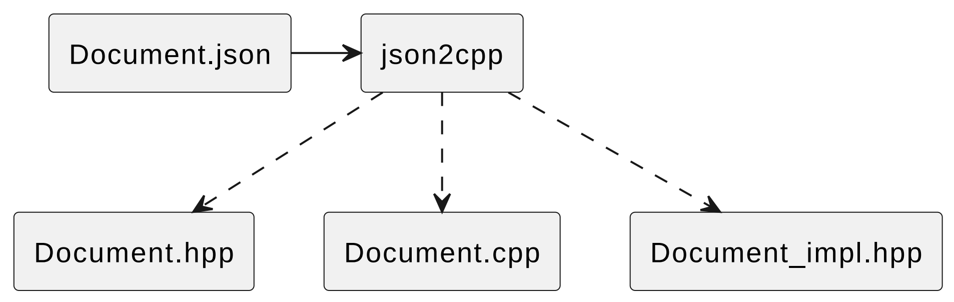
- Configuration files that do not change at run time?
- Schema files that do not change at run time?

json2cpp

json2cpp

- Takes json input file
- Generates C++ files with static constexpr (constant initialized) JSON data
- Generates a compilation firewall (useful for mitigating compile-time impact)
- Produces compile time object models that are **source compatible** with `nlohmann::json`
- Two use cases
  1. avoiding runtime parsing of a file
  2. use file data at compile-time

How The JSON to C++ Converter Works



- `Document.hpp` - header file for runtime access to data
- `Document_impl.hpp` - header file for compile-time access to data
- `Document.cpp` - compilation firewall to avoid all uses having to recompile the data structures

How The JSON to C++ Converter Works

JSON input file is loaded with `nlohmann::json`

```
[10.0, 20.0, 30.0, 40.0]
```

```cpp
// at runtime we can do this:
assert(json[0].get<double>()  == 10.0);
```

How The JSON to C++ Converter Works

The object model that `nlohmann::json` represents is then dumped to C++ (not very complicated, really).

We do a depth-first recursive dump of all of the objects.

Depth first is necessary and will become obvious in a minute.

How The JSON to C++ Converter Works

If the value is a primitive type:

```cpp
std::string compile(const nlohmann::json &value,
                    std::size_t &obj_count,
                    std::vector<std::string> &lines)
{
  // snip
  if (value.is_number_float()) {
    return fmt::format("double{{{}}}", value.get<double>());
  }
  // snip
}
```

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

(Repeat for each type: `int`, `bool`, `std::string`, etc)

How The JSON to C++ Converter Works

If the value is an array:

```cpp
std::string compile(const nlohmann::json &value,
                    std::size_t &obj_count, /// unique id for current object
                    std::vector<std::string> &lines)
{
  // snip
  if (value.is_array()) {
    std::vector<std::string> entries;
    std::transform(value.begin(), value.end(), std::back_inserter(entries),
      [&](const auto &child) {
        // convert objects into list of strings with recursion here
        return fmt::format("{{{}}},", compile(child, obj_count, lines)); ///
      });

    lines.push_back(fmt::format(
      "inline constexpr std::array<json, {}> object_data_{} = {{{{",
      entries.size(), current_object_number));

    std::transform(entries.begin(), entries.end(), std::back_inserter(lines),
      [](const auto &entry) { return fmt::format("  {}", entry); });

    lines.emplace_back("}};");
    return fmt::format("array_t{{object_data_{}}}", current_object_number);
  }
  /// snip
}
```

https://godbolt.org/z/MMP3M8dP1

How The JSON to C++ Converter Works

Repeat above for objects with key/value pairs.

How The Converter Works - `.hpp` output

`test.hpp` - the header file to include if you want the compilation firewall.

```cpp
#ifndef test_COMPILED_JSON
#define test_COMPILED_JSON
#include <json2cpp/json2cpp.hpp>

namespace compiled_json::test {
  const json2cpp::json &get();
}

#endif
```

https://godbolt.org/z/KTzqrTW7f

How The Converter Works - `.cpp` output

`test.cpp` - The C++ file to add to your project if you want the compilation firewall.

```cpp
#include "test_impl.hpp" ///
namespace compiled_json::test {
  const json2cpp::json &get() { return compiled_json::test::impl::document; }
}
```

https://godbolt.org/z/cT8q19o4K

How The Converter Works - `_impl.hpp` output

`test_impl.hpp` - all of the actual data.

```cpp
#ifndef test_COMPILED_JSON_IMPL
#define test_COMPILED_JSON_IMPL
#include <json2cpp/json2cpp.hpp>

namespace compiled_json::test::impl {
using json = json2cpp::basic_json<char>;
using data_t=json2cpp::data_variant<char>;
using string_view=std::basic_string_view<char>;
using array_t=json2cpp::basic_array_t<char>;
using object_t=json2cpp::basic_object_t<char>;
using value_pair_t=json2cpp::basic_value_pair_t<char>;

inline constexpr std::array<json, 4> object_data_0 = {{
  {double{10}},
  {double{20}},
  {double{30}},
  {double{40}},
}};

inline constexpr auto document = json{{array_t{object_data_0}}};
}
#endif
```

https://godbolt.org/z/o83jhse35

Do You Go To Your Local Meetup?

Why `inline constexpr` Instead of `static constexpr`?

Someone at my meetup pointed this out to me:

```cpp
// my hpp file that's included in 15 .cpp files
inline constexpr auto document = json{{array_t{object_data_0}}};
```

vs

```cpp
// my hpp file that's included in 15 .cpp files
static constexpr auto document = json{{array_t{object_data_0}}};
```

Why `inline constexpr` Instead of `static constexpr`?

> *The inlinespecifier can be applied to variables as well as to functions. A variable declared inline has the same semantics as a function declared inline: it can be defined, identically, in multiple translation units, must be defined in every translation unit in which it is odr used, and the behavior of the program is as if there is exactly one variable.*

P0386R2

Why `inline constexpr` Instead of `static constexpr`?

> *The inlinespecifier can be applied to variables as well as to functions. A variable declared inline has the same semantics as a function declared inline: it can be defined, identically, in multiple translation units, must be defined in every translation unit in which it is odr used, and the behavior of the program is as if there is exactly one variable.*

P0386R2

`inline` is necessary to prevent the global objects from being duplicated in each translation unit.

