Training Material

Programming with C++20

Online
2021-11-01

Andreas Fertig

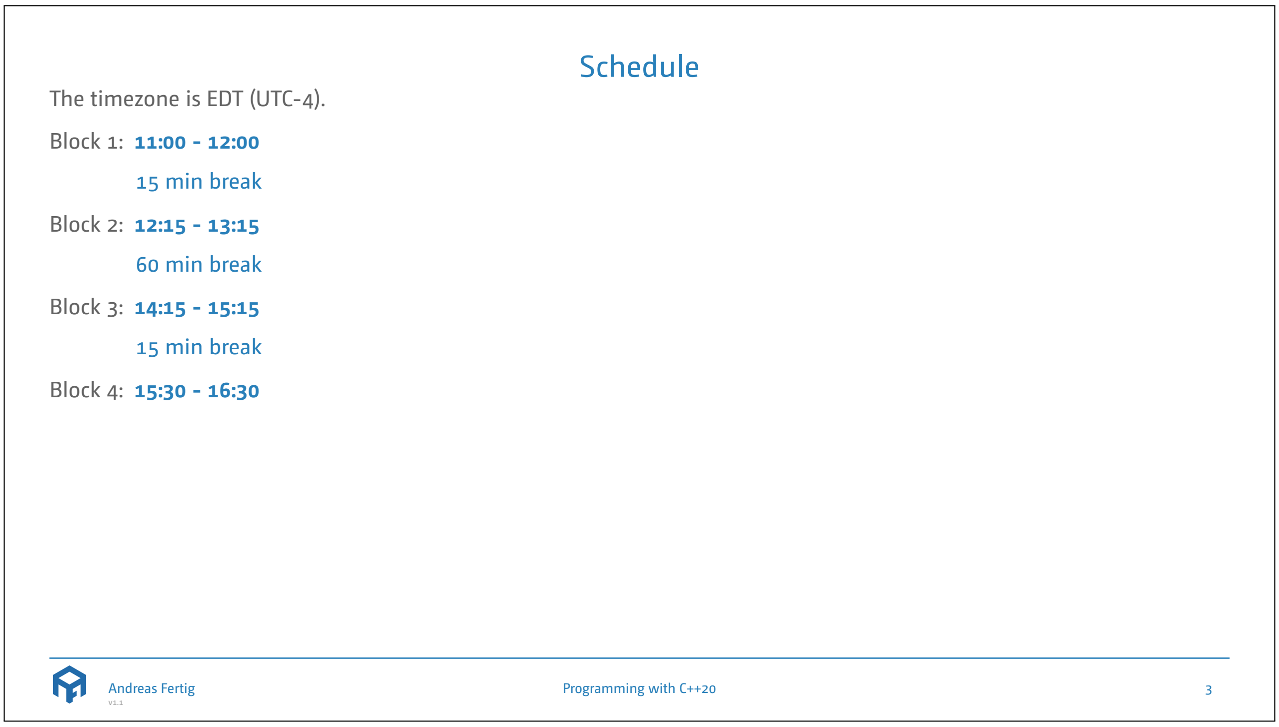
Write unique code!

Version: v1.1

Planung, Satz und Einbandentwurf:
Andreas Fertig

Herstellung und Verlag:
Andreas Fertig

1. main()



Schedule

The timezone is EDT (UTC-4).

Block 1: **11:00 - 12:00**

15 min break

Block 2: **12:15 - 13:15**

60 min break

Block 3: **14:15 - 15:15**

15 min break

Block 4: **15:30 - 16:30**

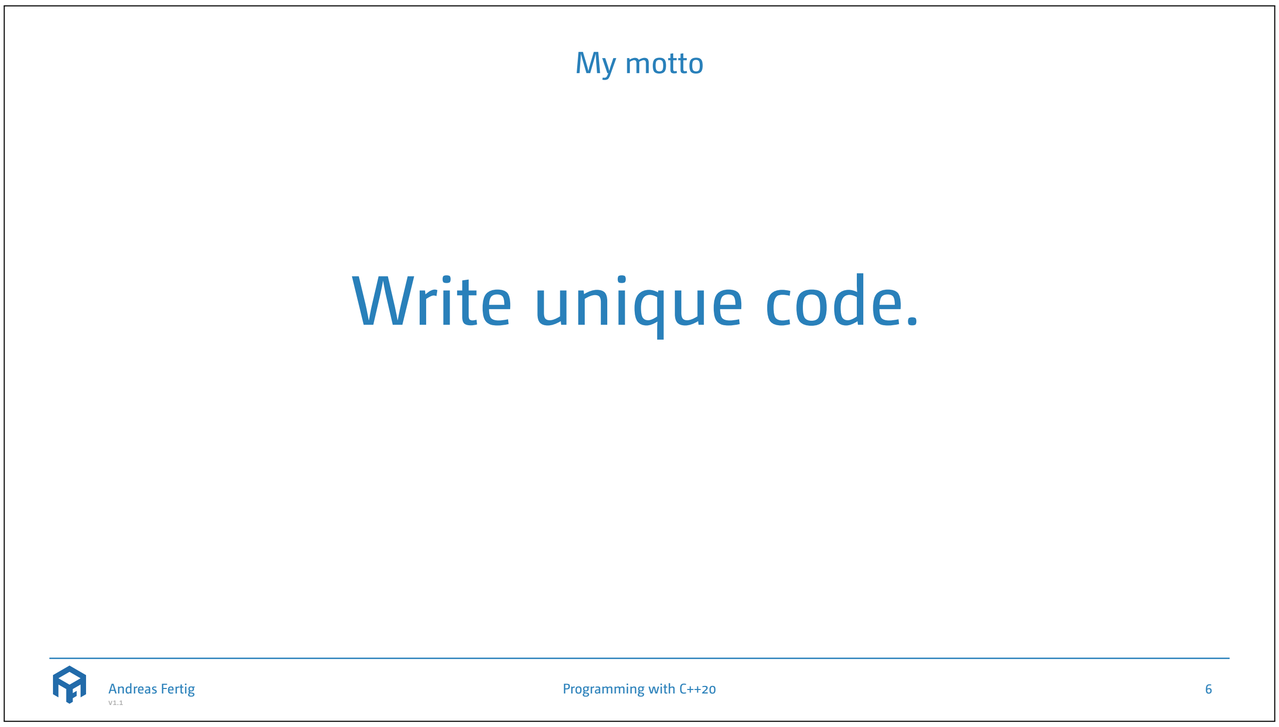
A note on live virtual classes

- Please turn your video on.
- All sessions start with audio off.
- Please keep your audio off and turn it on for questions only.
- For questions, either ask them directly via voice or use non-verbal communication like hand-raising.
- The internet is unstable sometimes
  - Either you or I might get disconnected due to some networking issues.
  - If I get disconnected, I try to get back as quickly as possible. The session should stay open. Just wait. Should there be a significant incident, I will call in via phone or write an email to inform you.
  - If one of you gets disconnected in the middle of a question, I will wait for a little if I realize it, and I will continue with you as soon as you are back.

