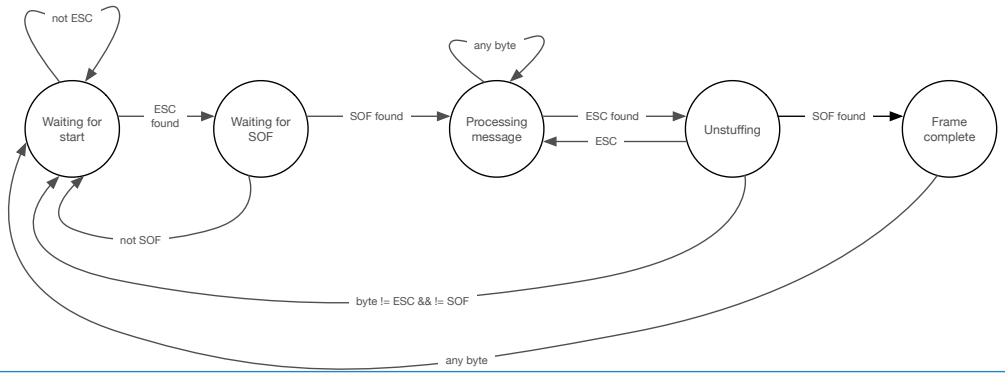
C++20: What's in there for you

Andreas Fertig
https://AndreasFertig.Info
post@AndreasFertig.Info
@Andreas__Fertig

Coroutines

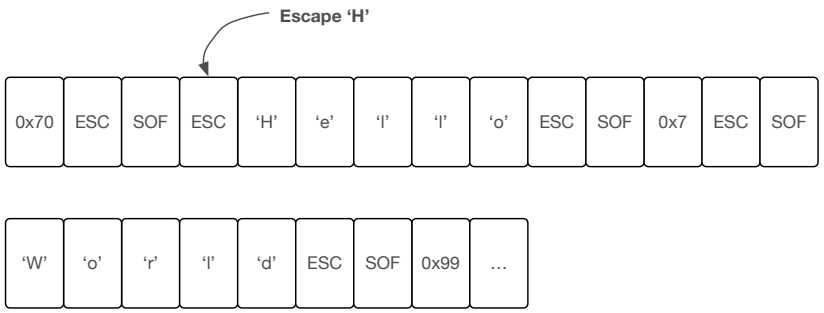
Coroutine example: A byte-stream parser

- Let's look at a byte-stream parser
- The protocol
  - ESC (`'H'`): Escape special bytes (commands) in the stream.
  - SOF (0x10) start of frame: Marks the beginning of a frame.
  - ESC + SOF mark the start of a frame.



Coroutine example: A byte-stream parser

- Let's look at a byte-stream parser
- The protocol
  - ESC (`'H'`): Escape special bytes (commands) in the stream.
  - SOF (0x10) start of frame: Marks the beginning of a frame.
  - ESC + SOF mark the start of a frame.
- The input is `Hello World.`



Coroutine example: A byte-stream parser

```
1  void ProcessNextByte(byte b, CompleteCb frameCompleted)
2  {
3    static std::string frame{};
4    static bool inHeader{}, wasESC{}, lookingForSOF{};
5
6    if(inHeader) {
7      if((ESC == b) && not wasESC) {
8        wasESC = true;
9        return;
10     } else if(wasESC) {
11       wasESC = false;
12
13       if((SOF == b) || (ESC != b)) {
14         // if b is not SOF discard the frame
15         if(SOF == b) { frameCompleted(frame); }
16         frame.clear();
17         inHeader = false;
18         return;
19       }
20     }
21     frame += static_cast<char>(b);
22
23   } else if((ESC == b) && not lookingForSOF) {
24     lookingForSOF = true;
25   } else if((SOF == b) && lookingForSOF) {
26     inHeader      = true;
27     lookingForSOF = false;
28   } else {
29     lookingForSOF = false;
30   }
31 }
```


Coroutine example: A byte-stream parser

```
1  async_generator<std::string, byte> Parse()
2  {
3    while(true) {
4      byte        b = co_await byte{};
5      std::string frame{};
6
7      if(ESC != b) { continue; }
8
9      A not looking at a start sequence
10     if(b = co_await byte{}; SOF != b) { continue; }
11
12     while(true) {  B capture the full frame
13       b = co_await byte{};
14
15       if(ESC == b) {
16         C skip this byte and look at the next one
17         b = co_await byte{};
18
19         if(SOF == b) {
20           co_yield frame;
21           break;
22         } else if(ESC != b) {
23           break;  D out of sync
24         }
25       }
26
27       frame += static_cast<char>(b);
28     }
29   }
30 }
```