How The JSON to C++ Converter Works

Run time nlohmann::json:

```cpp
// at runtime we can do this:
assert(json[0].get<double>() == 10.0);
```

Compile time json2cpp:

```cpp
// at compile-time we can do this:
constexpr auto &json = compiled_json::test::impl::document;
static_assert(json[0].get<double>() == 10.0);
```

More Complex JSON to C++

```json
{
  "title": "some data",
  "data": [1, 3.0, "Hello World", true, [0, 1]]
}
```

```cpp
inline constexpr std::array<json, 2> object_data_6 = {{
  {std::uint64_t{0}}, {std::uint64_t{1}},
}};
inline constexpr std::array<json, 5> object_data_1 = {{
  {std::uint64_t{1}}, {double{3}},
  {string_view{R"string(Hello World)string"}},
  {bool{true}}, {array_t{object_data_6}},
}};
inline constexpr std::array<value_pair_t, 2> object_data_0 = {
  value_pair_t{R"string(data)string", {array_t{object_data_1}}},
  value_pair_t{R"string(title)string", {string_view{R"string(some data)string"}}},
};

inline constexpr auto document = json{{object_t{object_data_0}}};
//                                                      https://godbolt.org/z/WEajPe341
```

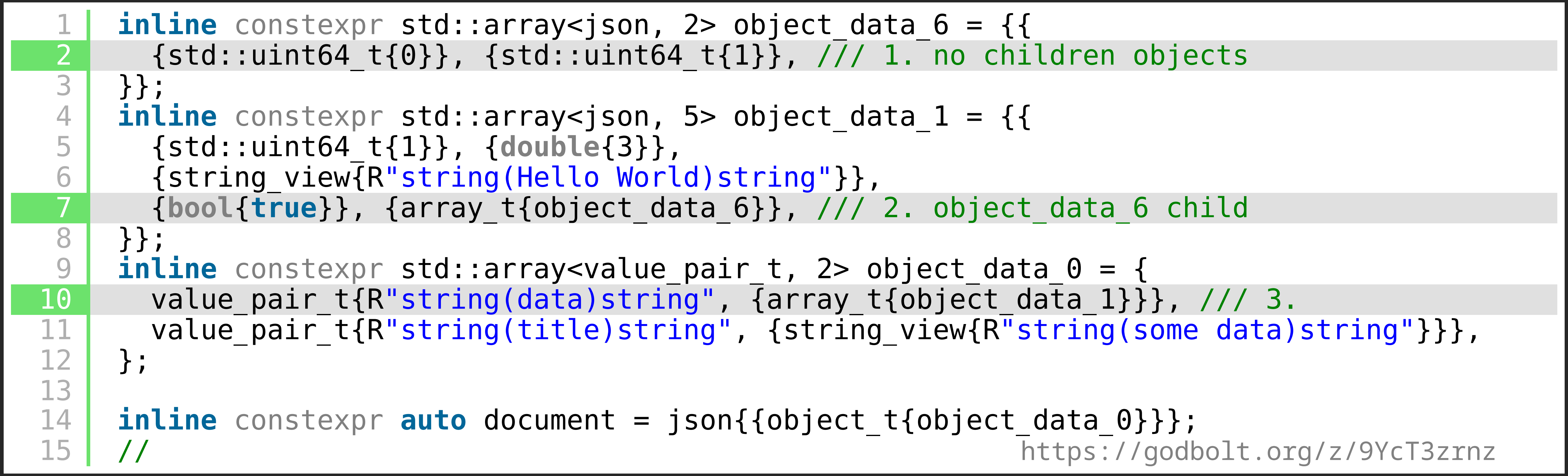
Depth-First Recursion

```
inline constexpr std::array<json, 2> object_data_6 = {{
  {std::uint64_t{0}}, {std::uint64_t{1}}, /// 1. no children objects
}};
inline constexpr std::array<json, 5> object_data_1 = {{
  {std::uint64_t{1}}, {double{3}},
  {string_view{R"string(Hello World)string"}},
  {bool{true}}, {array_t{object_data_6}},
}};
inline constexpr std::array<value_pair_t, 2> object_data_0 = {
  value_pair_t{R"string(data)string", {array_t{object_data_1}}},
  value_pair_t{R"string(title)string", {string_view{R"string(some data)string"}}},
};

inline constexpr auto document = json{{object_t{object_data_0}}};
//                                                      https://godbolt.org/z/Mjecz4qYE
```

Depth-First Recursion

```
inline constexpr std::array<json, 2> object_data_6 = {{
  {std::uint64_t{0}}, {std::uint64_t{1}}, /// 1. no children objects
}};
inline constexpr std::array<json, 5> object_data_1 = {{
  {std::uint64_t{1}}, {double{3}},
  {string_view{R"string(Hello World)string"}},
  {bool{true}}, {array_t{object_data_6}}, /// 2. object_data_6 child
}};
inline constexpr std::array<value_pair_t, 2> object_data_0 = {
  value_pair_t{R"string(data)string", {array_t{object_data_1}}},
  value_pair_t{R"string(title)string", {string_view{R"string(some data)string"}}},
};

inline constexpr auto document = json{{object_t{object_data_0}}};
//                                                      https://godbolt.org/z/MW5G85Y7z
```

Depth-First Recursion



```cpp
inline constexpr std::array<json, 2> object_data_6 = {{
  {std::uint64_t{0}}, {std::uint64_t{1}}, /// 1. no children objects
}};
inline constexpr std::array<json, 5> object_data_1 = {{
  {std::uint64_t{1}}, {double{3}},
  {string_view{R"string(Hello World)string"}},
  {bool{true}}, {array_t{object_data_6}}, /// 2. object_data_6 child
}};
inline constexpr std::array<value_pair_t, 2> object_data_0 = {
  value_pair_t{R"string(data)string", {array_t{object_data_1}}}, /// 3.
  value_pair_t{R"string(title)string", {string_view{R"string(some data)string"}}},
};

inline constexpr auto document = json{{object_t{object_data_0}}};
//                                                      https://godbolt.org/z/9YcT3zrnz
```

By following in depth-first order I never have to worry about sorting the output, since there are no cycles or links in JSON.