Sample code disclaimer

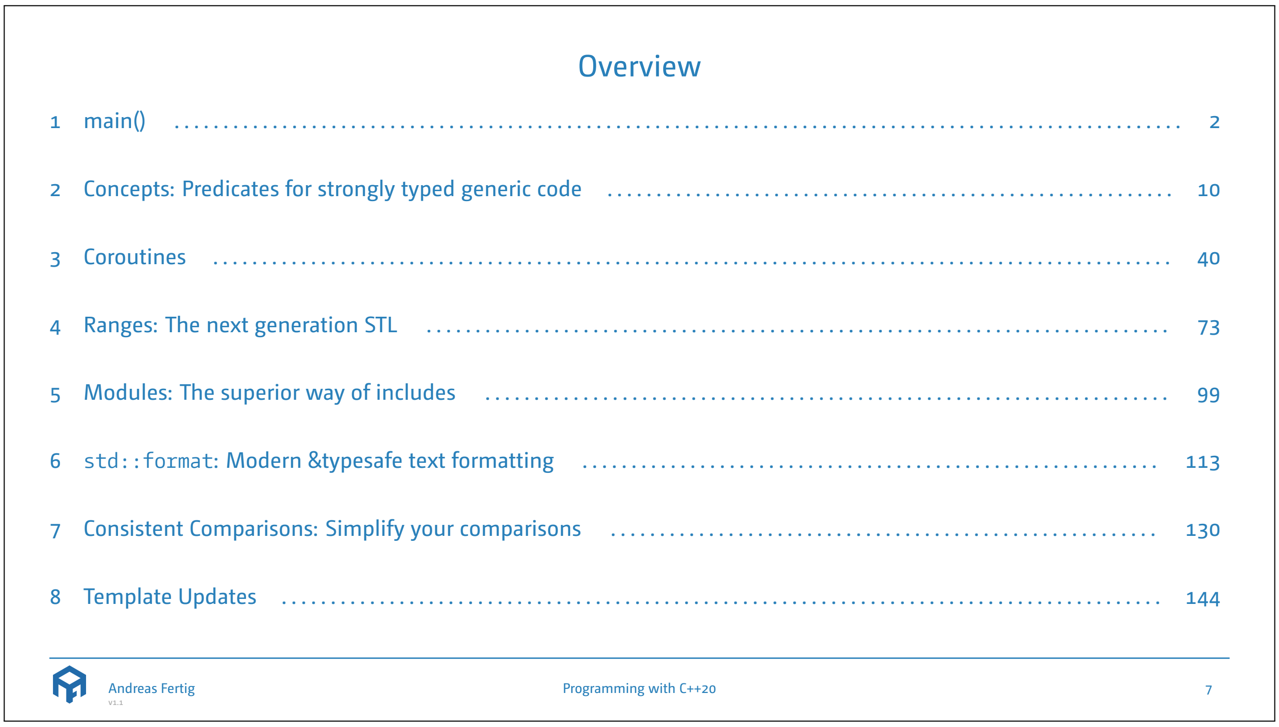
The source code examples in this material can be used without any warranty.

Please keep in mind that some of this code may be untested.