Coroutine example: A byte-stream parser

- The new **co_await** needs some machinery.
- The compiler looks for specific names (functions) in that type:
  - await_transform allows a type to return an awaiter.
- An awaiter must provide:
  - await_ready: The coroutine finite state machine (FSM) asks whether the awaiter has data and is ready to continue.
  - await_resume: Once the awaiter signals ready, the coroutine FSM obtains the value from the awaiter by calling await_resume. The value is then passed to the awaiting part, i.e., **co_await**.
  - await_suspend: Called once the awaiter-coroutine gets suspended.

```cpp
template<typename T>
struct Awaitable {
  std::optional<T> mRecentSignal{};

  [[nodiscard]] auto await_transform(T)
  {
    struct awaiter {
      std::optional<T>& mRecentSignal;

      bool await_ready() const
      {
        return mRecentSignal.has_value();
      }

      void await_suspend(std::coroutine_handle<>) {}

      T await_resume()
      {
        assert(mRecentSignal.has_value());
        auto tmp = *mRecentSignal;
        mRecentSignal.reset();
        return tmp;
      }
    };

    return awaiter{mRecentSignal};
  }
};
```


Coroutine example: A byte-stream parser

```cpp
template<typename T, typename U>
struct [[nodiscard]] async_generator {
  using promise_type = promise_type_base<T, async_generator, Awaitable<U>>;

  T operator()() { return std::exchange(mCoroHdl.promise().mValue, {}); }

  void SendSignal(U signal)
  {
    mCoroHdl.promise().mRecentSignal = signal;
    if(not mCoroHdl.done()) { mCoroHdl.resume(); }
  }

  async_generator(async_generator && rhs) : mCoroHdl{std::exchange(rhs.mCoroHdl, nullptr)} {}
  ~async_generator() { if(mCoroHdl) { mCoroHdl.destroy(); } }

private:
  friend promise_type;
  using Handle = std::coroutine_handle<promise_type>;

  explicit async_generator(promise_type * p)
  : mCoroHdl(Handle::from_promise(*p)) {}

  Handle mCoroHdl;
};
```

Coroutine example: A byte-stream parser

- To compose with `Awaitble` `promise_type_base` takes a variadic type pack from which it then derives.
- The variadic pack allows the pack to be empty!

```
1  template<typename T,
2           typename G,
3           typename... Bases>  A Allow multiple bases for awaiter
4  struct promise_type_base : public Bases... {
5    T mValue;
6
7    auto yield_value(T value)
8    {
9      mValue = std::move(value);
10     return std::suspend_always{};
11   }
12
13   G get_return_object() { return G{this}; };
14
15   std::suspend_always initial_suspend() { return {}; }
16   std::suspend_always final_suspend() noexcept
17   {
18     return {};
19   }
20
21   void return_void() {}
22   void unhandled_exception() { std::terminate(); }
23 };
```


Coroutine example: A byte-stream parser

```
1  generator<byte> sender(std::vector<byte> fakeBytes)
2  {
3    for(const auto& b : fakeBytes) { co_yield b; }
4  }
5
6  void ProcessStream(generator<byte>& stream, async_generator<std::string, byte>& parse)
7  {
8    for(const auto& b : stream) {
9      A Send the new byte to the waiting Parse coroutine
10     parse.SendSignal(b);
11
12     B Check whether we have a complete frame
13     if(const auto& res = parse(); res.length()) {
14       HandleFrame(res);
15     }
16   }
17 }
```

Coroutine example: A byte-stream parser

```
1  std::vector<byte> fakeBytes1{0x70_B, ESC, SOF, ESC, 'H'_B, 'e'_B, 'l'_B, 'l'_B, 'o'_B, ESC, SOF, 0x7_B, ESC, SOF};
2
3  A Simulate the first network stream
4  auto stream1 = sender(std::move(fakeBytes1));
5
6  B Create the Parse coroutine and store the handle in p
7  auto p = Parse();
8
9  C Process the bytes
10 ProcessStream(stream1, p);
11
12 D Simulate the reopening of the network stream
13 std::vector<byte> fakeBytes2{'W'_B, 'o'_B, 'r'_B, 'l'_B, 'd'_B,  ESC, SOF, 0x99_B};
14
15 E Simulate a second network stream
16 auto stream2 = sender(std::move(fakeBytes2));
17
18 F We still use the former p and feed it with new bytes
19 ProcessStream(stream2, p);
```

