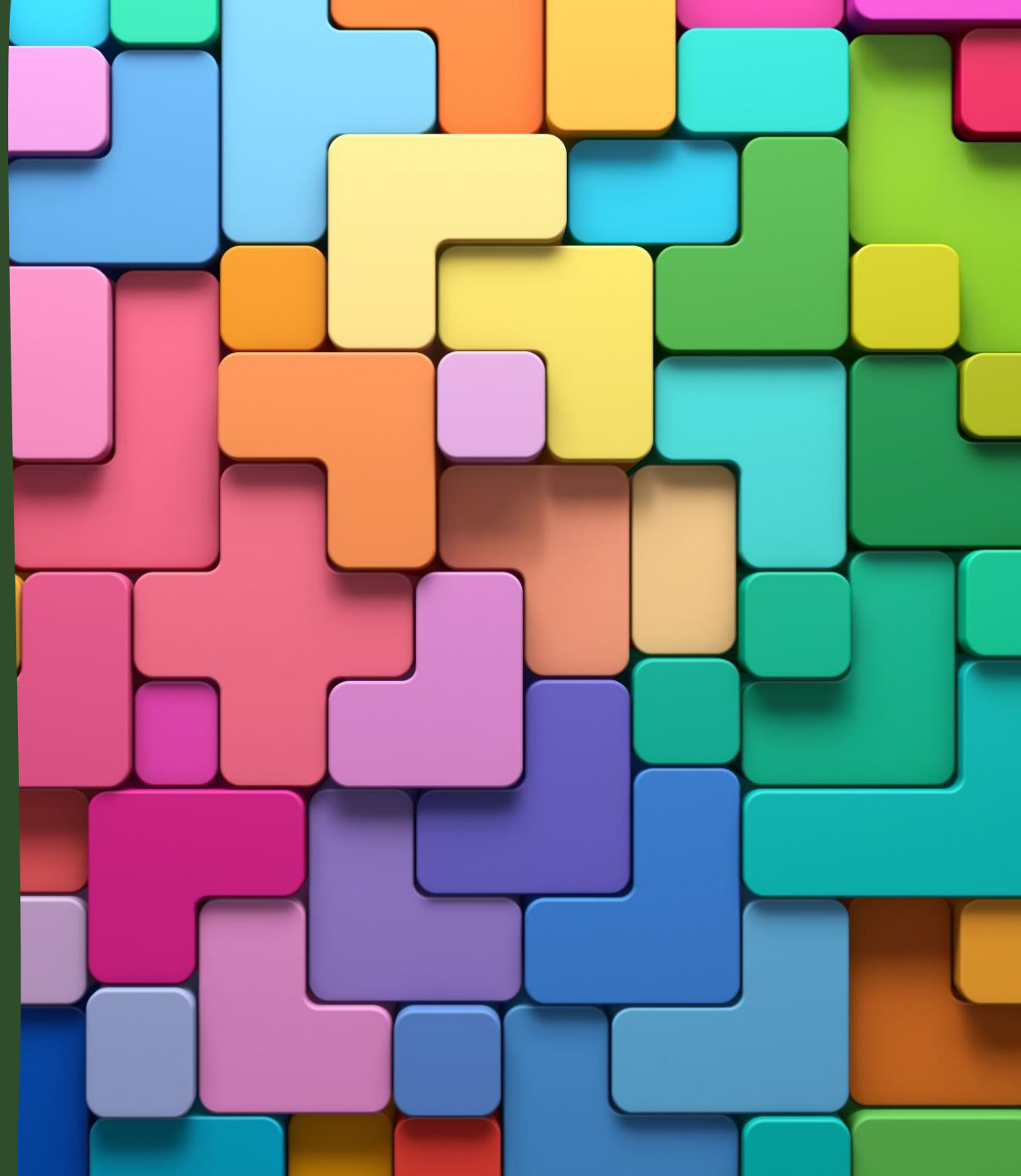
IS THAT C++?