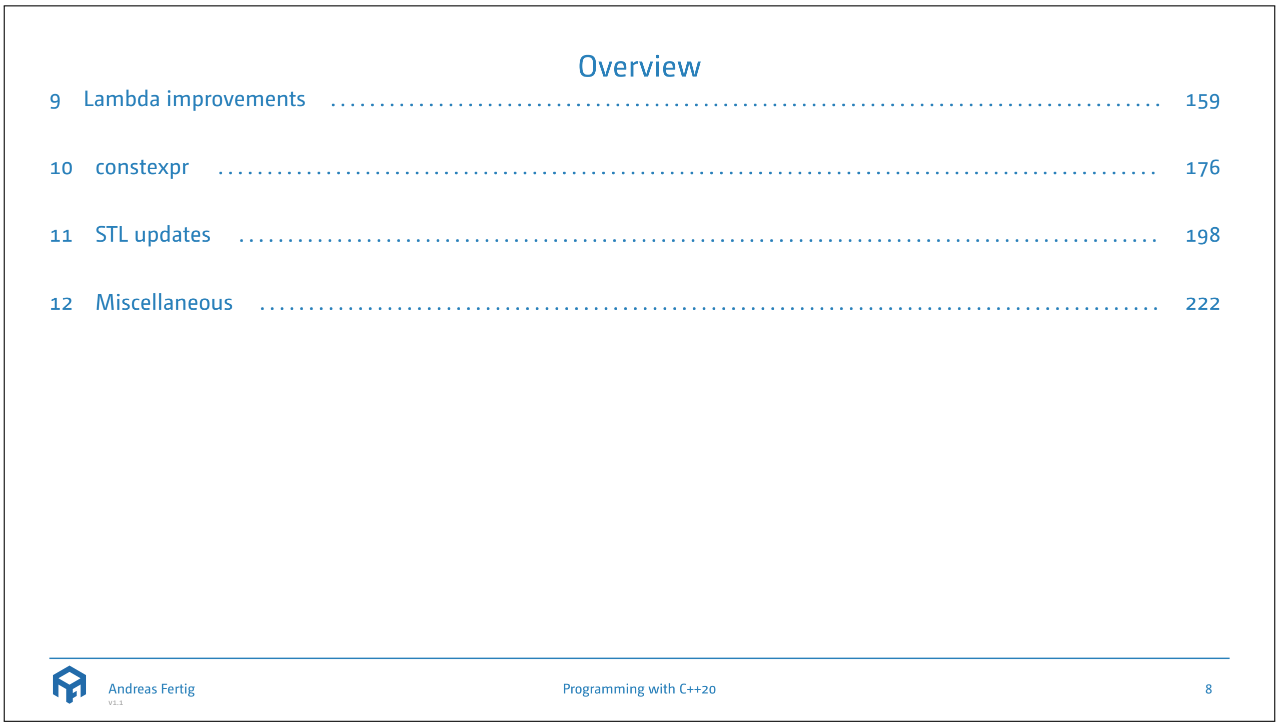


My motto

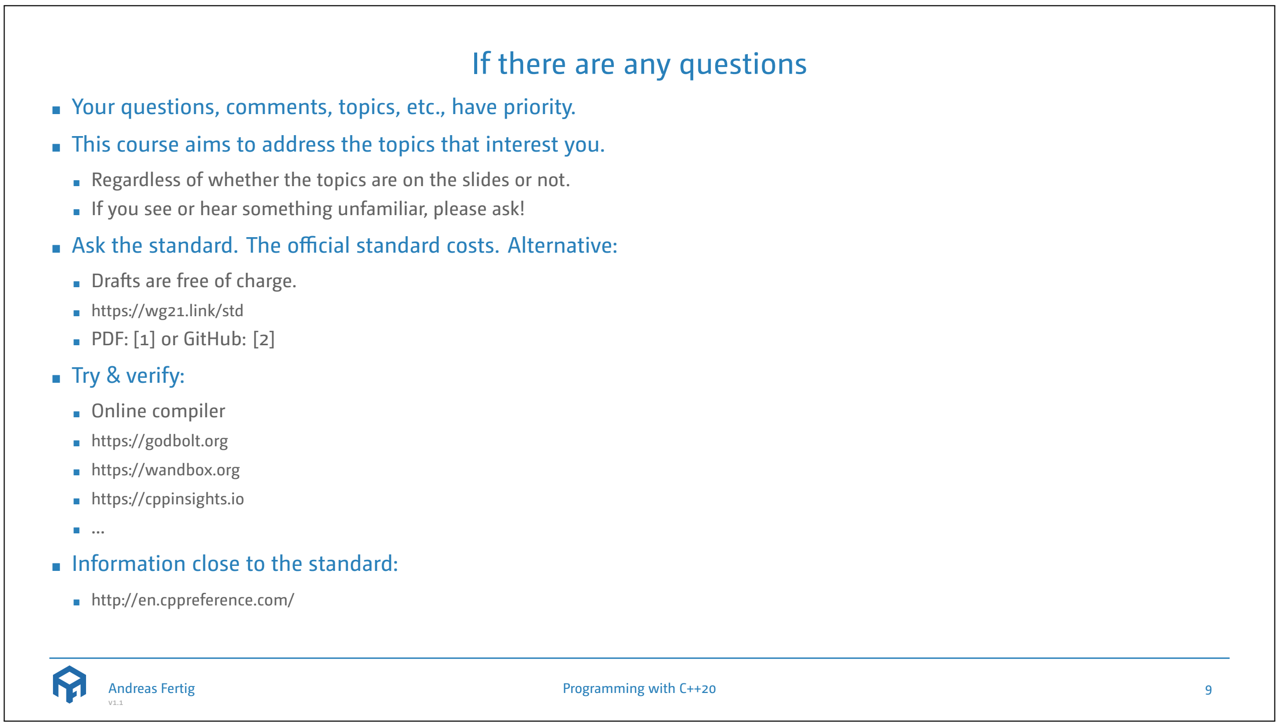
Write unique code.



Overview



Overview



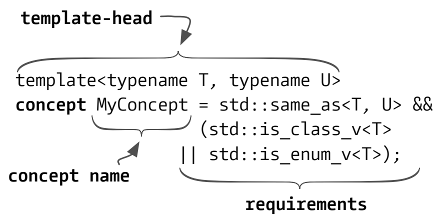
If there are any questions

- Your questions, comments, topics, etc., have priority.
- This course aims to address the topics that interest you.
  - Regardless of whether the topics are on the slides or not.
  - If you see or hear something unfamiliar, please ask!
- Ask the standard. The official standard costs. Alternative:
  - Drafts are free of charge.
  - https://wg21.link/std
  - PDF: [1] or GitHub: [2]
- Try & verify:
  - Online compiler
  - https://godbolt.org
  - https://wandbox.org
  - https://cppinsights.io
  - ...
- Information close to the standard:
  - http://en.cppreference.com/

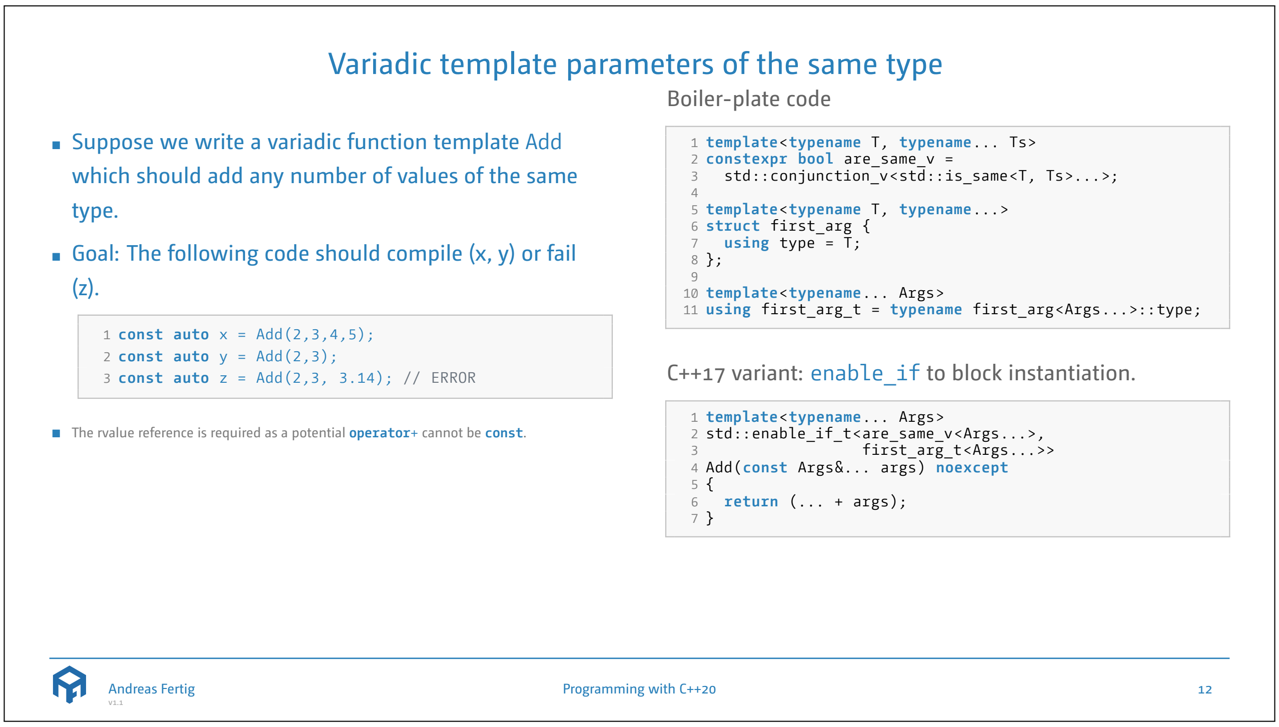
2. Concepts: Predicates for strongly typed generic code

Concepts

- With Concepts, we can formulate requirements for a type.
  - Comparable to `std::enable_if`.
  - Concepts consist of the definition of the concept (**`concept`**) and requirements (**`requires`**):
  - A concept is always a template and can be recognized by the new keyword **`concept`**. A concept itself consists of other concepts or requirements. The latter is defined by the keyword **`requires`**.



```
template<typename T, typename U>
concept MyConcept = std::same_as<T, U> &&
                    (std::is_class_v<T>
                  || std::is_enum_v<T>);
```



Variadic template parameters of the same type

- Suppose we write a variadic function template Add which should add any number of values of the same type.
- Goal: The following code should compile (x, y) or fail (z).

```cpp
const auto x = Add(2,3,4,5);
const auto y = Add(2,3);
const auto z = Add(2,3, 3.14); // ERROR
```

- The rvalue reference is required as a potential **operator+** cannot be **const**.