But…

Depth-First Recursion

```cpp
inline constexpr std::array<json, 2> object_data_6 = {{
  {std::uint64_t{0}}, {std::uint64_t{1}}, /// 1. no children objects
}};
inline constexpr std::array<json, 5> object_data_1 = {{
  {std::uint64_t{1}}, {double{3}},
  {string_view{R"string(Hello World)string"}},
  {bool{true}}, {array_t{object_data_6}}, /// 2. object_data_6 child
}};
inline constexpr std::array<value_pair_t, 2> object_data_0 = {
  value_pair_t{R"string(data)string", {array_t{object_data_1}}}, /// 3.
  value_pair_t{R"string(title)string", {string_view{R"string(some data)string"}}},
};

inline constexpr auto document = json{{object_t{object_data_0}}};
//                                                        https://godbolt.org/z/9YcT3zrnz
```

By following in depth-first order I never have to worry about sorting the output, since there are no cycles or links in JSON.

But…

Does object element order matter?

“Doing Work At Compile-Time” vs “Not Doing Work At Run-Time”

Doing Work at Compile-Time

What are some things YOU do at compile time?

Doing Work at Compile-Time

What are some things YOU do at compile time?

They all end up looking like this:

```cpp
static constexpr auto value = some_function();
```

Not Doing Work At Run-Time

This talk is not about “`constexpr` All The Things!”

Not Doing Work At Run-Time

This talk is not about “`constexpr` All The Things!”

It is about the end result of moving more data to compile-time constants

Time for an actual use case?

JSON Schema File

Current Status

JSON Schema File

Current Status

- 9,930,795 bytes of JSON schema

JSON Schema File

Current Status

- 9,930,795 bytes of JSON schema
- 155,990 “objects” (bools, objects, arrays, floats, ints…)

JSON Schema File

Current Status

- 9,930,795 bytes of JSON schema
- 155,990 “objects” (bools, objects, arrays, floats, ints…)
- Always known at compile time, always static

JSON Schema File

Current Status

- 9,930,795 bytes of JSON schema
- 155,990 “objects” (bools, objects, arrays, floats, ints…)
- Always known at compile time, always static
- Used exactly once at runtime - to validate parsing of exactly one JSON input file

JSON Schema File

Current Status

- 9,930,795 bytes of JSON schema
- 155,990 “objects” (bools, objects, arrays, floats, ints…)
- Always known at compile time, always static
- Used exactly once at runtime - to validate parsing of exactly one JSON input file
- Currently embedded in application as a CBOR (Concise Binary Object Representation) blob of data (which still must be parsed and the JSON object tree reconstructed at runtime)

JSON Schema File

Possible Future

JSON Schema File

Possible Future

- `static constexpr` objects representing the schema embedded in the binary

JSON Schema File

Possible Future

- `static constexpr` objects representing the schema embedded in the binary
- 0 runtime work

JSON Schema File

Possible Future

- `static constexpr` objects representing the schema embedded in the binary
- 0 runtime work
- 0 runtime allocations

JSON Schema File

Possible Future

- `static constexpr` objects representing the schema embedded in the binary
- 0 runtime work
- 0 runtime allocations
- Available on demand for the valijson validator with an appropriate adapter in place

JSON Schema File Results

Pre:

- 54,427,336 bytes binary
- 218,324 kbytes RAM
- 3,060,715 calls to `new`
- 10.18 s

Post:

- 65,445,608 bytes binary
- 204,896 kbytes RAM
- 2,176,467 calls to `new`
- 10.08 s

JSON Schema File

Results (release builds)

JSON Schema File

Results (release builds)

- binary size increased by ~11 MB - why? (remember we removed the existing CBOR data)

JSON Schema File

Results (release builds)

- binary size increased by ~11 MB - why? (remember we removed the existing CBOR data)
  - How many bytes does 4.2 as a double take up?

JSON Schema File

Results (release builds)

- binary size increased by ~11 MB - why? (remember we removed the existing CBOR data)
  - How many bytes does 4.2 as a double take up?
  - How many bytes does the string “4.2” take up? (CBOR is a compact representation)

JSON Schema File

Results (release builds)

- binary size increased by ~11 MB - why? (remember we removed the existing CBOR data)
  - How many bytes does 4.2 as a double take up?
  - How many bytes does the string “4.2” take up? (CBOR is a compact representation)
- RAM decreased by ~14 MB - why?

JSON Schema File

Results (release builds)

- binary size increased by ~11 MB - why? (remember we removed the existing CBOR data)
  - How many bytes does 4.2 as a double take up?
  - How many bytes does the string “4.2” take up? (CBOR is a compact representation)
- RAM decreased by ~14 MB - why?
  - No objects are created at run time, only accessed

JSON Schema File

JSON Schema File

- calls to `new` decreased by ~884,000 - why?

JSON Schema File

- calls to `new` decreased by ~884,000 - why?
  - No `std::map`, `std::vector`, or `std::string` to hold the data

JSON Schema File

- calls to `new` decreased by ~884,000 - why?
  - No `std::map`, `std::vector`, or `std::string` to hold the data
  - No *resizing* of string/map/vector data while loading!

JSON Schema File

- calls to `new` decreased by ~884,000 - why?
  - No `std::map`, `std::vector`, or `std::string` to hold the data
  - No *resizing* of string/map/vector data while loading!
- Runtime decreased by a consistent 100ms - why?

JSON Schema File

- calls to `new` decreased by ~884,000 - why?
  - No `std::map`, `std::vector`, or `std::string` to hold the data
  - No *resizing* of string/map/vector data while loading!
- Runtime decreased by a consistent 100ms - why?
  - 0 parsing of schema at runtime

JSON Schema File

- calls to `new` decreased by ~884,000 - why?
  - No `std::map`, `std::vector`, or `std::string` to hold the data
  - No *resizing* of string/map/vector data while loading!
- Runtime decreased by a consistent 100ms - why?
  - 0 parsing of schema at runtime

Note: *The 100ms decrease is a constant. No matter how the tool is used, it's always exactly 100ms faster. If you are running millions of tasks, that can really add up!*

A Quick Look at Compile-Time Exceptions

Compile-Time Exceptions

```
#include <cstdint>

constexpr std::uint8_t set_bit(std::uint8_t input, std::uint8_t bit) {
  return (input | static_cast<std::uint8_t>(1 << bit));
}

consteval std::uint8_t validate(std::uint8_t input) {
  if ((input & 2) != 0 && (input & 8) != 0) {
      throw "You crossed the streams!";
  }
  return input;
}

constexpr std::uint8_t build_value() {
  return validate(set_bit(set_bit(0, 1), 3));
}

int main() { constexpr auto value = build_value(); }  https://godbolt.org/z/sYP7aa65x
```

Note: `throw` will never make it into your binary in this use case.