Gareth Lloyd (ACCU York + Memgraph)

@glloyd@fosstodon.org



```cpp
#include <chrono>
#include <print>

int main()
{
    using namespace std::chrono_literals;
    using namespace std::chrono;

    constexpr auto first_month = 2025y / February;
    constexpr auto meeting_time = 18h + 30min;

    constexpr auto conf_start = 2025y / March / 31d;
    constexpr auto conf_end = 2025y / April / 4d;

    for (auto ym = first_month; ym.year() == 2025y; ym += months{1})
    {
        auto const day = year_month_day{ym / Wednesday[1]};
        if (conf_start <= day && day <= conf_end)
        {
            std::println(
                "🌟 ACCU Conf 🗓️ {:%B %d} - {:%B %d, %Y}", conf_start, conf_end);
        }
        else
        {
            auto const meetup = zoned_time{"Europe/London", local_days{day} + meeting_time};
            std::println("🗓️ {0:%B %d} | 🕕 {0:%R %Z}", meetup);
        }
    }
}
```

```cpp
#include <chrono>
#include <print>

int main()
{
    using namespace std::chrono_literals;
    using namespace std::chrono;

    constexpr auto first_month = 2025y / February;
    constexpr auto meeting_time = 18h + 30min;

    constexpr auto conf_start = 2025y / March / 31d;
    constexpr auto conf_end = 2025y / April / 4d;

    for (auto ym = first_month; ym.year() == 2025y; ym += months{1})
    {
        auto const day = year_month_day{ym / Wednesday[1]};
        if (conf_start <= day && day <= conf_end)
        {
            std::println(
                "🌟 ACCU Conf 🗓️ {:%B %d} - {:%B %d, %Y}", conf_start, conf_end);
        }
        else
        {
            auto const meetup = zoned_time{"Europe/London", local_days{day} + meeting_time};
            std::println("🗓️ {0:%B %d} | 🕕 {0:%R %Z}", meetup);
        }
    }
}
```

User-defined literals
(since C++11)

```cpp
#include <chrono>
#include <print>

int main()
{
    using namespace std::chrono_literals;
    using namespace std::chrono;

    constexpr auto first_month = 2025y / February;
    constexpr auto meeting_time = 18h + 30min;

    constexpr auto conf_start = 2025y / March / 31d;
    constexpr auto conf_end = 2025y / April / 4d;

    for (auto ym = first_month; ym.year() == 2025y; ym += months{1})
    {
        auto const day = year_month_day{ym / Wednesday[1]};
        if (conf_start <= day && day <= conf_end)
        {
            std::println(
                "🌟 ACCU Conf 🗓️ {:%B %d} - {:%B %d, %Y}", conf_start, conf_end);
        }
        else
        {
            auto const meetup = zoned_time{"Europe/London", local_days{day} + meeting_time};
            std::println("🗓️ {0:%B %d} | 🕕 {0:%R %Z}", meetup);
        }
    }
}
```

chrono (since C++11)

dates (since C++20)

time literals (since C++14)

```cpp
#include <chrono>
#include <print>

int main()
{
    using namespace std::chrono_literals;
    using namespace std::chrono;

    constexpr auto first_month = 2025y / February;
    constexpr auto meeting_time = 18h + 30min;

    constexpr auto conf_start = 2025y / March / 31d;
    constexpr auto conf_end = 2025y / April / 4d;

    for (auto ym = first_month; ym.year() == 2025y; ym += months{1})
    {
        if (conf_start <= day && day <= conf_end)
        {
            std::println(
                "🌟 ACCU Conf 📅 {:%B %d} - {:%B %d, %Y}", conf_start, conf_end);
        }
        else
        {
            auto const meetup = zoned_time{"Europe/London", local_days{day} + meeting_time};
            std::println("📅 {0:%B %d} | 🕐 {0:%R %Z}", meetup);
        }
    }
}
```

Operator overloading

```cpp
constexpr auto operator/(
            const std::chrono::year& y,
            const std::chrono::month& m
) noexcept -> std::chrono::year_month;
```