Ranges

Ranges: for_each

C++20

```
1 const std::vector<int> v{2, 3, 4, 5};
2
3 std::for_each(v.begin(), v.end(), [](auto e) { std::cout << e << '\n'; });
```

Ranges: for_each

C++20

```
1 const std::vector<int> v{2, 3, 4, 5};
2
3 std:: ranges:: for_each(v, [](auto e) { std::cout << e << '\n'; });
```

Ranges: for_each

C++20

```
1 std::ranges::for_each( std::as_const(v) ,
2                        [](auto e) { std::cout << e << '\n'; });
```


Ranges: for_each

C++20

```
1 std::ranges::for_each(v  | std::views::reverse ,
2                        [](auto e) { std::cout << e << '\n'; });
```

```
1 for(auto e : v  | std::views::reverse ) { std::cout << e << '\n'; }
```

Ranges: get a range of numbers

C++20

```
1 for(int i{}; i < 5; ++i) { std::cout << i << '\n'; }
```


Ranges: get a range of numbers

C++20

```
1 A Attention: iota runs to N-1
2 for(const auto& i : std::views::iota(0, 5)) { std::cout << i << '\n'; }
```

Ranges: counting words

C++20

```
1  const auto text{"Hello, C++20! How are we doing?"sv};
2
3  auto words = std::count_if(text.begin(), text.end(), [](const auto& c) {
4    return std::isspace(c);
5  });
6
7  A We've counted only spaces, and a length != 0 means there is a word
8  if(text.length()) { ++words; }
9
10 std::cout << "Words: " << words << '\n';
```


Ranges: counting words

C++20

```
1  const auto text{"Hello, C++20! How are we doing?"sv};
2
3  A Note that words can be const now
4  const auto words = std::ranges::distance(text | std::views::split(' '));
5
6  std::cout << "Words: " << words << '\n';
```

Spaceship

<=>

The goal

C++20

- Creating a value-type.

```
1 struct Point {
2   int x;
3   int y;
4 };
```

- Using it.

```
1 constexpr Point a{2, 3};
2 constexpr Point b{2, 3};
3 constexpr Point c{3, 3};
4
5 static_assert(a == b);
6 static_assert(not(a != b));
7 static_assert(not(a == c));
8 static_assert(a != c);
```

The spaceship operator: <=>

C++20

```
1 struct Point {
2   int x;
3   int y;
4
5   constexpr bool operator==(const Point& rhs) const noexcept { return (x == rhs.x) && (y == rhs.y); }
6   constexpr bool operator!=(const Point& rhs) const noexcept { return not(*this == rhs); }
7 };
```


Are you tired writing code like this?

C++20

```
1 bool operator!=(const T& t) { return not (*this == t); }
```

The spaceship operator: <=>

C++20

```
1 struct Point {
2   int x;
3   int y;
4
5   constexpr bool operator==(const Point& rhs) const noexcept { return (x == rhs.x) && (y == rhs.y); }
6 };
```


The spaceship operator: <=>

C++20

```
1 struct Point {
2   int x;
3   int y;
4
5   bool operator==(const Point&) const = default;
6 };
```

The spaceship operator: <=>

C++20

```
1 constexpr bool operator==(const QPoint& lhs, const Point& rhs) noexcept
2 {
3   return (lhs.x() == rhs.x) && (lhs.y() == rhs.y);
4 }
5
6 constexpr bool operator==(const Point& lhs, const QPoint& rhs) noexcept { return (rhs == lhs); }
7 constexpr bool operator!=(const QPoint& lhs, const Point& rhs) noexcept { return not(lhs == rhs); }
8 constexpr bool operator!=(const Point& lhs, const QPoint& rhs) noexcept { return (rhs != lhs); }
```

The spaceship operator: <=>

C++20

```
1 constexpr bool operator==(const QPoint& lhs, const Point& rhs) noexcept
2 {
3   return (lhs.x() == rhs.x) && (lhs.y() == rhs.y);
4 }
```