JSON For Compile-Time Configuration

JSON For Compile-Time Configuration

Scenario (partially fabricated)

JSON For Compile-Time Configuration

Scenario (partially fabricated)

- Embedded device

JSON For Compile-Time Configuration

Scenario (partially fabricated)

- Embedded device
- Hardware configuration options fixed when device is created

JSON For Compile-Time Configuration

Scenario (partially fabricated)

- Embedded device
- Hardware configuration options fixed when device is created
- Storage and run time at a premium

JSON For Compile-Time Configuration

Scenario (partially fabricated)

- Embedded device
- Hardware configuration options fixed when device is created
- Storage and run time at a premium
- Want to configure bits to be set at runtime for an embedded device “port”

JSON For Compile-Time Configuration

Scenario (partially fabricated)

- Embedded device
- Hardware configuration options fixed when device is created
- Storage and run time at a premium
- Want to configure bits to be set at runtime for an embedded device "port"
- You want to set up the configuration at compile-time

JSON For Compile-Time Configuration

Scenario (partially fabricated)

- Embedded device
- Hardware configuration options fixed when device is created
- Storage and run time at a premium
- Want to configure bits to be set at runtime for an embedded device "port"
- You want to set up the configuration at compile-time
- Inspired partially by discussions with Odin Holmes in ~2016 about Kvasir

JSON For Compile-Time Configuration

Scenario (partially fabricated)

- Embedded device
- Hardware configuration options fixed when device is created
- Storage and run time at a premium
- Want to configure bits to be set at runtime for an embedded device “port”
- You want to set up the configuration at compile-time
- Inspired partially by discussions with Odin Holmes in ~2016 about Kvasir
- Also inspired by my “Applied `constexpr`” class I teach

JSON For Compile-Time Configuration

Scenario (partially fabricated)

- Embedded device
- Hardware configuration options fixed when device is created
- Storage and run time at a premium
- Want to configure bits to be set at runtime for an embedded device “port”
- You want to set up the configuration at compile-time
- Inspired partially by discussions with Odin Holmes in ~2016 about Kvasir
- Also inspired by my “Applied `constexpr`” class I teach
- This is pseudo code

JSON For Compile-Time Configuration

```cpp
int main() {
  // no work at runtime, these are just a few bytes
  static constexpr auto PORTA =
      configure_port(compiled_json::configuration::document(), "A");
  static constexpr auto PORTB =
      configure_port(compiled_json::configuration::document(), "B");

  // PORTA is something like 0x4567
  // PORTB is something like 0x1234

  // some writes at compile-time
  write_config_register(registers::PORTA, PORTA);
  write_config_register(registers::PORTB, PORTB);
}
```

https://godbolt.org/z/5noMaaPK1

JSON For Compile-Time Configuration

```cpp
constexpr port configure_port(const compiled_json::json &config_file,
                              const std::string_view port_name) {
  port result;
  // compile-time failure for any misconfigured input file
  const auto &port_data = config_file["ports"][port_name];

  for (const auto &pin : port_data.pins) {
    const auto input = pin["direction"].get<std::string_view>() == "in";
    const auto pullup = pin["pullup"].get<bool>() == true;
    // validate assumptions at compile-time
    if (!input && pullup) { throw "pullup && output pin is illogical"; }

    set_bit(result.direction, pin["pin_number"].get<int>(), input);
    set_bit(result.pullup, pin["pin_number"].get<int>(), pullup);
  }

  return result;
}
```

https://godbolt.org/z/o4bx6EYhG

JSON For Compile-Time Configuration

JSON For Compile-Time Configuration

- Zero cost at runtime

JSON For Compile-Time Configuration

- Zero cost at runtime
- Compile-time validation of configuration is easy to do

JSON For Compile-Time Configuration

- Zero cost at runtime
- Compile-time validation of configuration is easy to do
- Probably lower cost than whatever you are currently doing

I know what you’re thinking.

“But my configuration isn’t in a JSON file!”

OK, that’s fair, why would it be?

Making Compile-Time Configurations

```cpp
#include <map>
#include <string>

constexpr auto make_config()
{

  std::map<std::string, std::map<std::string, bool>> config_data;
  config_data["port"]["input"] = true;
  config_data["port"]["pullup"] = false;
  return config_data;
}

int main()
{
  static constexpr auto config = make_config();
}
```

https://godbolt.org/z/zPxfdhhc1

Making Compile-Time Configurations

```cpp
#include <map>
#include <string>

constexpr auto make_config()
{

  std::map<std::string, std::map<std::string, bool>> config_data;
  config_data["port"]["input"] = true;
  config_data["port"]["pullup"] = false;
  return config_data;
}

int main()
{
  static constexpr auto config = make_config();
}
```

https://godbolt.org/z/zPxfdhhc1

Easy, right?

Making Compile-Time Configurations

```cpp
#include <map>
#include <string>

constexpr auto make_config()
{

  std::map<std::string, std::map<std::string, bool>> config_data;
  config_data["port"]["input"] = true;
  config_data["port"]["pullup"] = false;
  return config_data;
}

int main()
{
  static constexpr auto config = make_config();
}
```

https://godbolt.org/z/zPxfdhhc1

Easy, right?

WRONG!

Making Compile-Time Configurations

```cpp
#include <map>
#include <string>

constexpr auto make_config()
{

  std::map<std::string, std::map<std::string, bool>> config_data;
  config_data["port"]["input"] = true;
  config_data["port"]["pullup"] = false;
  return config_data;
}

int main()
{
  static constexpr auto config = make_config();
}
```

https://godbolt.org/z/zPxfdhhc1

How wrong is this?

Making Compile-Time Configurations

```cpp
#include <map>
#include <string>

constexpr auto make_config()
{
  /// map not constexpr
  std::map<std::string, std::map<std::string, bool>> config_data; ///
  config_data["port"]["input"] = true;
  config_data["port"]["pullup"] = false;
  return config_data;
}

int main()
{
  static constexpr auto config = make_config(); /// cannot escape compile time
}
```

https://godbolt.org/z/bjnrhMjjj

An aside on `constexpr` `std::string`

An aside on `constexpr` `std::string`

```cpp
#include <string>

constexpr std::string get_string() {
  return "My long string data";
}

int main() {
  constexpr auto value = get_string();

}
```

https://godbolt.org/z/9ofn469xP

Is this OK?