Boiler-plate code

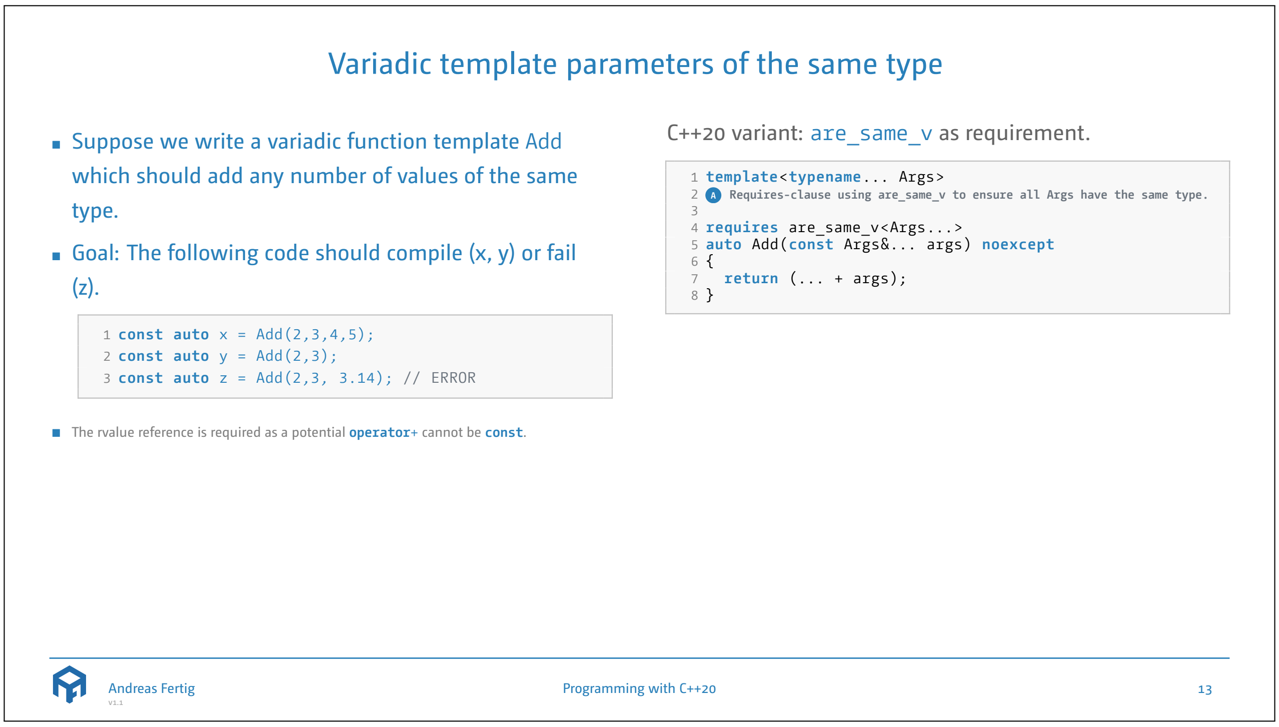
```cpp
template<typename T, typename... Ts>
constexpr bool are_same_v =
  std::conjunction_v<std::is_same<T, Ts>...>;

template<typename T, typename...>
struct first_arg {
  using type = T;
};

template<typename... Args>
using first_arg_t = typename first_arg<Args...>::type;
```

C++17 variant: `enable_if` to block instantiation.

```cpp
template<typename... Args>
std::enable_if_t<are_same_v<Args...>,
                 first_arg_t<Args...>>
Add(const Args&... args) noexcept
{
  return (... + args);
}
```



Variadic template parameters of the same type

- Suppose we write a variadic function template Add which should add any number of values of the same type.
- Goal: The following code should compile (x, y) or fail (z).

```cpp
const auto x = Add(2,3,4,5);
const auto y = Add(2,3);
const auto z = Add(2,3, 3.14); // ERROR
```

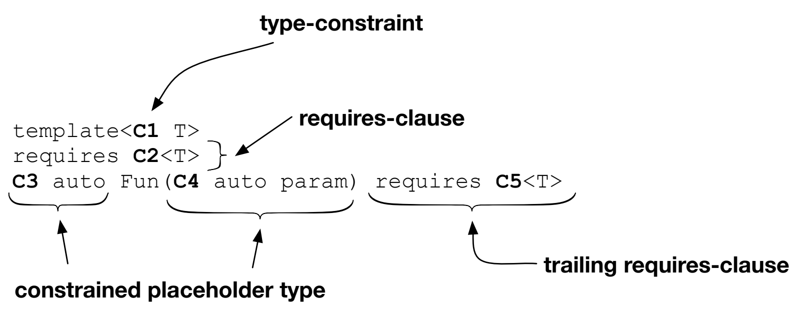
- The rvalue reference is required as a potential **operator+** cannot be **const**.

C++20 variant: `are_same_v` as requirement.

```cpp
template<typename... Args>
A Requires-clause using are_same_v to ensure all Args have the same type.

requires are_same_v<Args...>
auto Add(const Args&... args) noexcept
{
  return (... + args);
}
```

