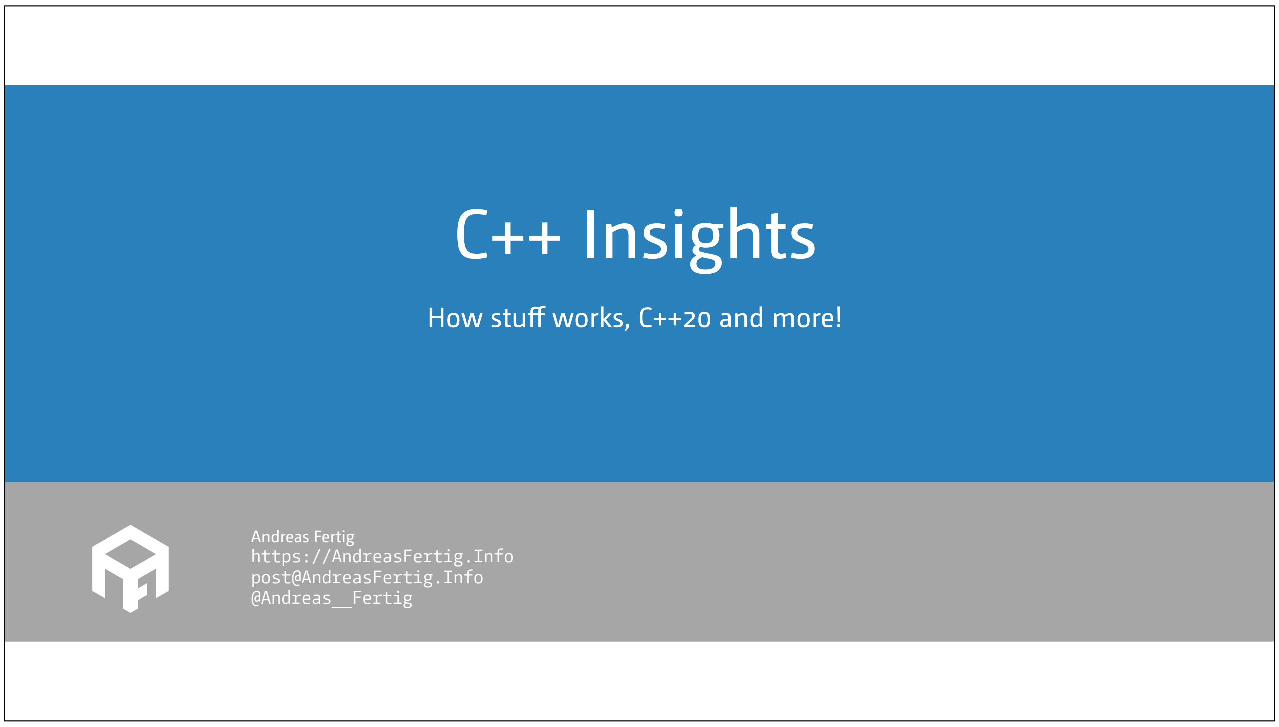
C++ Insights: How Stuff Works, Lambdas and More!

Andreas Fertig

C++ now
2021
MAY 2-7
Aspen, Colorado, USA



C++ Insights

How stuff works, C++20 and more!

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

fertig

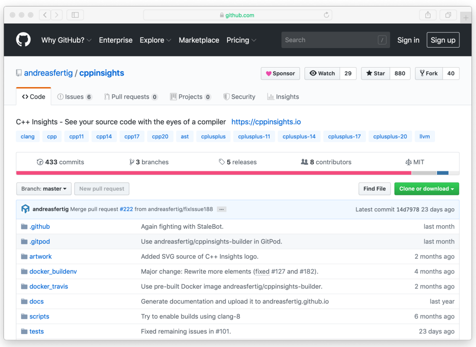
adjective /ˈfɛrtɪç/

finished
ready
complete
completed

C++ Insights

- Show what is going on.
- Make invisible things visible to assist in teaching.
- Create valid code.
- Create code that compiles.
- *Of course, it is open-source.*

https://github.com/andreasfertig/cppinsights/



andreasfertig / cppinsights

C++ Insights - See your source code with the eyes of a compiler https://cppinsights.io

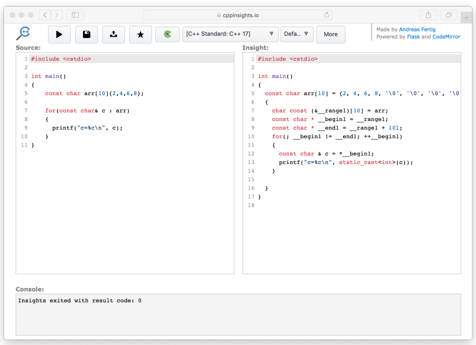
clang cpp cpp11 cpp14 cpp17 cpp20 ast cplusplus cplusplus-11 cplusplus-14 cplusplus-17 cplusplus-20 llvm

433 commits | 3 branches | 5 releases | 8 contributors | MIT

andreasfertig Merge pull request #222 from andreasfertig/fixIssue188 … Latest commit 14d7978 23 days ago

.github	Again fighting with StaleBot.	last month
.gitpod	Use andreasfertig/cppinsights-builder in GitPod.	last month
artwork	Added SVG source of C++ Insights logo.	2 months ago
docker_buildenv	Major change: Rewrite more elements (fixed #127 and #182).	4 months ago
docker_travis	Use pre-built Docker image andreasfertig/cppinsights-builder.	2 months ago
docs	Generate documentation and upload it to andreasfertig.github.io	last year
scripts	Try to enable builds using clang-8	6 months ago
tests	Fixed remaining issues in #101.	23 days ago

https://cppinsights.io



Source:

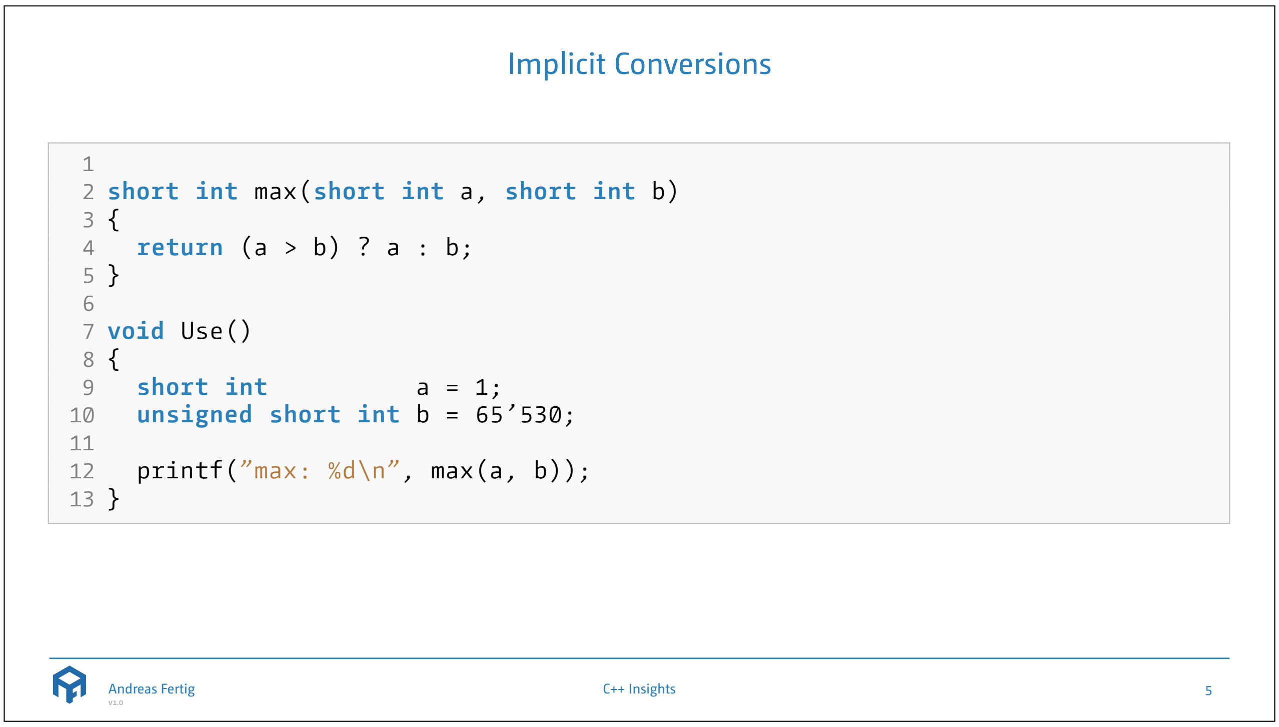
```
#include <cstdio>

int main()
{
    const char arr[10]{2,4,6,8};

    for(const char& c : arr)
    {
      printf("c=%c\n", c);
    }
}
```