Placeholder type specifiers (since C++11)
deduced to be **`std::chrono::year_month`**

```
1  #include <vector>
2
3  int main()
4  {
5      std::vector<int> vec;
6      vec.push_back(42);
7
8      for (std::vector<int>::const_iterator it = vec.cbegin(); it != vec.cend(); ++it) {
9          int const & value = *it;
10         // do something
11     }
12 }
```

```
1 #include <vector>
2
3 int main()
4 {
5     std::vector<int> vec;
6     vec.push_back(42);
7
8     for (auto it = vec.cbegin(); it != vec.cend(); ++it) {
9         int const & value = *it;
10        // do something
11    }
12 }
```

```cpp
#include <vector>

int main()
{
    std::vector<int> vec;
    vec.push_back(42);

    for (auto const & value : vec) {
        // do something
    }
}
```

```
1  #include <vector>
2
3  int main()
4  {
5      auto vec = std::vector{42};
6
7      for (auto const & value : vec) {
8          // do something
9      }
10 }
```

std::initilizer_list (since C++11)

Class template argument deduction (CTAD) (since C++17)

```cpp
#include <array>

int main()
{
    auto arr = std::array{42};

    for (auto const & value : arr) {
        // do something
    }
}
```

std::array (since C++11)

```cpp
#include <chrono>
#include <print>

int main()
{
    using namespace std::chrono_literals;
    using namespace std::chrono;

    constexpr auto first_month = 2025y / February;
    constexpr auto meeting_time = 18h + 30min;

    constexpr auto conf_start = 2025y / March / 31d;
    constexpr auto conf_end = 2025y / April / 4d;

    for (auto ym = first_month; ym.year() == 2025y; ym += months{1})
    {
        auto const day = year_month_day{ym / Wednesday[1]};
        if (conf_start <= day && day <= conf_end)
        {
            std::println(
                "🌟 ACCU Conf 🗓️ {:%B %d} - {:%B %d, %Y}", conf_start, conf_end);
        }
        else
        {
            auto const meetup = zoned_time{"Europe/London", local_days{day} + meeting_time};
            std::println("🗓️ {0:%B %d} | 🕕 {0:%R %Z}", meetup);
        }
    }
}
```

constexpr specifier (since C++11)