Application areas for Concepts

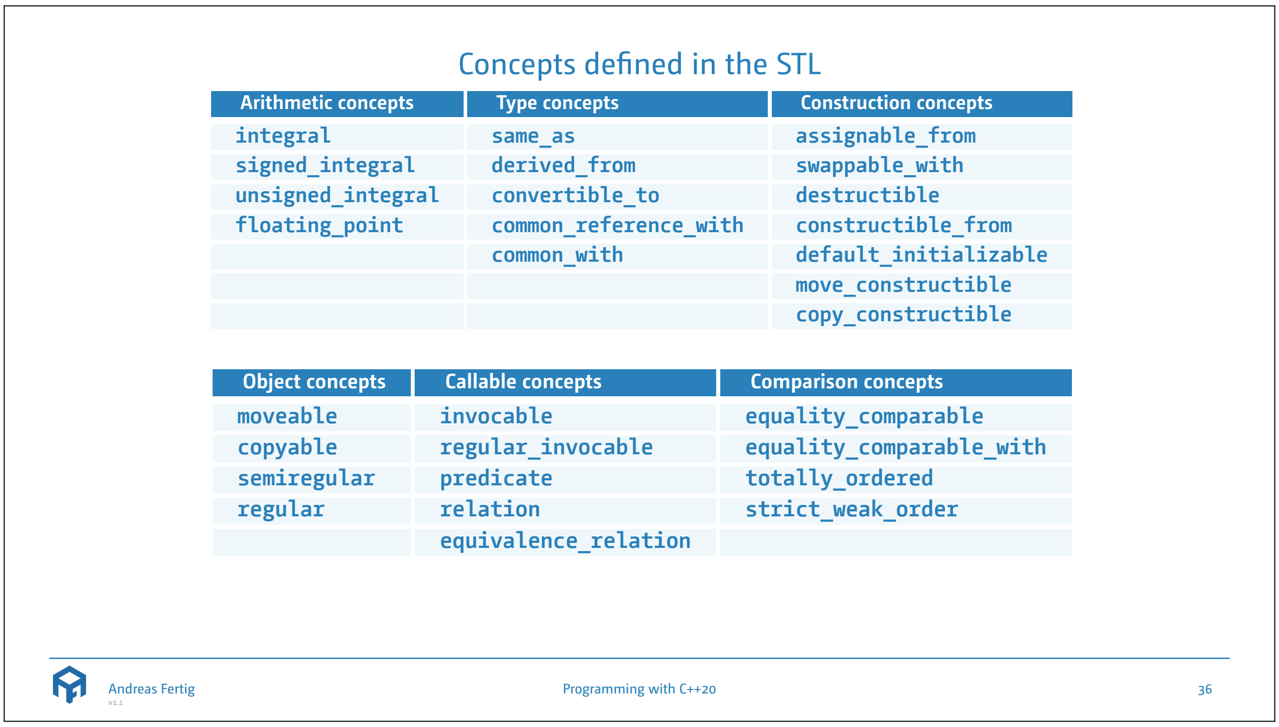


```
template<C1 T>
requires C2<T>
C3 auto Fun(C4 auto param) requires C5<T>
```

Type	Use this
type-constraint	Use this option if you already know that a template type parameter has a certain limitation. For example, not all types are allowed.
requires-clause	Use this option if you need to add restrictions for multiple type or non-type template parameters.
trailing requires-clause	Use this on a function in a class template to constrain the function based on the class template parameters.

There is more

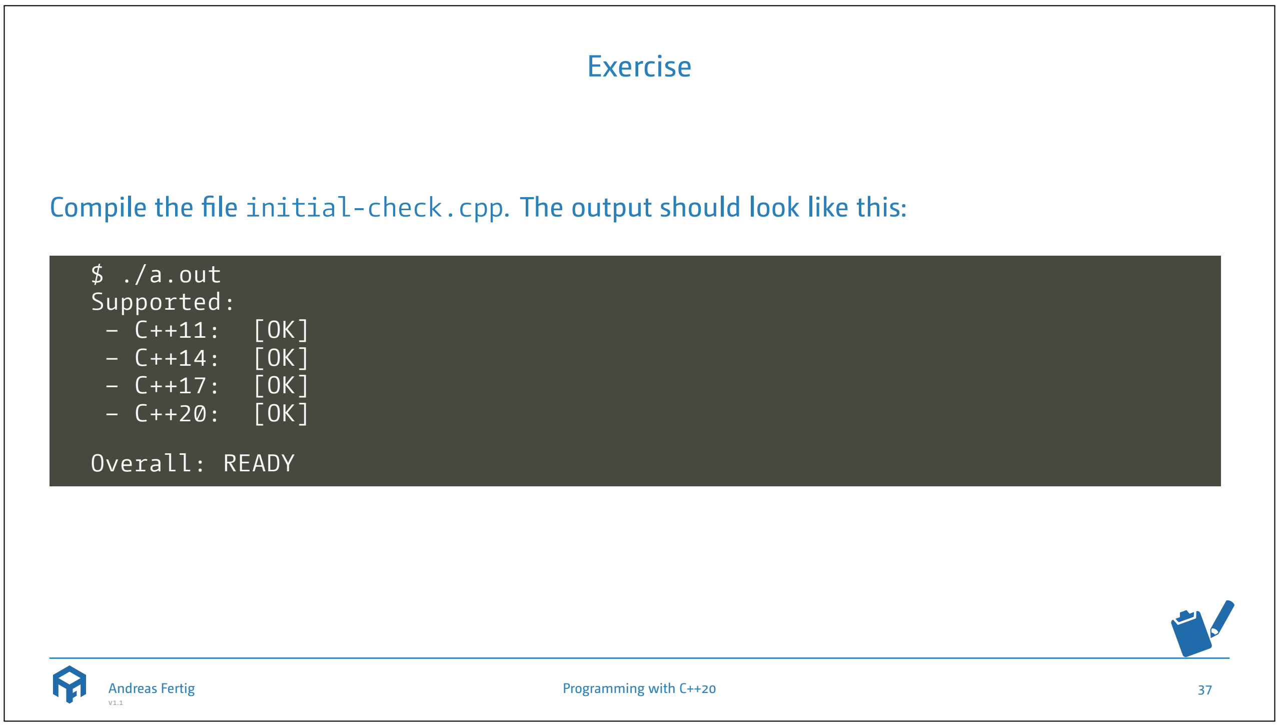
- Currently, Add only prevents
  - A mixed types.
- The current version of Add leaves a lot unspecified:
  - B Add can nonsensically be called with only one parameter.
  - C The type used in Args must support the + operation.
  - D The operation + should be **`noexcept`** since Add itself is **`noexcept`**.
  - E The return type of the operation + should match that of Args.