Insight:

```
#include <cstdio>

int main()
{
  const char arr[10] = {2, 4, 6, 8, '\0', '\0', '\0', '\0
  {
    char const (&__range1)[10] = arr;
    const char * __begin1 = __range1;
    const char * __end1 = __range1 + 10l;
    for(; __begin1 != __end1; ++__begin1)
    {
      const char & c = *__begin1;
      printf("c=%c\n", static_cast<int>(c));
    }

  }
}
```

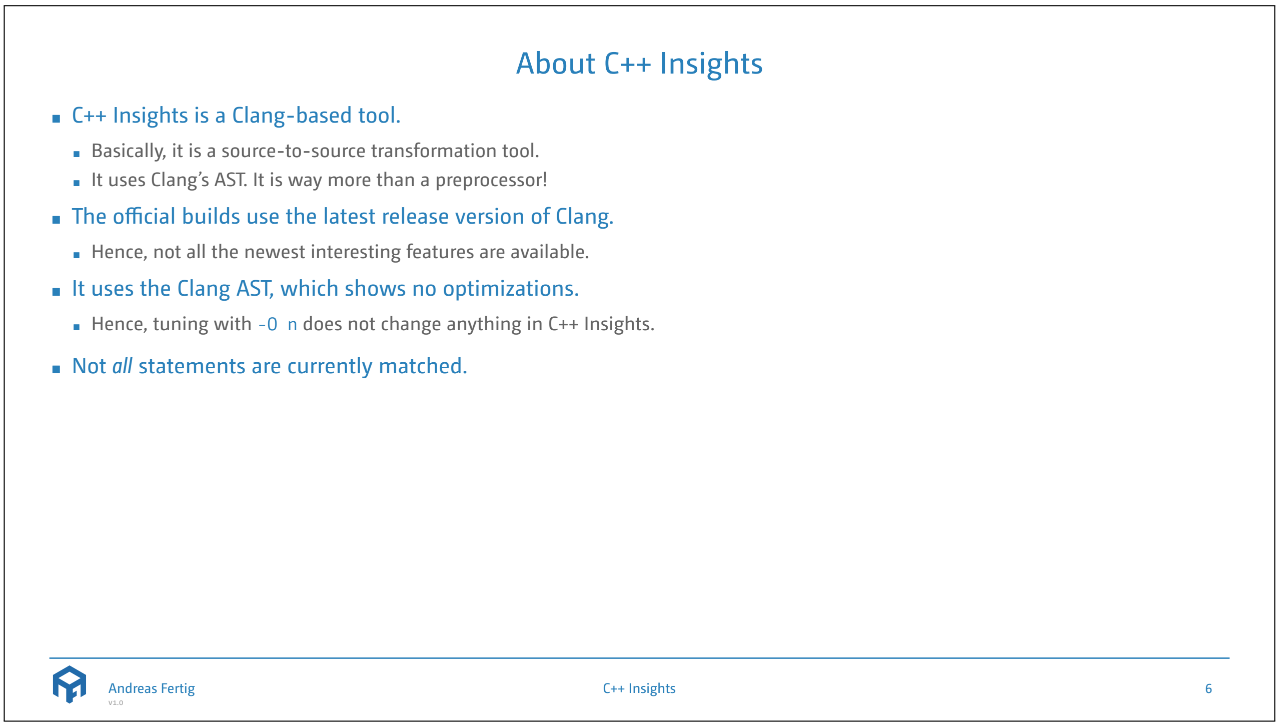
Console:

```
Insights exited with result code: 0
```



Implicit Conversions

```cpp

short int max(short int a, short int b)
{
  return (a > b) ? a : b;
}

void Use()
{
  short int          a = 1;
  unsigned short int b = 65’530;

  printf(”max: %d\n”, max(a, b));
}
```



About C++ Insights

- C++ Insights is a Clang-based tool.
  - Basically, it is a source-to-source transformation tool.
  - It uses Clang’s AST. It is way more than a preprocessor!
- The official builds use the latest release version of Clang.
  - Hence, not all the newest interesting features are available.
- It uses the Clang AST, which shows no optimizations.
  - Hence, tuning with `-O n` does not change anything in C++ Insights.
- Not *all* statements are currently matched.

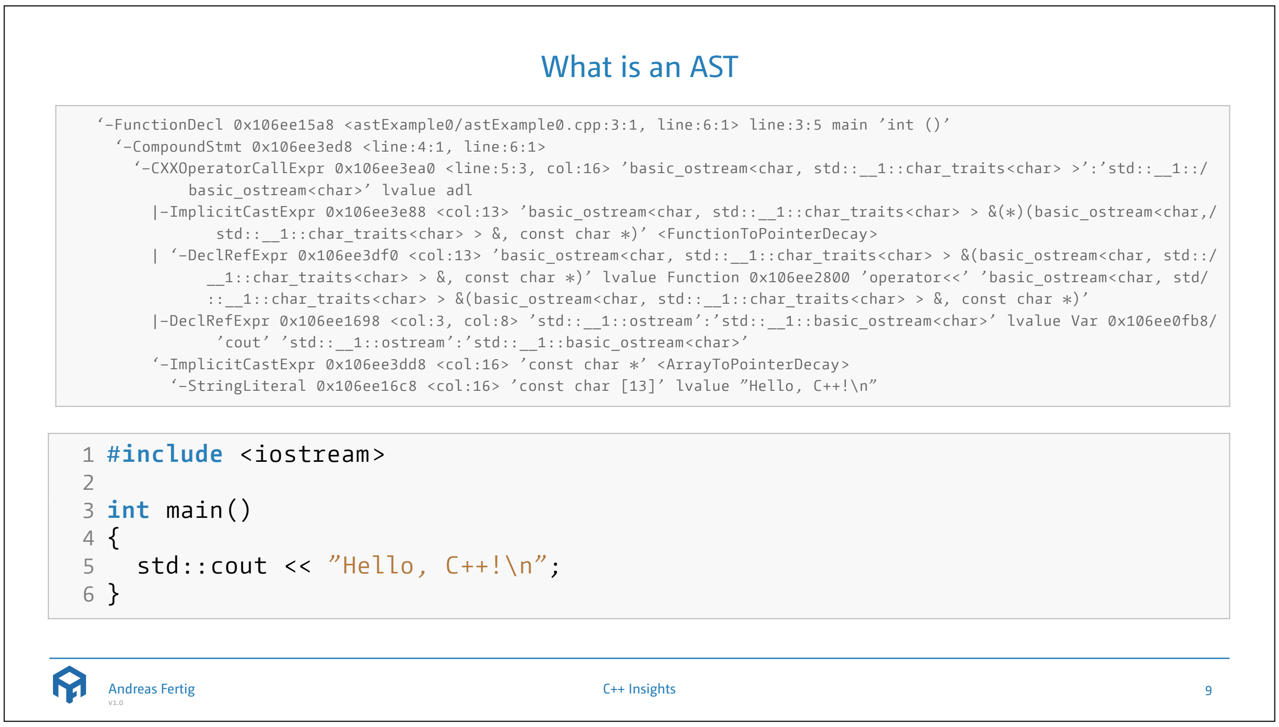
A word about limitations: Templates

- Creating code that compiles from templates is hard.
- To make it a bit easier for me, there is a `#ifdef INSIGHTS_USE_TEMPLATE` to have the code, but inactive.

```cpp
template<typename T>
void Func()
{}

class Demo
{
};

int main()
{
  Func<Demo>();
}
```

What is an AST

```
  ‘-FunctionDecl 0x106ee15a8 <astExample0/astExample0.cpp:3:1, line:6:1> line:3:5 main ’int ()’
    ‘-CompoundStmt 0x106ee3ed8 <line:4:1, line:6:1>
      ‘-CXXOperatorCallExpr 0x106ee3ea0 <line:5:3, col:16> ’basic_ostream<char, std::__1::char_traits<char> >’:’std::__1::/
            basic_ostream<char>’ lvalue adl
        |-ImplicitCastExpr 0x106ee3e88 <col:13> ’basic_ostream<char, std::__1::char_traits<char> > &(*)(basic_ostream<char,/
               std::__1::char_traits<char> > &, const char *)’ <FunctionToPointerDecay>
        | ‘-DeclRefExpr 0x106ee3df0 <col:13> ’basic_ostream<char, std::__1::char_traits<char> > &(basic_ostream<char, std::/
              __1::char_traits<char> > &, const char *)’ lvalue Function 0x106ee2800 ’operator<<’ ’basic_ostream<char, std/
              ::__1::char_traits<char> > &(basic_ostream<char, std::__1::char_traits<char> > &, const char *)’
        |-DeclRefExpr 0x106ee1698 <col:3, col:8> ’std::__1::ostream’:’std::__1::basic_ostream<char>’ lvalue Var 0x106ee0fb8/
               ’cout’ ’std::__1::ostream’:’std::__1::basic_ostream<char>’
        ‘-ImplicitCastExpr 0x106ee3dd8 <col:16> ’const char *’ <ArrayToPointerDecay>
          ‘-StringLiteral 0x106ee16c8 <col:16> ’const char [13]’ lvalue ”Hello, C++!\n”
```