The spaceship operator: <=>

C++20

```
1 struct Point {
2   int x;
3   int y;
4
5   constexpr bool operator==(const Point& rhs) const noexcept { return (x == rhs.x) && (y == rhs.y); }
6
7   constexpr bool operator<(const Point& rhs) const noexcept
8   { return (x < rhs.x) || ((x == rhs.x) && (y < rhs.y)); }
9
10  constexpr bool operator>(const Point& rhs) const noexcept { return not(*this <= rhs); }
11  constexpr bool operator<=(const Point& rhs) const noexcept { return (*this == rhs) || (*this < rhs); }
12  constexpr bool operator>=(const Point& rhs) const noexcept { return (*this == rhs) || (*this > rhs); }
13 };
```


The spaceship operator: <=>

C++20

```
1 struct Point {
2   int x;
3   int y;
4
5   bool operator==(const Point&) const  = default;
6   auto operator<=>(const Point&) const = default;
7 };
```

Concepts

Dependent destructor

```
1  struct COMLike {
2    ~COMLike() {}   A Make it not trivially destructible
3
4    void Release();   B Release all data
5
6    // Some data fields
7  };
8
9  struct TriviallyDestructible {};   C A type which is trivially destructible
10
11 static_assert(not std::is_trivially_destructible_v<Wrapper<COMLike>>);
12 static_assert(std::is_trivially_destructible_v<Wrapper<TriviallyDestructible>>);
```

Dependent destructor

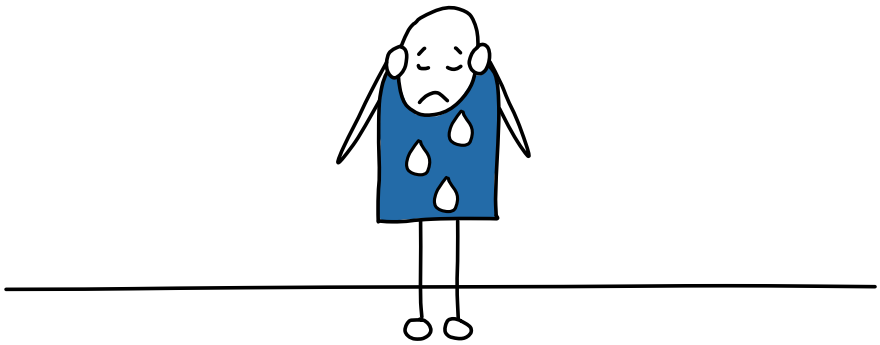
```
1  template<typename T, typename = void>
2  struct has_release : std::false_type {};
3
4  template<typename T>
5  struct has_release<T, decltype(std::declval<T>().Release())> : std::true_type {};
6
7  template<typename T>
8  class Wrapper {
9    T mData;
10
11 public:
12   ~Wrapper()
13   {
14     if constexpr(has_release<T>::value) { mData.Release(); }
15   }
16 };
```

Dependent destructor

```
1  template<typename T>
2  class Wrapper {
3    T mData;
4
5  public:
6    ~Wrapper() requires requires(T t) { t.Release(); }
7    {
8      mData.Release();
9    }
10
11   ~Wrapper() = default;
12 };
```

Error messages

```
1  template<typename T, typename U = void>  A
2  struct is_container : std::false_type {};
3
4  template<typename T>
5  struct is_container<
6    T,
7    std::void_t<typename T::value_type,  B
8                typename T::size_type,
9                typename T::allocator_type,
10               typename T::iterator,
11               typename T::const_iterator,
12               decltype(std::declval<T>().size()),
13               decltype(std::declval<T>().begin()),
14               decltype(std::declval<T>().end()),
15               decltype(std::declval<T>().cbegin()),
16               decltype(std::declval<T>().cend())>>
17 : std::true_type {};
18
19 struct A {};
20
21 static_assert(!is_container<A>::value);  C
22 static_assert(is_container<std::vector<int>>::value);
```

Error messages: Now helpful

```
1  template<typename T>
2  concept container = requires(T t)
3  {
4    typename T::value_type;
5    typename T::size_type;
6    typename T::allocator_type;
7    typename T::iterator;
8    typename T::const_iterator;
9    t.size();
10   t.begin();
11   t.end();
12   t.cbegin();
13   t.cend();
14 };
15
16 struct A {};
17
18 static_assert(not container<A>);
19 static_assert(container<std::vector<int>>);
```

Andreas Fertig
v1.0

C++20: What's in there for you

37

Andreas Fertig
v1.0

C++20: What's in there for you

38

https://AndreasFertig.Info
post@AndreasFertig.Info

constexpr & friends

C++20

std::is_constant_evaluated

```
1 constexpr bool Fun()
2 {
3   Log("%s\n", "hello");
4
5   return true;
6 }
```