Concepts defined in the STL

Arithmetic concepts	Type concepts	Construction concepts
integral	same_as	assignable_from
signed_integral	derived_from	swappable_with
unsigned_integral	convertible_to	destructible
floating_point	common_reference_with	constructible_from
	common_with	default_initializable
		move_constructible
		copy_constructible

Object concepts	Callable concepts	Comparison concepts
moveable	invocable	equality_comparable
copyable	regular_invocable	equality_comparable_with
semiregular	predicate	totally_ordered
regular	relation	strict_weak_order
	equivalence_relation	

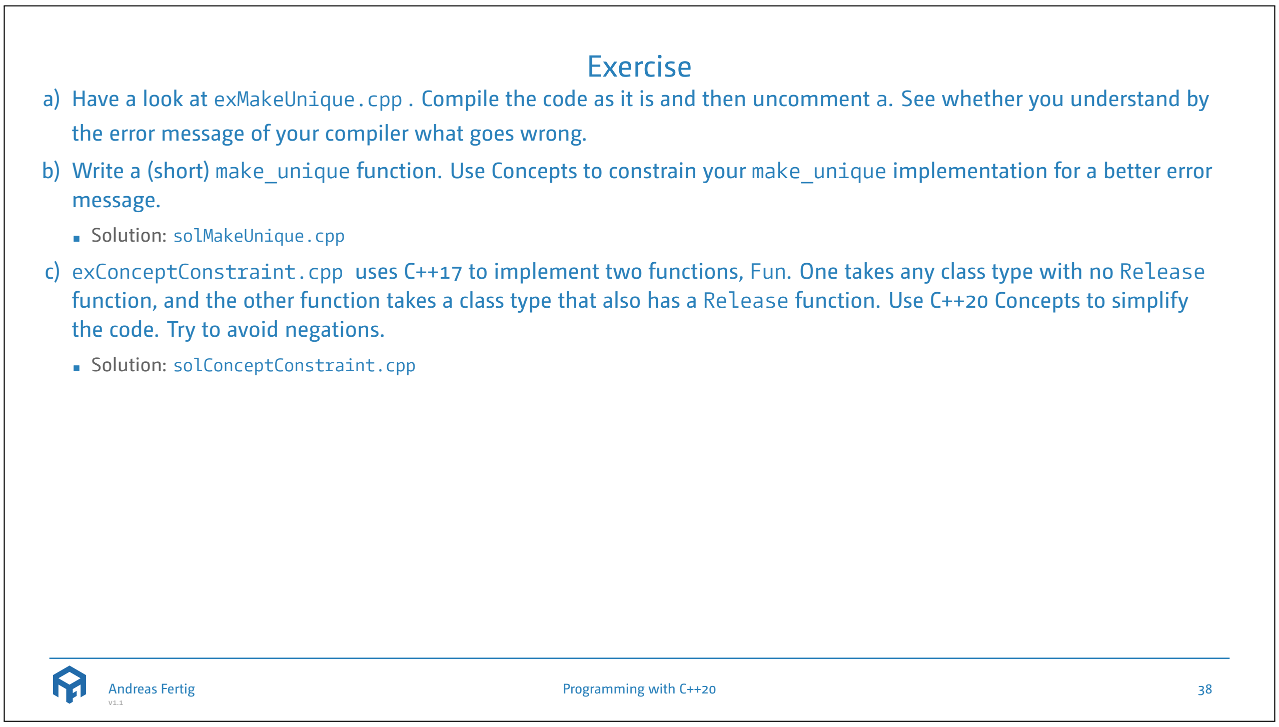


Exercise

Compile the file `initial-check.cpp`. The output should look like this:

```
$ ./a.out
Supported:
 - C++11:  [OK]
 - C++14:  [OK]
 - C++17:  [OK]
 - C++20:  [OK]

Overall: READY
```



Exercise

a) Have a look at `exMakeUnique.cpp` . Compile the code as it is and then uncomment a. See whether you understand by the error message of your compiler what goes wrong.

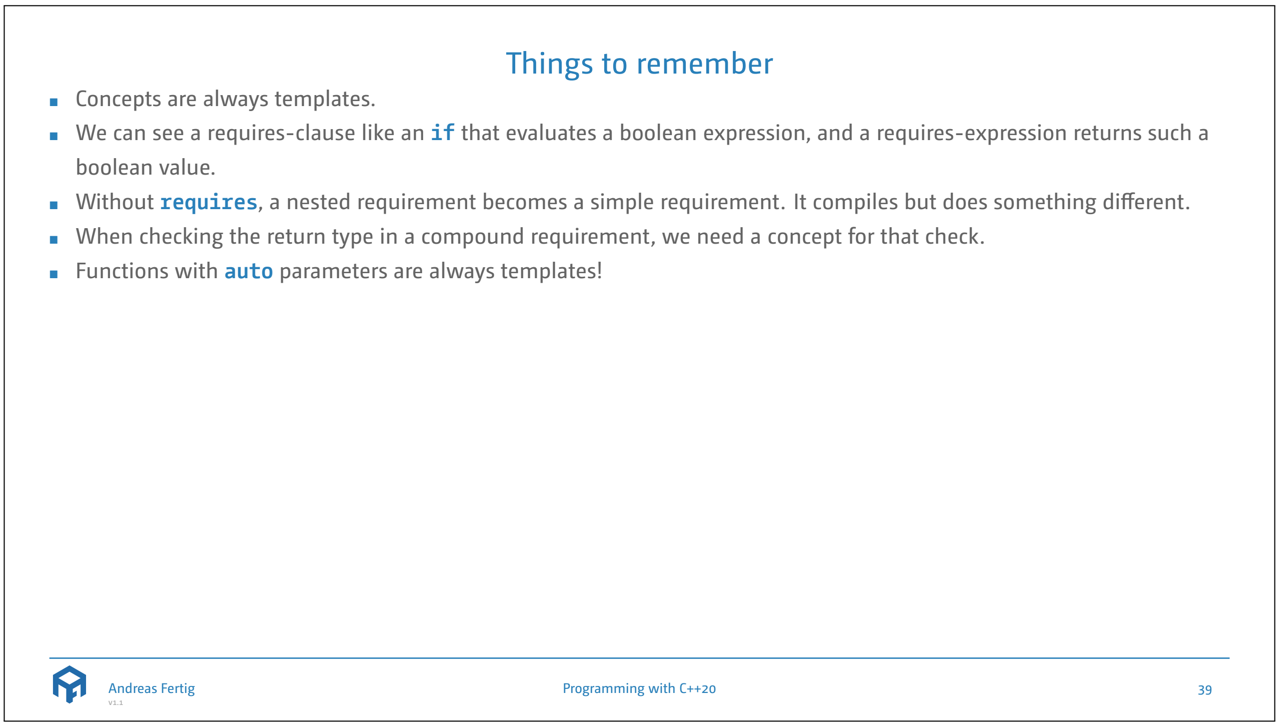
b) Write a (short) `make_unique` function. Use Concepts to constrain your `make_unique` implementation for a better error message.

- Solution: `solMakeUnique.cpp`

c) `exConceptConstraint.cpp` uses C++17 to implement two functions, `Fun`. One takes any class type with no `Release` function, and the other function takes a class type that also has a `Release` function. Use C++20 Concepts to simplify the code. Try to avoid negations.

- Solution: `solConceptConstraint.cpp`

Andreas Fertig v1.1 Programming with C++20 38



Things to remember

- Concepts are always templates.
- We can see a requires-clause like an **`if`** that evaluates a boolean expression, and a requires-expression returns such a boolean value.
- Without **`requires`**, a nested requirement becomes a simple requirement. It compiles but does something different.
- When checking the return type in a compound requirement, we need a concept for that check.
- Functions with **`auto`** parameters are always templates!