An aside on `constexpr` `std::string`

```cpp
#include <string>

constexpr std::string get_string() {
  return "My long string data";
}

int main() {
  constexpr auto value = get_string();
  // a `constexpr std::string` cannot exist
}
```

https://godbolt.org/z/84n53MoTT

Compile error!

An aside on `constexpr` `std::string`

```cpp
#include <string>

constexpr std::string get_string() {
  return "My long string data";
}

constexpr std::size_t length() {
  auto data = get_string();
  data += " hello world";
  return data.size();
}

int main() {
  constexpr auto value = length();
}
```

https://godbolt.org/z/oxadhWd9G

Is this OK?

An aside on `constexpr` `std::string`

```cpp
#include <string>

constexpr std::string get_string() {
  return "My long string data";
}

constexpr std::size_t length() {
  auto data = get_string();
  data += " hello world";
  return data.size();
}

int main() {
  constexpr auto value = length();
}
```

https://godbolt.org/z/oxadhWd9G

Is this OK?

Yes! We can use `std::string` at compile time, but we cannot access a compile-time string at run-time.

What Do We Need?

- A `constexpr` compatible map type
- A way to move compile-time allocated data to runtime

constexpr Helpers - vector

```cpp
#include <array>

template<typename Value, std::size_t MaxSize>
class vector {
private:
  std::array<Value, MaxSize> data;
  std::size_t current_size = 0;

public:
  Value &operator[](const std::size_t idx);
  const Value &operator[](const std::size_t idx) const;
  std::size_t size() { return current_size; }
  void push_back(const Value &value) { data[current_size++] = value; }
  void push_back(Value &&value) { data[current_size++] = std::move(value); }
};
```

https://godbolt.org/z/359KhMs3c

constexpr Helpers - vector

```cpp
#include <array>

template<typename Value, std::size_t MaxSize>
class vector {
private:
  std::array<Value, MaxSize> data;
  std::size_t current_size = 0;

public:
  Value &operator[](const std::size_t idx);
  const Value &operator[](const std::size_t idx) const;
  std::size_t size() { return current_size; }
  void push_back(const Value &value) { data[current_size++] = value; }
  void push_back(Value &&value) { data[current_size++] = std::move(value); }
};
```

https://godbolt.org/z/359KhMs3c

Erm… something missing?

constexpr Helpers - vector

```cpp
#include <array>

template<typename Value, std::size_t MaxSize>
class vector {
private:
  std::array<Value, MaxSize> data;
  std::size_t current_size = 0;

public:
  constexpr Value &operator[](const std::size_t idx);
  constexpr const Value &operator[](const std::size_t idx) const;
  constexpr std::size_t size() { return current_size; }
  constexpr void push_back(const Value &value) { data[current_size++] = value; }
  constexpr void push_back(Value &&value) { data[current_size++] = std::move(value);
}
};
```

https://godbolt.org/z/nz6hsW4Ke

constexpr Helpers - vector

```cpp
#include <array>

template<typename Value, std::size_t MaxSize>
class vector {
private:
  std::array<Value, MaxSize> data;
  std::size_t current_size = 0;

public:
  constexpr Value &operator[](const std::size_t idx);
  constexpr const Value &operator[](const std::size_t idx) const;
  constexpr std::size_t size() { return current_size; }
  constexpr void push_back(const Value &value) { data[current_size++] = value; }
  constexpr void push_back(Value &&value) { data[current_size++] = std::move(value);
}
};
```

https://godbolt.org/z/nz6hsW4Ke

Anything else?

constexpr Helpers - vector

```cpp
#include <array>

template<typename Value, std::size_t MaxSize>
class vector {
private:
  std::array<Value, MaxSize> data;
  std::size_t current_size = 0;

public:
  [[nodiscard]] constexpr Value &operator[](const std::size_t idx);
  [[nodiscard]] constexpr const Value &operator[](const std::size_t idx) const;
  [[nodiscard]] constexpr std::size_t size() { return current_size; }
  constexpr void push_back(const Value &value) { data[current_size++] = value; }
  constexpr void push_back(Value &&value) { data[current_size++] = std::move(value);
}
};
```

https://godbolt.org/z/ja5vhfqe6

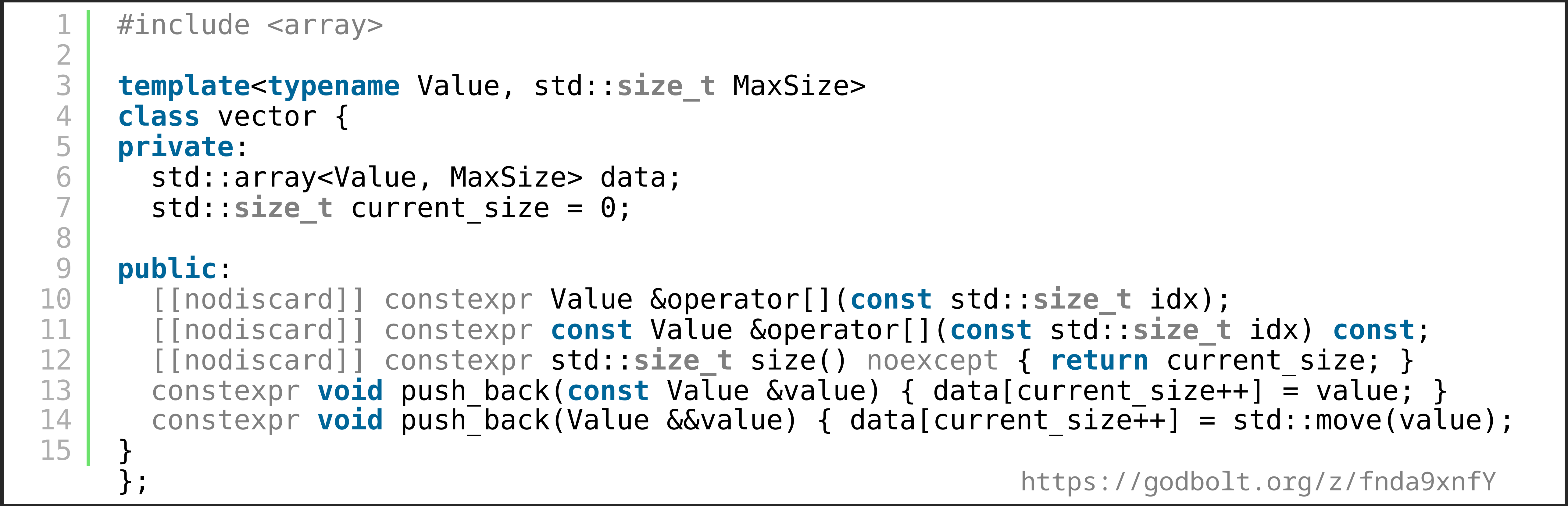
constexpr Helpers - vector

```
#include <array>

template<typename Value, std::size_t MaxSize>
class vector {
private:
  std::array<Value, MaxSize> data;
  std::size_t current_size = 0;

public:
  [[nodiscard]] constexpr Value &operator[](const std::size_t idx);
  [[nodiscard]] constexpr const Value &operator[](const std::size_t idx) const;
  [[nodiscard]] constexpr std::size_t size() { return current_size; }
  constexpr void push_back(const Value &value) { data[current_size++] = value; }
  constexpr void push_back(Value &&value) { data[current_size++] = std::move(value);
}
};
```