Use:

```
1 auto           a = Fun();
2 constexpr auto b = Fun();
```

std::is_constant_evaluated

C++20

```
1 template<typename... Args>
2 constexpr void Log(std::string_view fmt, const Args&... args)
3 {
4   if(not std::is_constant_evaluated()) { printf(fmt.data(), args...); }
5 }
```


std::is_constant_evaluated

C++20

```
1 template<typename... Args>
2 constexpr void Log(std::string_view fmt, const Args&... args)
3 {
4   A The constexpr is bad, it doesn't do what is intended
5   if  constexpr (not std::is_constant_evaluated()) {
6       printf(fmt.data(), args...);
7   }
8 }
```

Static Initialization Order Fiasco

```
1  struct Air {
2    Air(int amount)
3    : _amount{amount}
4    {}
5
6    void Consume(int v) { _amount -= v; }
7    int  Available() const { return _amount; }
8
9  private:
10   int _amount;
11 };
12
13 struct Human {
14   Human(int breath);
15 };
16
17 Air   air{9};     A Create global air object
18 Human human{5};  B Create global human object
19
20 Human::Human(int breath)
21 {
22   air.Consume(breath);  C Depends on air
23 }
```

Static Initialization Order Fiasco

```
1  struct Air {
2    Air(int amount)
3    : _amount{amount}
4    {}
5
6    void Consume(int v) { _amount -= v; }
7    int  Available() const { return _amount; }
8
9  private:
10   int _amount;
11 };
12
13 struct Human {
14   Human(int breath);
15 };
16
17 Human human{5};  B Create global human object
18 Air   air{9};     A Create global air object
19
20 Human::Human(int breath)
21 {
22   air.Consume(breath);  C Depends on air
23 }
```

Static Initialization Order Fiasco

```
1  struct Air {
2     constexpr  Air(int amount)
3    : _amount{amount}
4    {}
5
6    void Consume(int v) { _amount -= v; }
7    int  Available() const { return _amount; }
8
9  private:
10   int _amount;
11 };
```


Static Initialization Order Fiasco

```
1  struct Air {
2     constexpr  Air(int amount)
3    : _amount{amount}
4    {}
5
6    void Consume(int v) { _amount -= v; }
7    int  Available() const { return _amount; }
8
9  private:
10   int _amount;
11 };
```

```
1  Human          human{5};   B Create global human object
2   constinit   Air air{9};   A Create global air object
```

C++20

consteval

```
1 constexpr int Calc(int x)  A
2 {
3   return 4 * x;
4 }
5
6 int main()
7 {
8   auto res = Calc(2);  B
9 }
```


C++20

consteval

```
1 constexpr int Calc(int x)  A
2 {
3   return 4 * x;
4 }
5
6 int main()
7 {
8   constexpr auto res = Calc(2);  B
9 }
```

C++20

consteval

```
1  consteval  int Calc(int x)  A
2 {
3   return 4 * x;
4 }
5
6 int main()
7 {
8   auto res = Calc(2);  B
9
10  ++res;  C Modify res at run-time
11 }
```


C++20

consteval

```
1 consteval auto as_constant(auto value)
2 {
3   return value;
4 }
```

consteval

C++20

```
1  constexpr int Calc(int x)   A constexpr again
2  {
3    return 4 * x;
4  }
5
6  int main()
7  {
8    B Forcing compile-time with as_constant
9    auto res = as_constant(Calc(2));
10
11   ++res;   C Modify res at run-time
12
13   res = Calc(res);   D Run-time use of Calc
14 }
```


constexpr and dynamic memory

C++20

```
1  struct Car {   A Base class for all cars
2    virtual ~Car()             = default;
3    virtual int speed() const = 0;
4  };
5
6  B Various concrete cars with individual speed
7  struct Mercedes : Car {
8    int speed() const override { return 5; }
9  };
10 struct Toyota : Car {
11   int speed() const override { return 6; }
12 };
13 struct Tesla : Car {
14   int speed() const override { return 9; }
15 };
16
17 C A factory function to create a car
18 Car* CreateCar(int i)
19 {
20   switch(i) {
21     case 0: return new Mercedes{}; break;
22     case 1: return new Toyota{}; break;
23     case 2: return new Tesla{}; break;
24   }
25
26   return nullptr;
27 }
```