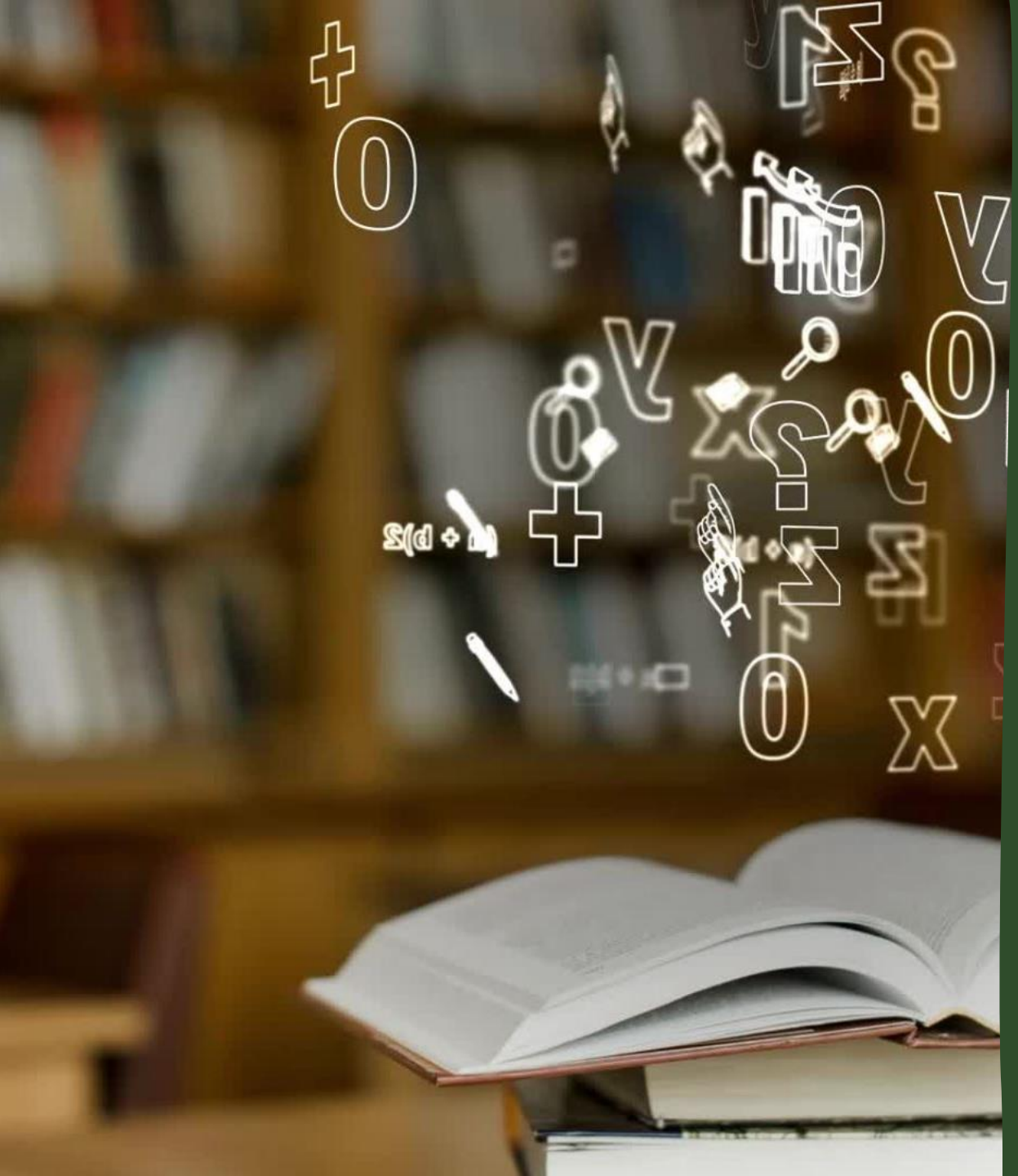
```cpp
template <int N>
struct Fib {
    static int const value = Fib<N-1>::value + Fib<N-2>::value;
};

template <>
struct Fib<1> {
    static int const value = 1;
};

template <>
struct Fib<2> {
    static int const value = 1;
};

int main() {
    return Fib<10>::value; // 55
}
```

```cpp
constexpr auto fib(int n) -> int
{
    if (n == 1 || n == 2)
    {
        return 1;
    }
    return fib(n-1) + fib(n-2);
}

int main() {
    constexpr auto result = fib(10); //55
    return result;
}
```

```cpp
#include <chrono>
#include <print>

int main()
{
    using namespace std::chrono_literals;
    using namespace std::chrono;

    constexpr auto first_month = 2025y / February;
    constexpr auto meeting_time = 18h + 30min;

    constexpr auto conf_start = 2025y / March / 31d;
    constexpr auto conf_end = 2025y / April / 4d;

    for (auto ym = first_month; ym.year() == 2025y; ym += months{1})
    {
        auto const day = year_month_day{ym / Wednesday[1]};
        if (conf_start <= day && day <= conf_end)
        {
            std::println(
                "{:%B %d} - {:%B %d, %Y}", conf_start, conf_end);
        }
        else
        {
            auto const meetup = zoned_time{"Europe/London", local_days{day} + meeting_time};
            std::println("📅 {0:%B %d} | 🕒 {0:%R %Z}", meetup);
        }
    }
}
```

std::format_string (since C++20)

std::print (since C++23)



Takeaway

- C++ has always has a philosophy/approach of performance
  - Zero cost abstractions
- Runtime considered wasteful, move as much as you can to compile-time
- Ergonomics of using C++ has improved
- Never stop learning

ACCU York

1st Wednesday of the month

- 📅 February 05 | 🕐 18:30 GMT
- 📅 March 05 | 🕐 18:30 GMT
- 🌟 ACCU Conf 📅 March 31 - April 04, 2025
- 📅 May 07 | 🕐 18:30 BST
- 📅 June 04 | 🕐 18:30 BST
- 📅 July 02 | 🕐 18:30 BST
- 📅 August 06 | 🕐 18:30 BST
- 📅 September 03 | 🕐 18:30 BST
- 📅 October 01 | 🕐 18:30 BST
- 📅 November 05 | 🕐 18:30 GMT
- 📅 December 03 | 🕐 18:30 GMT

Extra demos

- Variadic + folds + template variables + template lambda
  - https://godbolt.org/z/e5cqbGExW
- Sorting by key in different collection
  - https://godbolt.org/z/5xno9nsdn