Andreas Fertig v1.1 Programming with C++20 39

12. Miscellaneous

Further Information

- Detect the standard of the compiler:
  - Pre-defined compiler-macros, like `__cplusplus = 201703L`, can be found in the standard at: [cpp.predefined].
  - Alternative: [5]
- A list of C++ standards and related drafts:
  - C++-03: N1638 (a little after 03 but one that is for free)
  - C++-11: N3337
  - C++-14: N4296
  - C++-17: N4640
- Code formatting helper:
  - clang-tidy [6]: Contains functionality like modernize.
  - clang-format [7]: Automatically convert the source code to a specific format. Helps with style guides.
- Conferences
  - Meeting C++, Germany, https://meetingcpp.com, https://www.youtube.com/user/MeetingCPP
  - CppCon, USA, https://cppcon.org, https://www.youtube.com/user/CppCon
  - emBO++, Germany, http://embo.io
  - ACCU, UK, https://conference.accu.org, https://www.youtube.com/channel/UCJhay24LTpO1s4bIZxuIqKw
  - ADC++, Germany, http://www.adcpp.de

Further Information

- code::dive, Poland, https://codedive.pl, https://www.youtube.com/channel/UCU0Rt8VHO5-YNQXwIjkf-1g
- Pod-/Screencast
  - C++ Weekly, https://www.youtube.com/playlist?list=PLs3KjaCtOwSZ2tbuV1hx8Xz-rFZTan2J1
  - CppCast, http://cppcast.com
- Books
  - A Tour of C++ [8]
  - Embracing Modern C++ Safely [9]
  - Beautiful C++ [10]
  - Effective Modern C++ [11]
  - C++ Templates: The Complete Guide [12]
  - C++17 in Detail [13]
- Blogs
  - https://fluentcpp.com
  - https://akrzemi1.wordpress.com

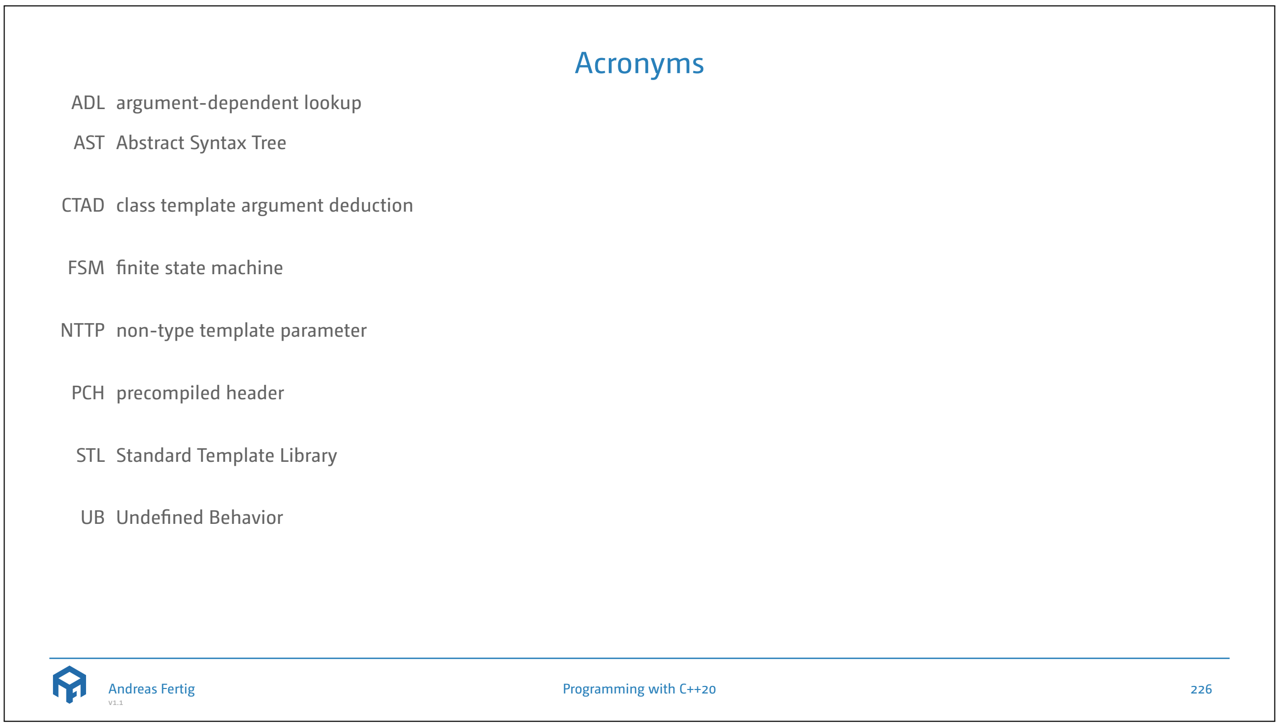
Used Compilers & Typography

Used Compilers

- Compilers used to compile (most of) the examples.
  - g++ 10.2.0
  - clang version 11.0.0 (https://github.com/llvm/llvm-project.git 176249bd6732a8044d457092ed932768724a6f06)