C++20

constexpr and dynamic memory

```
1  int FastestCar()
2  {
3    int max   = -1;
4    int maxId = -1;
5
6    for(int i = 0; i < 3; ++i) {
7      const auto* car = CreateCar(i);
8
9      if(car->speed() > max) {
10       max   = car->speed();
11       maxId = i;
12     }
13
14     delete car;
15   }
16
17   return maxId;
18 }
```

C++20

constexpr and dynamic memory

```
1  struct Car {   A Base class for all cars
2    virtual ~Car()                          = default;
3    constexpr virtual int speed() const = 0;
4  };
5
6  B Various concrete cars with individual speed
7  struct Mercedes : Car {
8    constexpr int speed() const override { return 5; }
9  };
10 struct Toyota : Car {
11   constexpr int speed() const override { return 6; }
12 };
13 struct Tesla : Car {
14   constexpr int speed() const override { return 9; }
15 };
16
17 C A factory function to create a car
18 constexpr Car* CreateCar(int i)
19 {
20   switch(i) {
21     case 0: return new Mercedes{}; break;
22     case 1: return new Toyota{}; break;
23     case 2: return new Tesla{}; break;
24   }
25
26   return nullptr;
27 }
```

constexpr and dynamic memory

C++20

```
1  constexpr int FastestCar()
2  {
3    int max   = -1;
4    int maxId = -1;
5
6    for(const int i : std::views::iota(0, 3)) {
7      const auto* car = CreateCar(i);
8
9      if(car->speed() > max) {
10       max   = car->speed();
11       maxId = i;
12     }
13
14     delete car;
15   }
16
17   return maxId;
18 }
```


constexpr and dynamic memory

C++20

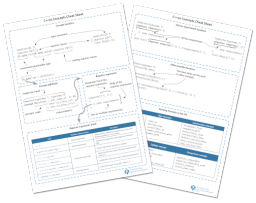
Hopefully in C++23:

```
1  constexpr int FastestCar()
2  {
3    int max   = -1;
4    int maxId = -1;
5
6    for(const int i : std::views::iota(0, 3)) {
7      if(const std::unique_ptr car{CreateCar(i)};
8         car->speed() > max) {
9        max   = car->speed();
10       maxId = i;
11     }
12   }
13
14   return maxId;
15 }
```

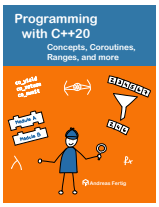
}

I am Fertig.

C++20 Concepts Cheat Sheet



fertig.to/subscribe



fertig.to/bpwcpp20

Used Compilers & Typography

Used Compilers

- Compilers used to compile (most of) the examples.
  - g++ 11.1.0
  - clang version 13.0.0

Typography

- Main font:
  - Camingo Dos Pro by Jan Fromm (https://janfromm.de/)
- Code font:
  - CamingoCode by Jan Fromm licensed under Creative Commons CC BY-ND, Version 3.0 http://creativecommons.org/licenses/by-nd/3.0/

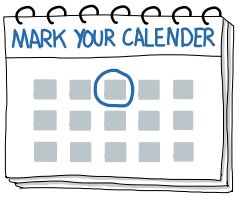
References

Images:
36: Franziska Panter
38: Franziska Panter
60: Franziska Panter

Upcoming Events

Training Classes

- *C++ Clean Code – Best Practices für Programmierer*, golem Akademie, March 29 - 31
- *Programming with C++11 to C++17*, Andreas Fertig, April 25 - 29
- *Programming with C++20*, Andreas Fertig, July 11 - 15
- *Programmieren mit C++20*, Andreas Fertig, October 05 - 07
- *Programming with C++11 to C++17*, Andreas Fertig, November 07 - 11



For my upcoming talks you can check https://andreasfertig.info/talks/.
For my courses you can check https://andreasfertig.info/courses/.
Like to always be informed? Subscribe to my newsletter: https://andreasfertig.info/newsletter/.

About Andreas Fertig

Photo: Kristijan Matic www.kristijanmatic.de

Andreas Fertig, CEO of Unique Code GmbH, is an experienced trainer and lecturer for C++ for standards 11 to 20.
Andreas is involved in the C++ standardization committee, in which the new standards are developed. At international conferences, he presents how code can be written better. He publishes specialist articles, e.g., for iX magazine, and has published several textbooks on C++.
With C++ Insights (https://cppinsights.io), Andreas has created an internationally recognized tool that enables users to look behind the scenes of C++ and thus to understand constructs even better.
Before working as a trainer and consultant, he worked for Philips Medizin Systeme GmbH for ten years as a C++ software developer and architect focusing on embedded systems.