https://godbolt.org/z/ja5vhfqe6

Anything else?

constexpr Helpers - vector



```cpp
#include <array>

template<typename Value, std::size_t MaxSize>
class vector {
private:
  std::array<Value, MaxSize> data;
  std::size_t current_size = 0;

public:
  [[nodiscard]] constexpr Value &operator[](const std::size_t idx);
  [[nodiscard]] constexpr const Value &operator[](const std::size_t idx) const;
  [[nodiscard]] constexpr std::size_t size() noexcept { return current_size; }
  constexpr void push_back(const Value &value) { data[current_size++] = value; }
  constexpr void push_back(Value &&value) { data[current_size++] = std::move(value);
}
};
```

https://godbolt.org/z/fnda9xnfY

constexpr Helpers - vector

```cpp
#include <array>

template<typename Value, std::size_t MaxSize>
class vector {
private:
  std::array<Value, MaxSize> data;
  std::size_t current_size = 0;

public:
  [[nodiscard]] constexpr Value &operator[](const std::size_t idx);
  [[nodiscard]] constexpr const Value &operator[](const std::size_t idx) const;
  [[nodiscard]] constexpr std::size_t size() noexcept { return current_size; }
  constexpr void push_back(const Value &value) { data[current_size++] = value; }
  constexpr void push_back(Value &&value) { data[current_size++] = std::move(value);
}
};
```

https://godbolt.org/z/fnda9xnfY

Anything else?

OK, we’ll try to keep it simpler for the slides

constexpr Helpers - flat_map

```cpp
template<typename Key, typename Value, std::size_t MaxSize>
class flat_map {
private:
  vector<std::pair<Key, Value>, MaxSize> data;

public:
  constexpr Value &operator[](const Key &);
  constexpr const Value &operator[](const Key &) const;
  constexpr std::size_t size() { return vector.size(); }
};
```

https://godbolt.org/z/a1vET858s

constexpr Helpers - flat_map

```
template<typename Key, typename Value, std::size_t MaxSize>
class flat_map {
private:
  vector<std::pair<Key, Value>, MaxSize> data;

public:
  constexpr Value &operator[](const Key &);
  constexpr const Value &operator[](const Key &) const;
  constexpr std::size_t size() { return vector.size(); }
};
```

https://godbolt.org/z/a1vET858s

(Implementation left as an exercise for the reader.)

`constexpr` Helpers - string

```cpp
template<std::size_t MaxSize>
class string {
private:
  std::array<char, MaxSize> data;
  std::size_t current_size = 0;

public:
  template<std::size_t Size>
  constexpr string(const char (&str)[Size])
    : current_size{Size-1}
  {
    std::copy(begin(str), end(str), begin(data));
  }
};
```

https://godbolt.org/z/h6xEqWa65

Making Compile-Time Configurations - Take 2

```cpp
constexpr auto make_config()
{
  flat_map<string<10>, flat_map<string<10>, bool, 10>, 10> config_data;
  config_data["port"]["input"] = true;
  config_data["port"]["pullup"] = false;
  return config_data;
}

int main()
{
  static constexpr auto config = make_config();
}
```

https://godbolt.org/z/cn8WKEE4G

This works, but is not ideal, because…

Making Compile-Time Configurations - Take 2

```cpp
constexpr auto make_config()
{
  flat_map<string<10>, flat_map<string<10>, bool, 10>, 10> config_data;
  config_data["port"]["input"] = true;
  config_data["port"]["pullup"] = false;
  return config_data;
}

int main()
{
  static constexpr auto config = make_config();
}
```

https://godbolt.org/z/cn8WKEE4G

This works, but is not ideal, because…

we have to over provision things.

Let’s NOT dig into the details for how to minimize this…

But there are techniques.
(See Ep313)

```cpp
// from https://github.com/lefticus/tools
#include </home/jason/tools/include/lefticus/tools/flat_map.hpp>
#include </home/jason/tools/include/lefticus/tools/static_views.hpp>
std::string_view get_first();
std::string_view get_second();

int main() {
  auto make_config = [] {
    lefticus::tools::flat_map<std::string,
                              lefticus::tools::flat_map<std::string, bool>>
                config_data;
    config_data["port"]["input"] = true;
    config_data["port"]["pullup"] = false;
    return config_data;
  };

  // '10' indicates the compile-time temporary storage size allowed
  static constexpr auto config =
    lefticus::tools::minimized_stackify<10>(make_config);

  static_assert(config.size() == 1);
  static_assert(config.begin()->second.size() == 2);

  return config.at(get_first()).at(get_second());
}
```