What is an AST

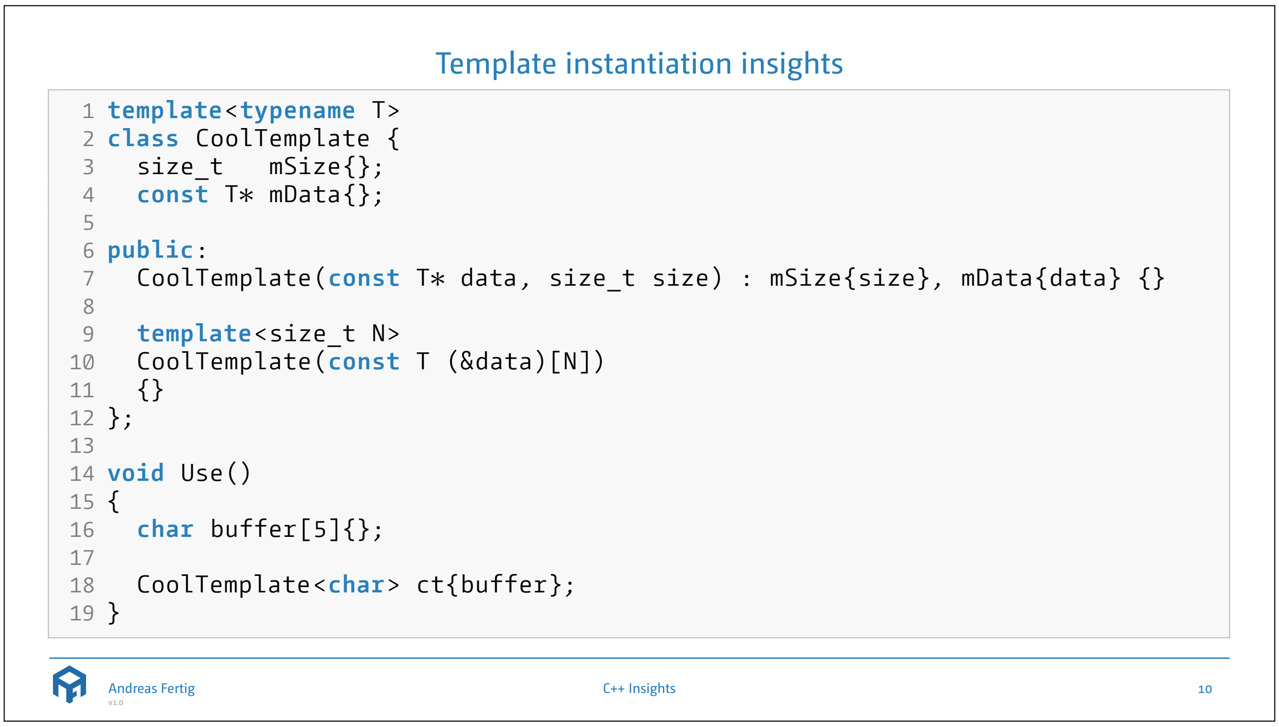
```
'-FunctionDecl 0x106ee15a8 <astExample0/astExample0.cpp:3:1, line:6:1> line:3:5 main 'int ()'
  '-CompoundStmt 0x106ee3ed8 <line:4:1, line:6:1>
    '-CXXOperatorCallExpr 0x106ee3ea0 <line:5:3, col:16> 'basic_ostream<char, std::__1::char_traits<char> >':'std::__1::/
          basic_ostream<char>' lvalue adl
      |-ImplicitCastExpr 0x106ee3e88 <col:13> 'basic_ostream<char, std::__1::char_traits<char> > &(*)(basic_ostream<char,/
             std::__1::char_traits<char> > &, const char *)' <FunctionToPointerDecay>
      | '-DeclRefExpr 0x106ee3df0 <col:13> 'basic_ostream<char, std::__1::char_traits<char> > &(basic_ostream<char, std::/
            __1::char_traits<char> > &, const char *)' lvalue Function 0x106ee2800 'operator<<' 'basic_ostream<char, std/
            ::__1::char_traits<char> > &(basic_ostream<char, std::__1::char_traits<char> > &, const char *)'
      |-DeclRefExpr 0x106ee1698 <col:3, col:8> 'std::__1::ostream':'std::__1::basic_ostream<char>' lvalue Var 0x106ee0fb8/
             'cout' 'std::__1::ostream':'std::__1::basic_ostream<char>'
      '-ImplicitCastExpr 0x106ee3dd8 <col:16> 'const char *' <ArrayToPointerDecay>
        '-StringLiteral 0x106ee16c8 <col:16> 'const char [13]' lvalue "Hello, C++!\n"
```

```cpp
#include <iostream>

int main()
{
  std::cout << "Hello, C++!\n";
}
```

Andreas Fertig
v1.0

C++ Insights

9



Template instantiation insights

```cpp
template<typename T>
class CoolTemplate {
  size_t   mSize{};
  const T* mData{};

public:
  CoolTemplate(const T* data, size_t size) : mSize{size}, mData{data} {}

  template<size_t N>
  CoolTemplate(const T (&data)[N])
  {}
};

void Use()
{
  char buffer[5]{};

  CoolTemplate<char> ct{buffer};
}
```

Andreas Fertig
v1.0

C++ Insights

10

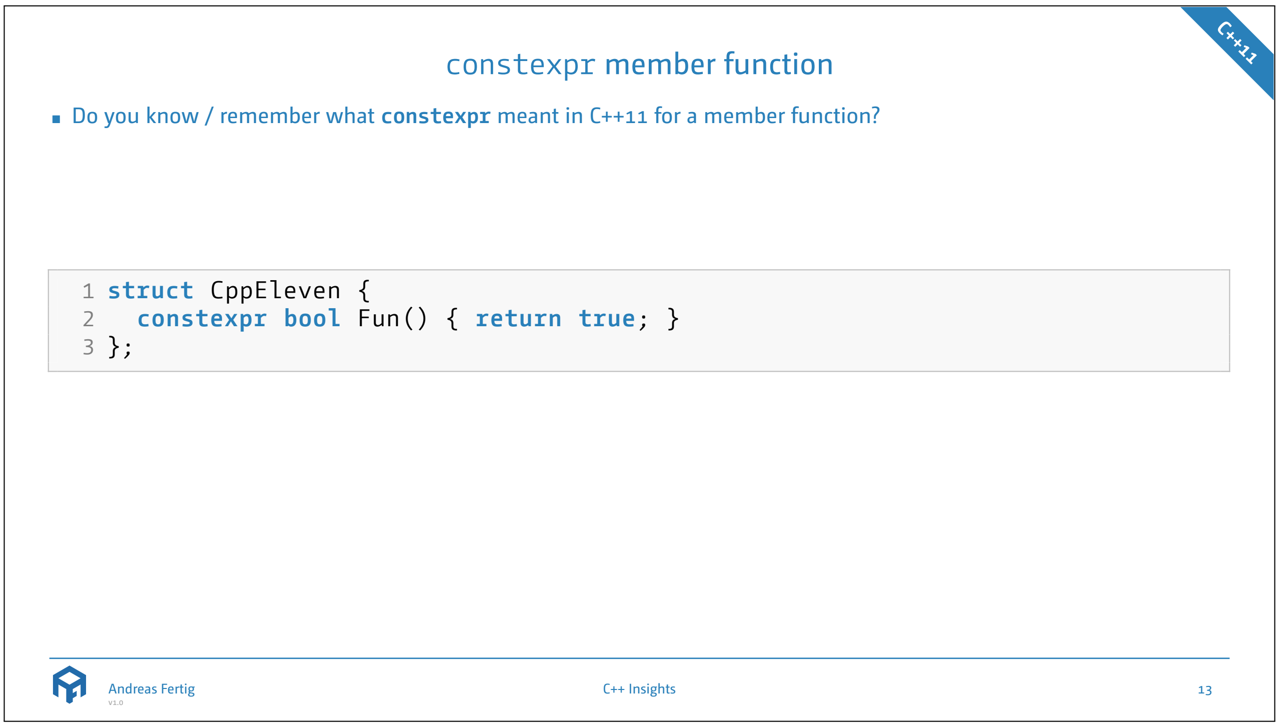
Default Parameter

- How does a default parameter take effect?

```cpp
void Func(int x = 23) {}

int main()
{
  Func();
}
```