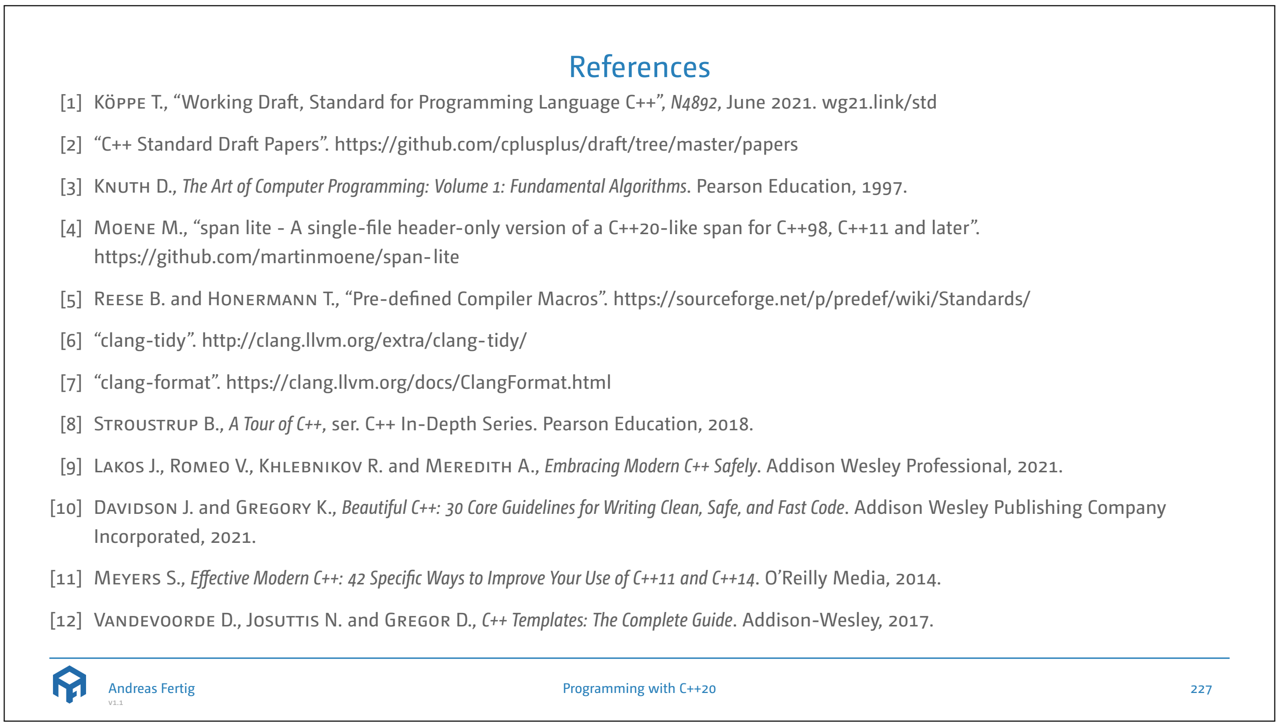
Typography

- 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/



Acronyms

ADL	argument-dependent lookup
AST	Abstract Syntax Tree
CTAD	class template argument deduction
FSM	finite state machine
NTTP	non-type template parameter
PCH	precompiled header
STL	Standard Template Library
UB	Undefined Behavior



References

[1] Köppe T., "Working Draft, Standard for Programming Language C++", *N4892*, June 2021. wg21.link/std

[2] "C++ Standard Draft Papers". https://github.com/cplusplus/draft/tree/master/papers

[3] Knuth D., *The Art of Computer Programming: Volume 1: Fundamental Algorithms*. Pearson Education, 1997.

[4] Moene M., "span lite - A single-file header-only version of a C++20-like span for C++98, C++11 and later". https://github.com/martinmoene/span-lite

[5] Reese B. and Honermann T., "Pre-defined Compiler Macros". https://sourceforge.net/p/predef/wiki/Standards/

[6] "clang-tidy". http://clang.llvm.org/extra/clang-tidy/

[7] "clang-format". https://clang.llvm.org/docs/ClangFormat.html

[8] Stroustrup B., *A Tour of C++*, ser. C++ In-Depth Series. Pearson Education, 2018.

[9] Lakos J., Romeo V., Khlebnikov R. and Meredith A., *Embracing Modern C++ Safely*. Addison Wesley Professional, 2021.

[10] Davidson J. and Gregory K., *Beautiful C++: 30 Core Guidelines for Writing Clean, Safe, and Fast Code*. Addison Wesley Publishing Company Incorporated, 2021.

[11] Meyers S., *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media, 2014.

[12] Vandevoorde D., Josuttis N. and Gregor D., *C++ Templates: The Complete Guide*. Addison-Wesley, 2017.

References

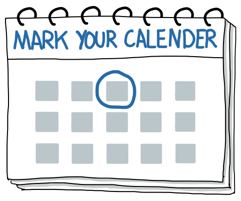
[13] FILIPEK B., *C++17 in Detail: Learn the Exciting Features of the New C++ Standard!* Amazon Digital Services LLC - KDP Print US, 2019.

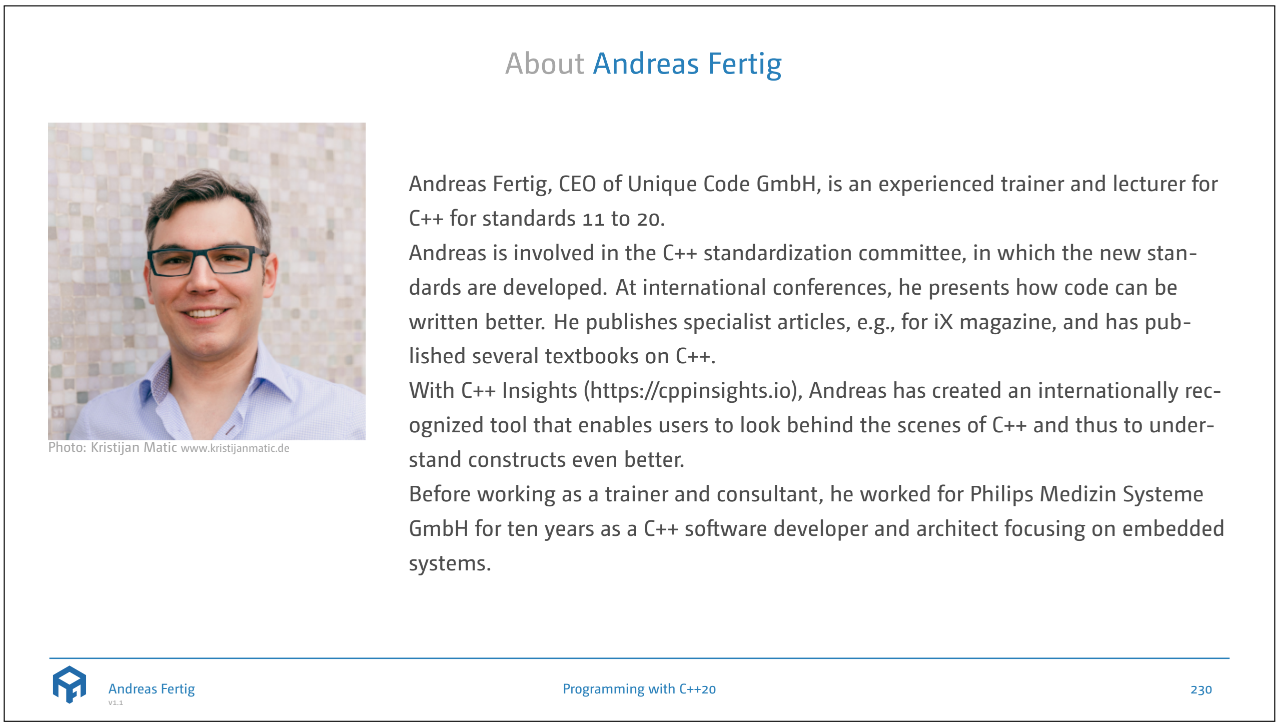
Images:

229: Franziska Panter

Upcoming Events

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.

Andreas Fertig v1.1 Programming with C++20 230