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

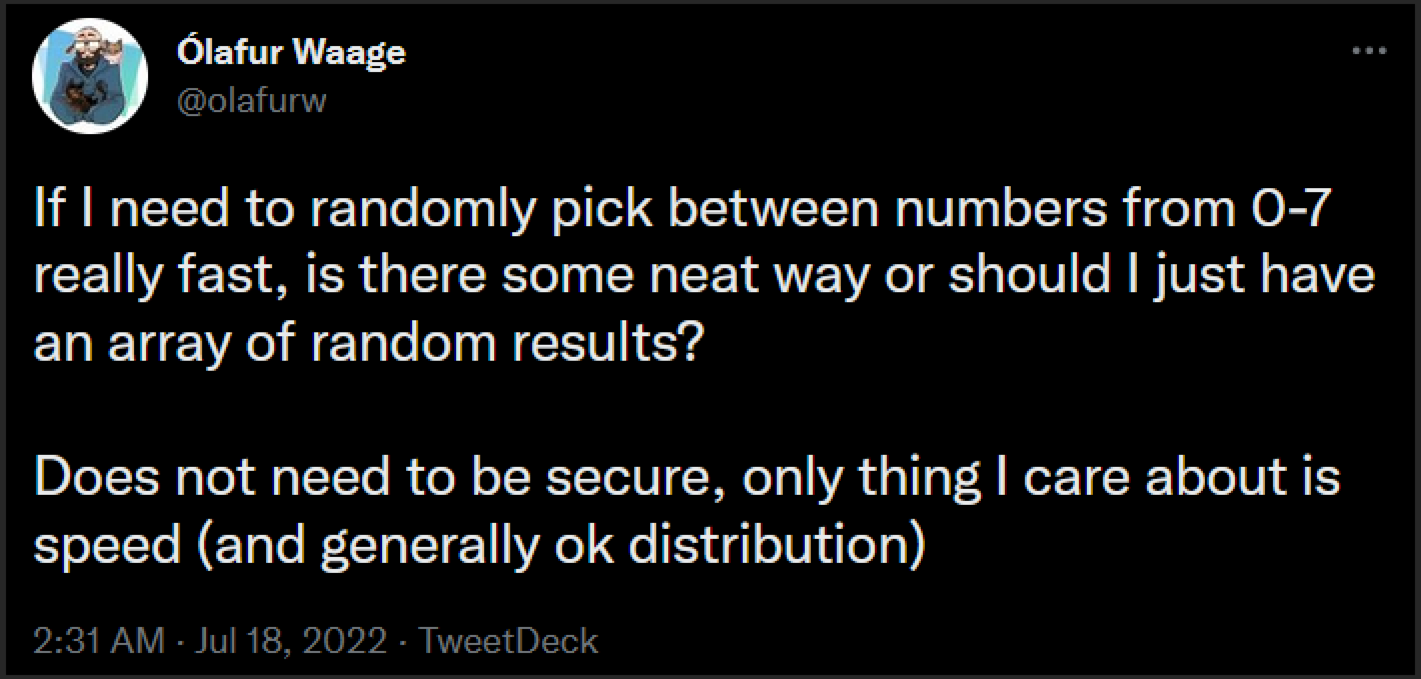
This will create a `map<string<4>, map<string<6>, bool, 2>, 1>`

Compared to `std::map`

```cpp
#include <string>
#include <map>

std::string get_first();
std::string get_second();

int main() {
  auto make_config = [] {
    std::map<std::string, std::map<std::string, bool>> config_data;
    config_data["port"]["input"] = true;
    config_data["port"]["pullup"] = false;
    return config_data;
  };


  static const auto config = make_config();

  return config.at(get_first()).at(get_second());
}
```

https://godbolt.org/z/c7b4xGPKr

Not Doing Work at Runtime



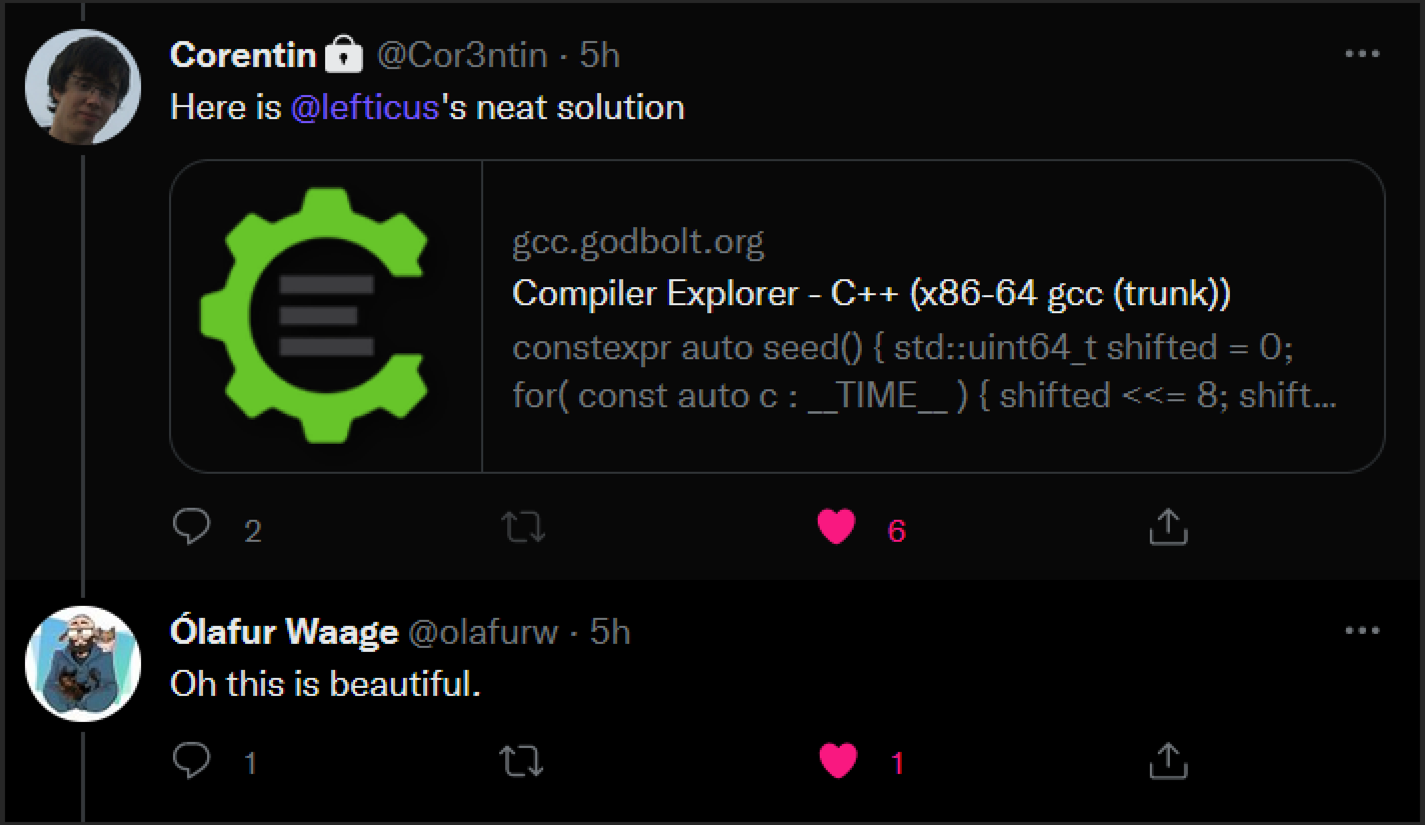
Ólafur Waage
@olafurw

If I need to randomly pick between numbers from 0-7 really fast, is there some neat way or should I just have an array of random results?