Default Member Initializer

```cpp
class Init {
public:
  Init()
  : i{9}
  {}

  int              i{0};
  std::vector<int> v{2, 3, 4};
  std::string      s{"Hello"};
};
```

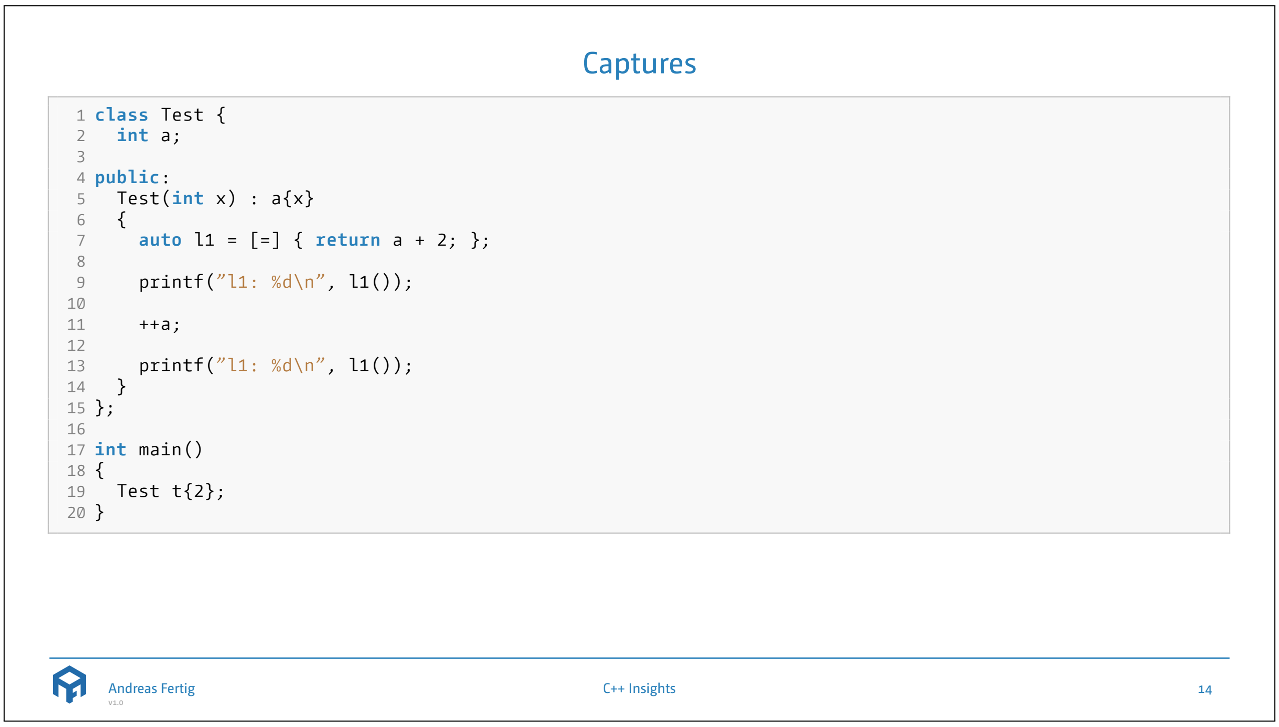


C++11

constexpr member function

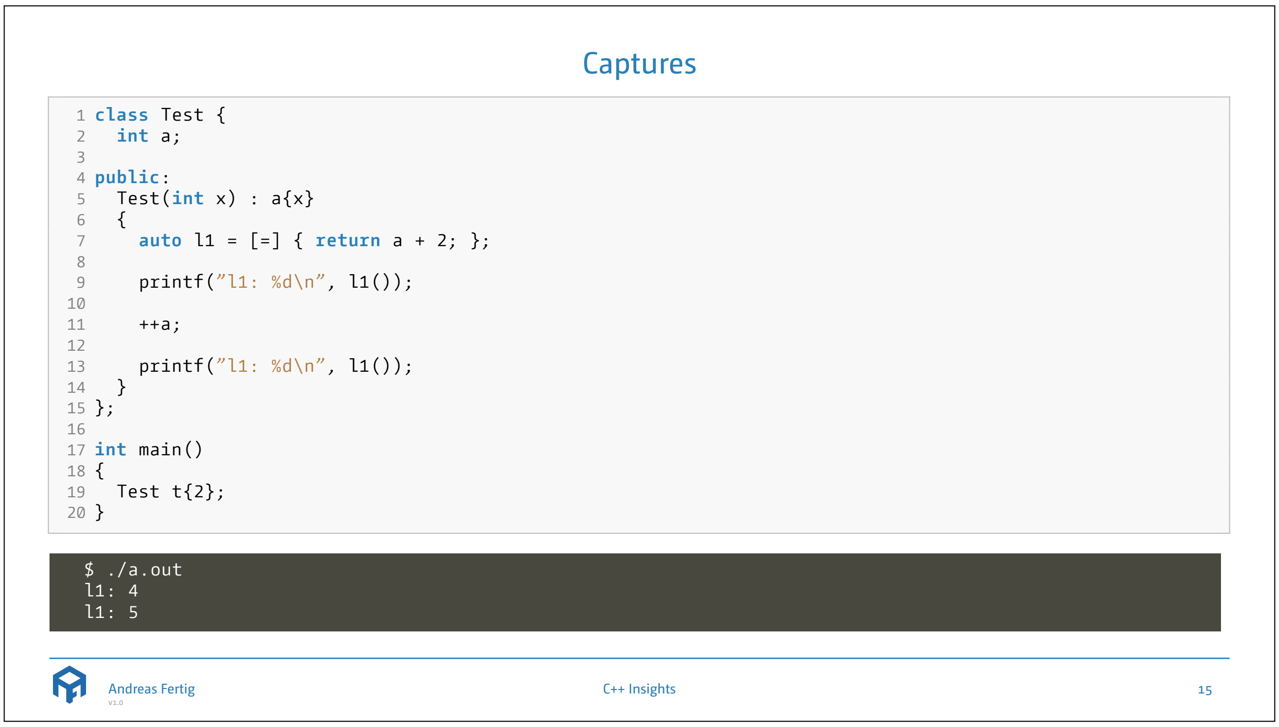
- Do you know / remember what **constexpr** meant in C++11 for a member function?

```
struct CppEleven {
  constexpr bool Fun() { return true; }
};
```



Captures

```
class Test {
  int a;

public:
  Test(int x) : a{x}
  {
    auto l1 = [=] { return a + 2; };

    printf("l1: %d\n", l1());

    ++a;

    printf("l1: %d\n", l1());
  }
};

int main()
{
  Test t{2};
}
```

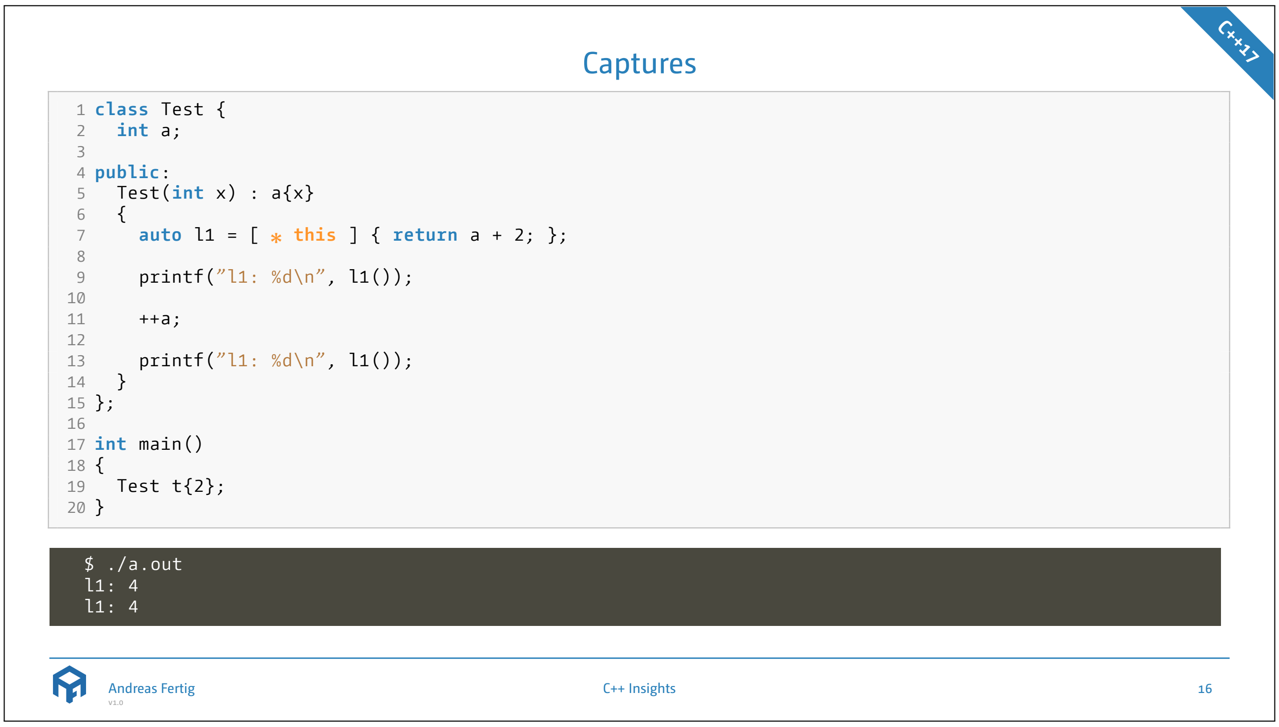


Captures

```cpp
class Test {
  int a;

public:
  Test(int x) : a{x}
  {
    auto l1 = [=] { return a + 2; };

    printf(”l1: %d\n”, l1());

    ++a;

    printf(”l1: %d\n”, l1());
  }
};

int main()
{
  Test t{2};
}
```

```
$ ./a.out
l1: 4
l1: 5
```

Andreas Fertig v1.0 C++ Insights 15



C++17

Captures

```cpp
class Test {
  int a;

public:
  Test(int x) : a{x}
  {
    auto l1 = [ * this ] { return a + 2; };

    printf(”l1: %d\n”, l1());

    ++a;

    printf(”l1: %d\n”, l1());
  }
};

int main()
{
  Test t{2};
}
```

```
$ ./a.out
l1: 4
l1: 4
```

Andreas Fertig v1.0 C++ Insights 16

C++17

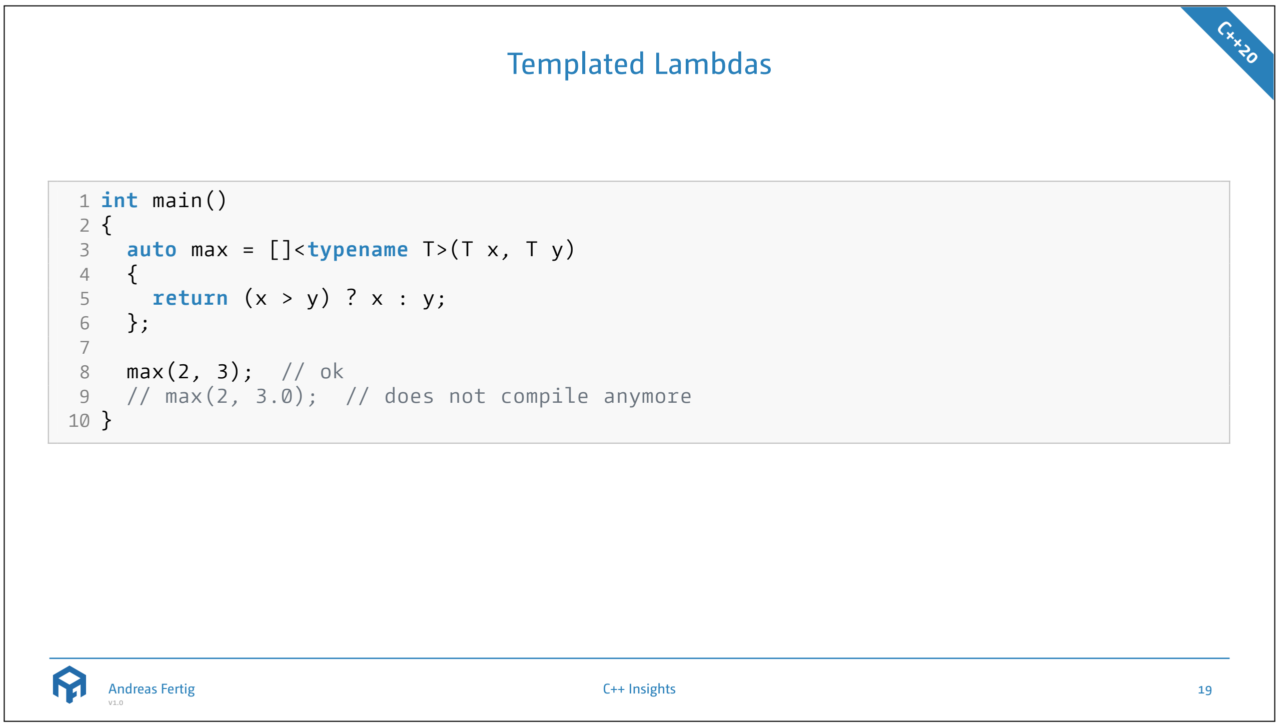
Captures

```cpp
class Test {
  int a;
  int b{};

public:
  Test(int x) : a{x}
  {
    auto l2 = [*this] { return a + 2; };
  }
};
```

C++20

Templated Lambdas

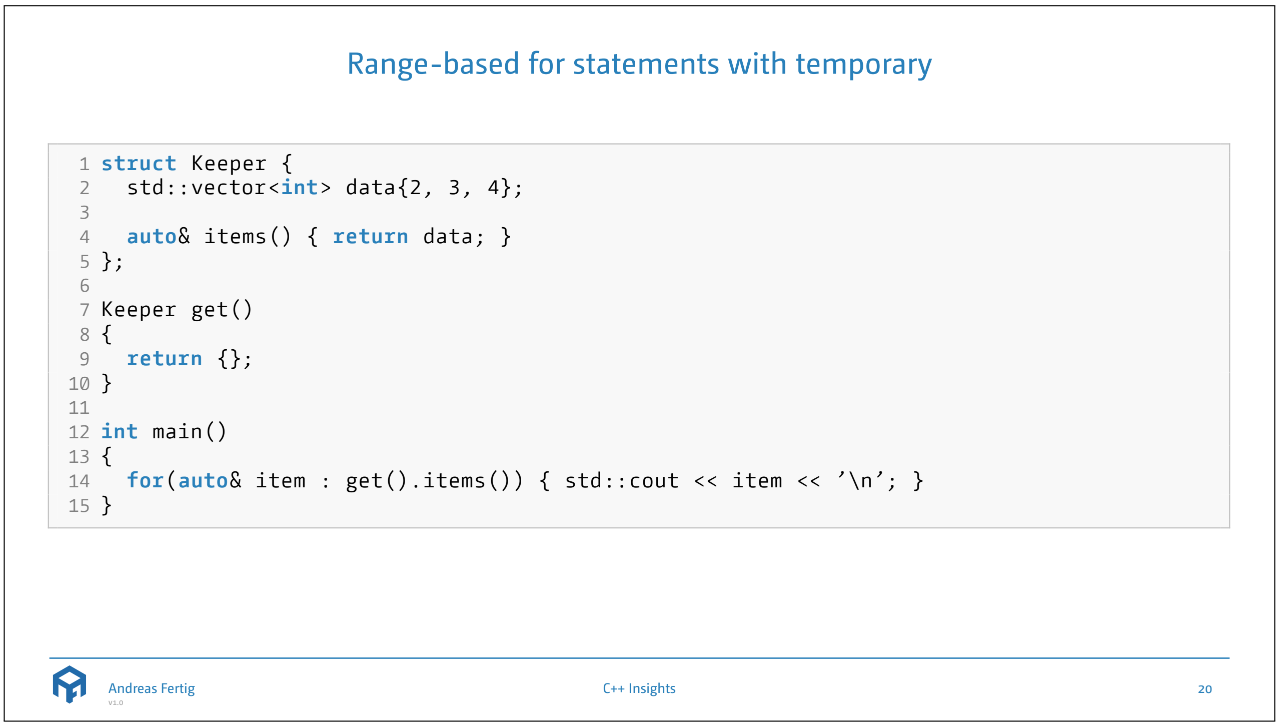
```cpp
int main()
{
  auto max = [](auto x, auto y) {
    return (x > y) ? x : y;
  };

  max(2, 3);    // ok
  max(2, 3.0);  // not wanted
}
```