Does not need to be secure, only thing I care about is speed (and generally ok distribution)

2:31 AM · Jul 18, 2022 · TweetDeck

Not Doing Work at Runtime



Corentin @Cor3ntin · 5h

Here is @lefticus's neat solution

gcc.godbolt.org

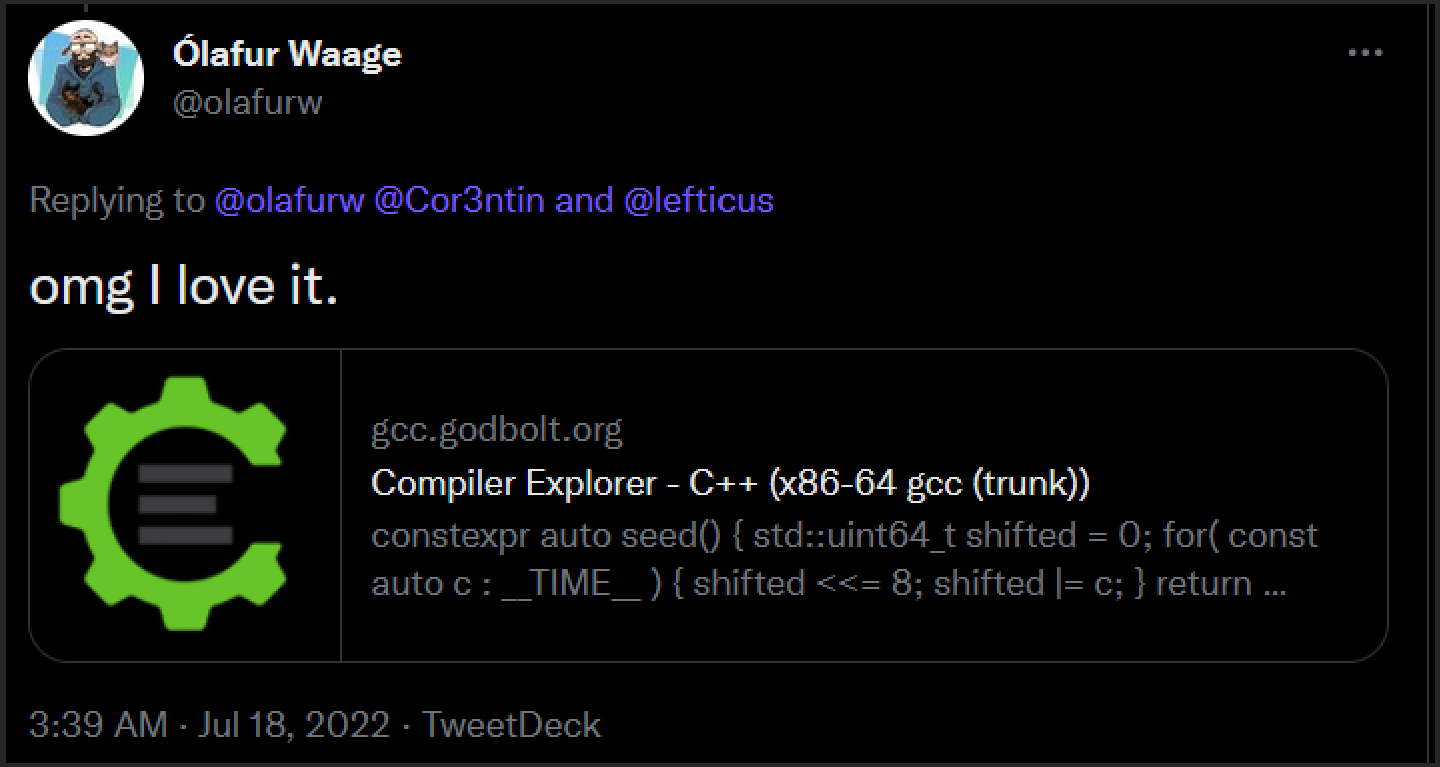
Compiler Explorer - C++ (x86-64 gcc (trunk))

constexpr auto seed() { std::uint64_t shifted = 0; for(const auto c : __TIME__) { shifted <<= 8; shift...

Ólafur Waage @olafurw · 5h

Oh this is beautiful.

Not Doing Work at Runtime



Ólafur Waage
@olafurw

Replying to @olafurw @Cor3ntin and @lefticus

omg I love it.

gcc.godbolt.org
Compiler Explorer - C++ (x86-64 gcc (trunk))
constexpr auto seed() { std::uint64_t shifted = 0; for(const auto c : __TIME__) { shifted <<= 8; shifted |= c; } return ...

3:39 AM · Jul 18, 2022 · TweetDeck

Not Doing Work at Runtime

```cpp
int main()
{
  constexpr std::array<uint8_t, 4> numbers{
      static_cast<uint8_t>(get_random(10) % 256),
      static_cast<uint8_t>(get_random(10) % 256),
      static_cast<uint8_t>(get_random(10) % 256),
      static_cast<uint8_t>(get_random(10) % 256)
  };
  return numbers[0];
}
```

https://godbolt.org/z/qEP6qeqvb

Beyond JSON

Beyond JSON

Beyond JSON

- If you don’t like JSON, but do use configuration files, what format?

Beyond JSON

- If you don’t like JSON, but do use configuration files, what format?
- If you just don’t like JSON and want to pay me to write a yaml2cpp translator, let me know!

Jason Turner

- Host of C++ Weekly https://www.youtube.com/c/lefticus1
- Author
  - C++ Best Practices
  - OpCode, Copy and Reference, Object Lifetime Puzzlers
  - https://amzn.to/3xWh8Ox
  - https://leanpub.com/u/jason_turner
- Developer
  - https://cppbestpractices.com
  - https://github.com/cpp-best-practices
- Microsoft MVP for C++ 2015-present

Jason Turner

Independent and available for training and code reviews

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