C++20

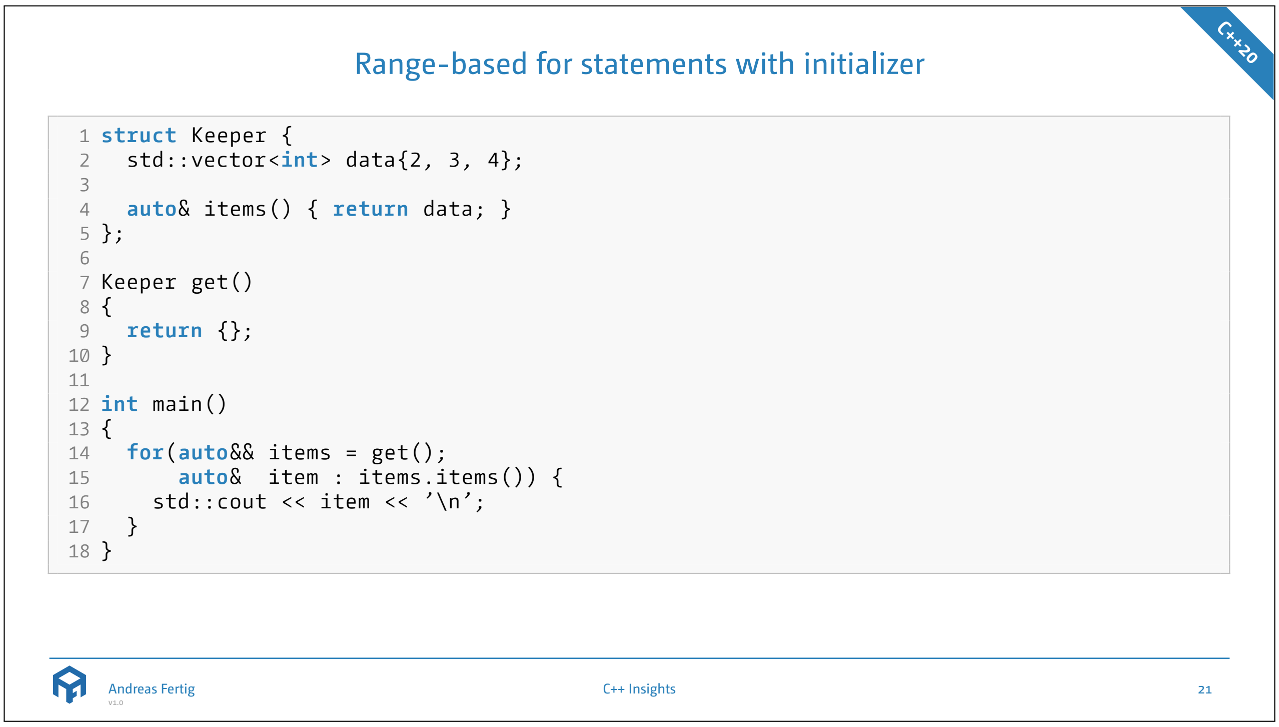
Templated Lambdas

```cpp
int main()
{
  auto max = []<typename T>(T x, T y)
  {
    return (x > y) ? x : y;
  };

  max(2, 3);  // ok
  // max(2, 3.0);  // does not compile anymore
}
```



Range-based for statements with temporary

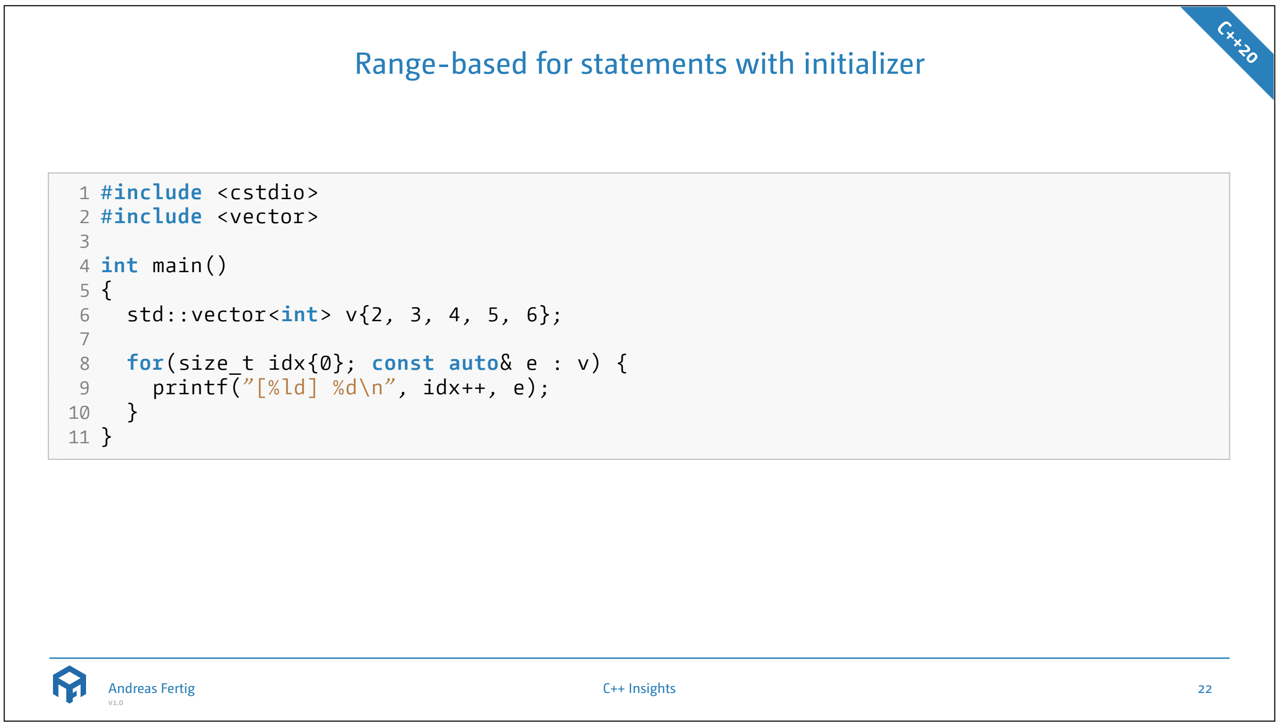
```cpp
struct Keeper {
  std::vector<int> data{2, 3, 4};

  auto& items() { return data; }
};

Keeper get()
{
  return {};
}

int main()
{
  for(auto& item : get().items()) { std::cout << item << '\n'; }
}
```



C++20

Range-based for statements with initializer

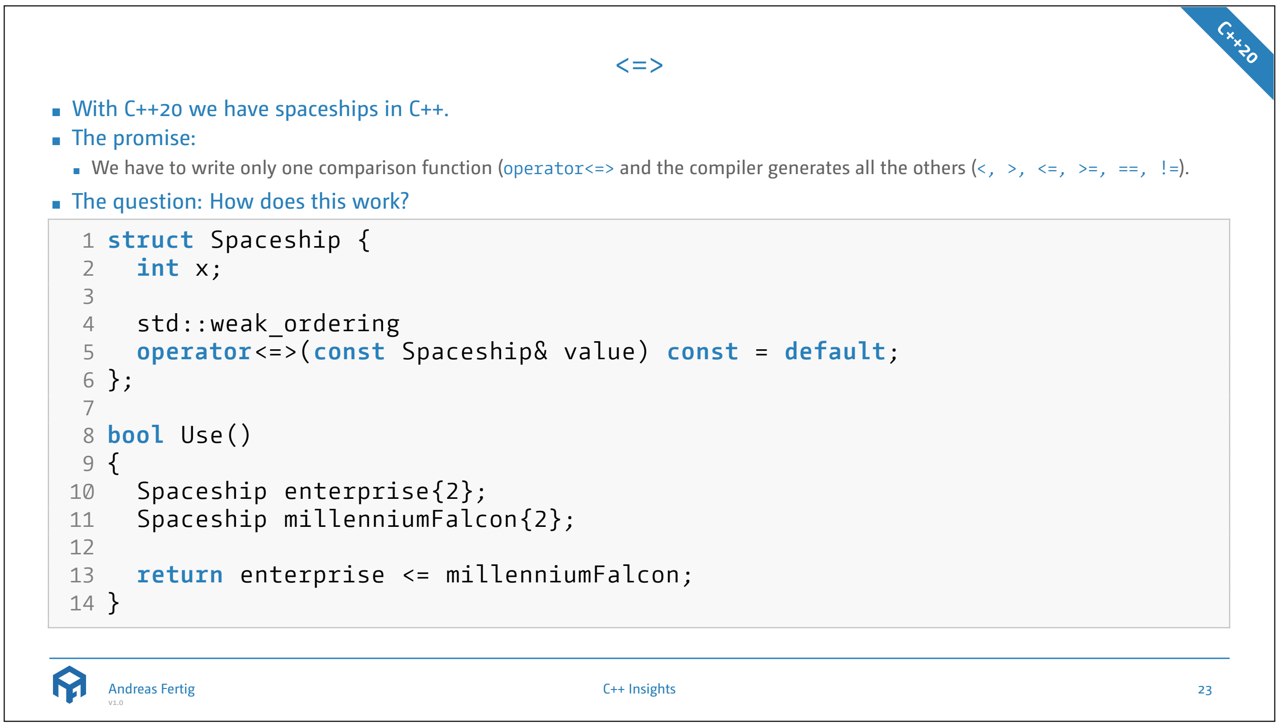
```cpp
struct Keeper {
  std::vector<int> data{2, 3, 4};

  auto& items() { return data; }
};

Keeper get()
{
  return {};
}

int main()
{
  for(auto&& items = get();
      auto&  item : items.items()) {
    std::cout << item << '\n';
  }
}
```



C++20

Range-based for statements with initializer

```cpp
#include <cstdio>
#include <vector>

int main()
{
  std::vector<int> v{2, 3, 4, 5, 6};

  for(size_t idx{0}; const auto& e : v) {
    printf("[%ld] %d\n", idx++, e);
  }
}
```



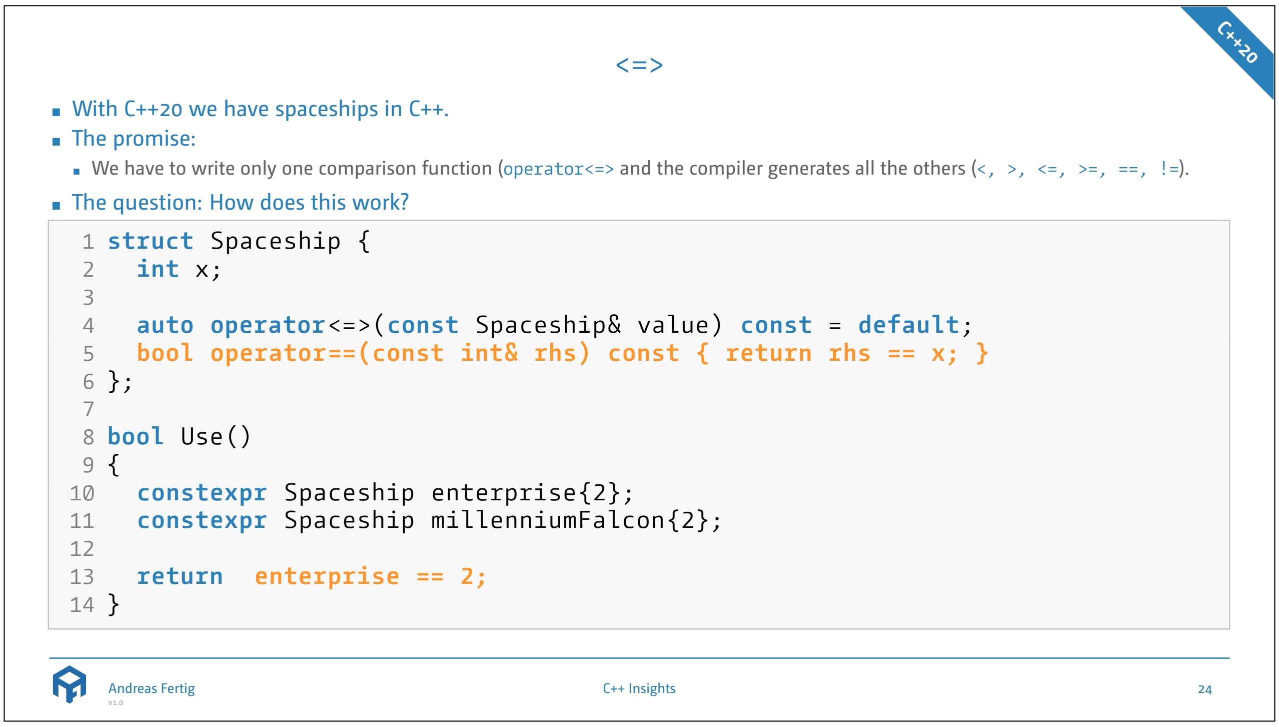
C++20

<=>

- With C++20 we have spaceships in C++.
- The promise:
  - We have to write only one comparison function (`operator<=>` and the compiler generates all the others (`<, >, <=, >=, ==, !=`).
- The question: How does this work?

```cpp
struct Spaceship {
  int x;

  std::weak_ordering
  operator<=>(const Spaceship& value) const = default;
};

bool Use()
{
  Spaceship enterprise{2};
  Spaceship millenniumFalcon{2};

  return enterprise <= millenniumFalcon;
}
```

Andreas Fertig
v1.0
C++ Insights
23



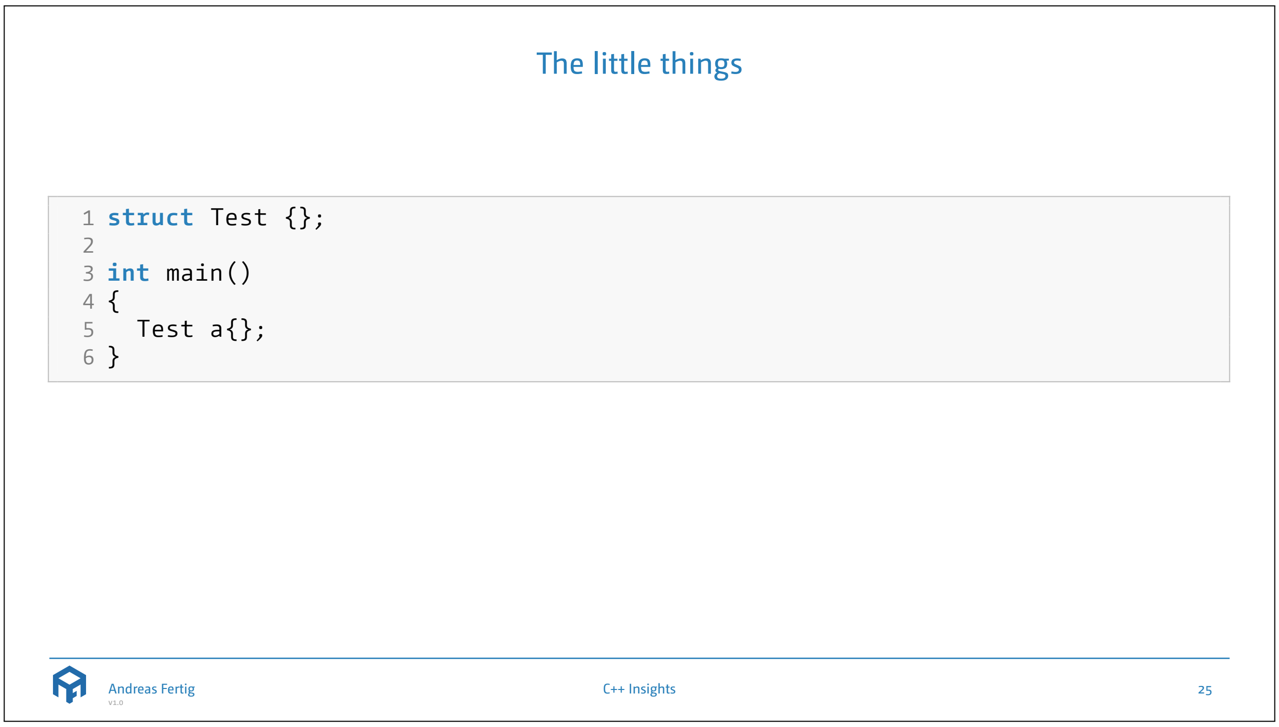
C++20

<=>

- With C++20 we have spaceships in C++.
- The promise:
  - We have to write only one comparison function (`operator<=>` and the compiler generates all the others (`<, >, <=, >=, ==, !=`).
- The question: How does this work?

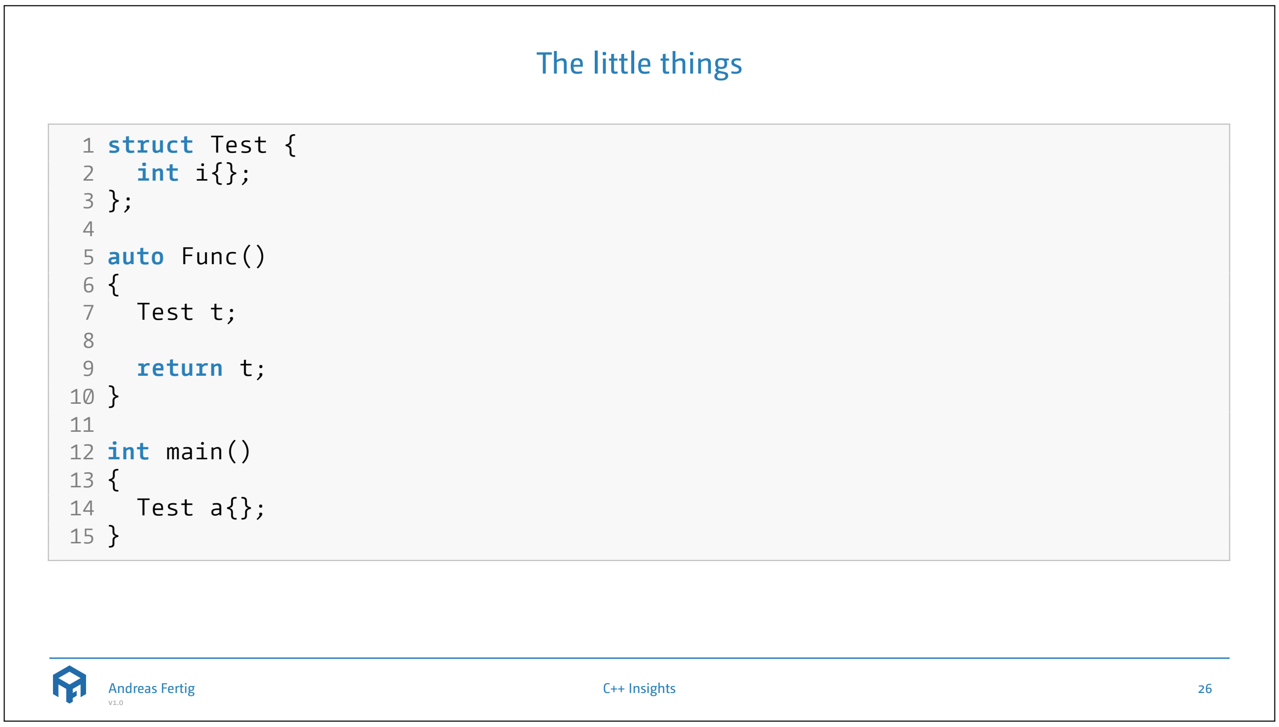
```cpp
struct Spaceship {
  int x;

  auto operator<=>(const Spaceship& value) const = default;
  bool operator==(const int& rhs) const { return rhs == x; }
};

bool Use()
{
  constexpr Spaceship enterprise{2};
  constexpr Spaceship millenniumFalcon{2};

  return  enterprise == 2;
}
```

Andreas Fertig
v1.0
C++ Insights
24



The little things

```cpp
struct Test {};

int main()
{
  Test a{};
}
```



The little things

```cpp
struct Test {
  int i{};
};

auto Func()
{
  Test t;

  return t;
}

int main()
{
  Test a{};
}
```

What can C++ Insights do for you?

- I don't know ;-)
- The following is my experience with it:
  - Seeing is a very valuable thing. Even if you know something in general, C++ Insights may put your attention on it.
  - Classes I taught using C++ Insights (as well as Matt Godbolt's Compiler Explorer) tend to be more interactive. Attendees start asking broader questions about certain constructs.
  - C++ Insights can help to settle two different opinions by visualizing what the compiler (at least Clang) does.
  - Like Integrated Development Environments (IDEs), C++ Insights visualizes template instantiations. Seeing them often helps, but seeing the absence of a specific instantiation may lead you to the issue you're looking for.

Support the project

https://github.com/andreasfertig/cppinsights

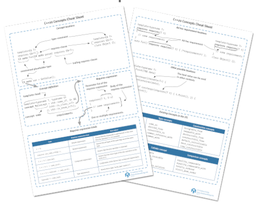
https://www.patreon.com/cppinsights

https://shop.spreadshirt.de/cppinsights

}

I am Fertig.

C++20 Concepts Cheat Sheet



fertig.to/subscribe

Used Compilers & Typography

Used Compilers

- Compilers used to compile (most of) the examples.
  - g++ (GCC) 10.1.0
  - clang version 10.0.0 (https://github.com/llvm/llvm-project.git d32170dbd5b0d54436537b6b75beaf44324e0c28)

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:
32: Franziska Panter

Upcoming Events

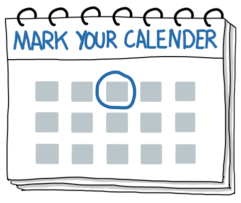
Talks

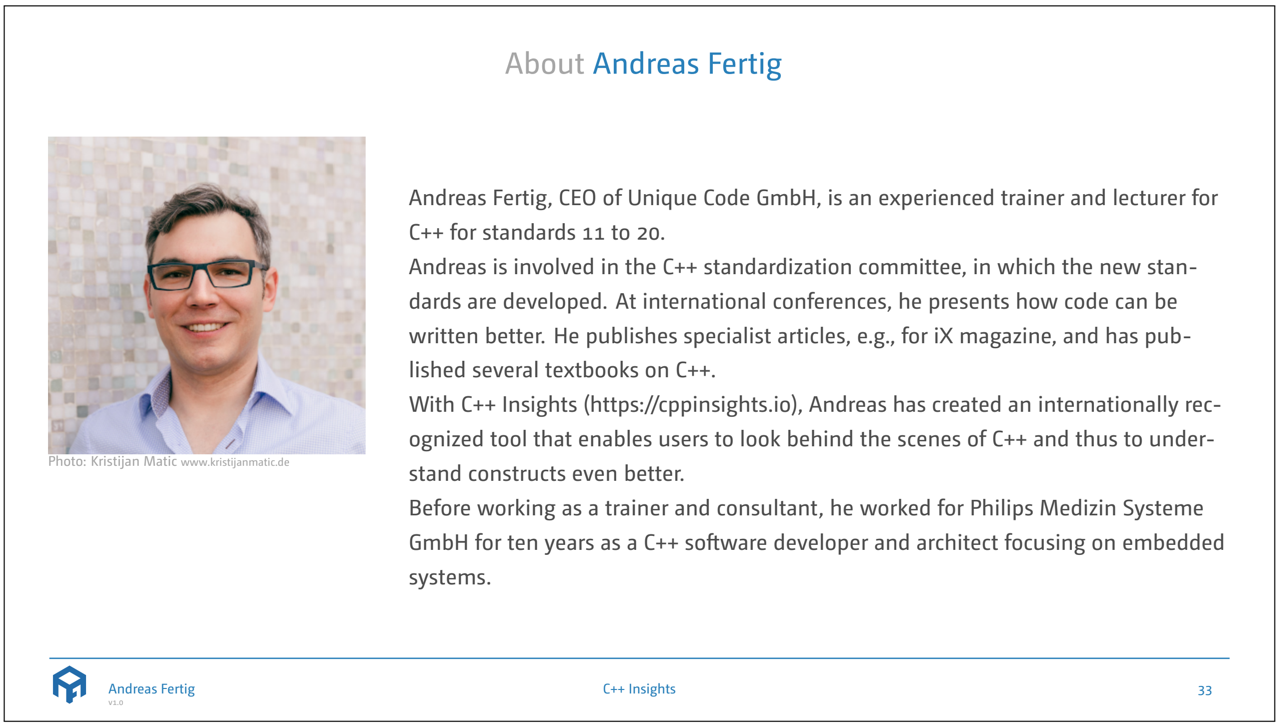
- *C++: • Demystified*, ADC++, May 18

Training Classes

- *C++1x für eingebettete Systeme*, ADC++, May 17
- *C++ Clean Code – Best Practices für Programmierer*, golem Akademie, June 07 - 11
- *C++ Clean Code – Best Practices für Programmierer*, golem Akademie, September 13 - 17
- *Programmieren mit C++20*, Andreas Fertig, September 27 - 29
- *C++1x für eingebettete Systeme*, QA Systems, October 14 - 15

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 v1.0 C++ Insights 33