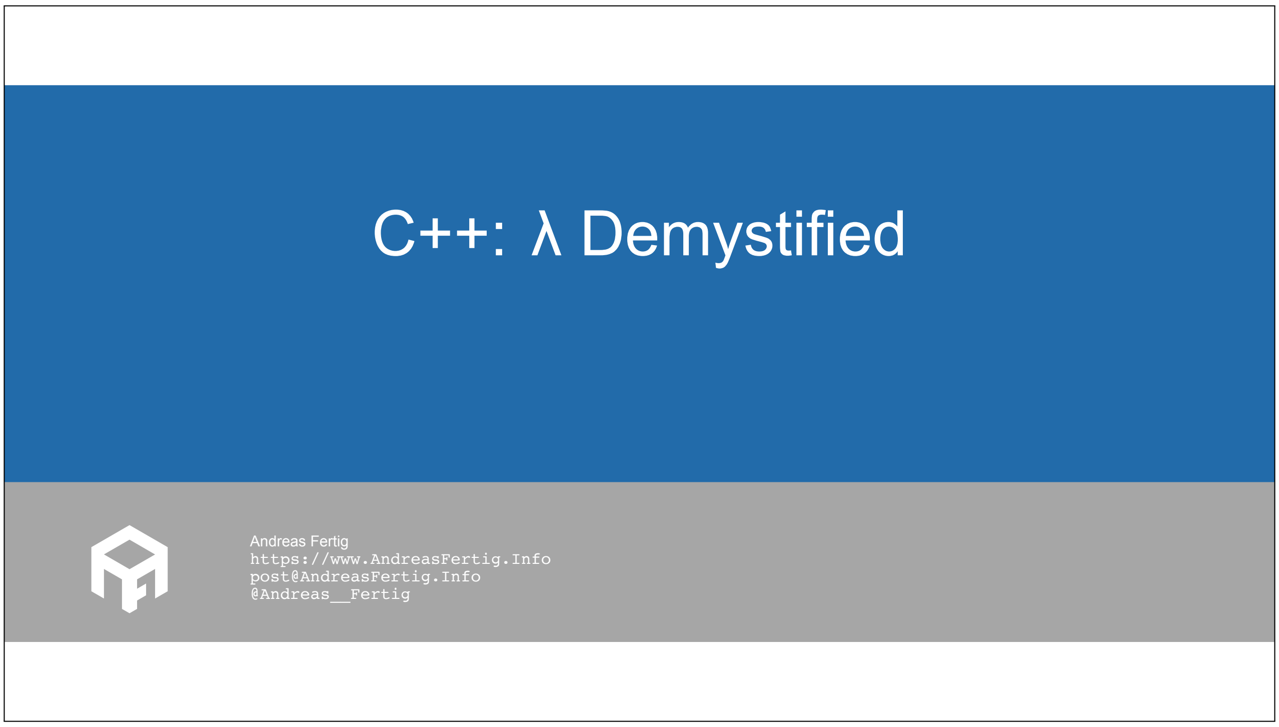


C++: λ Demystified

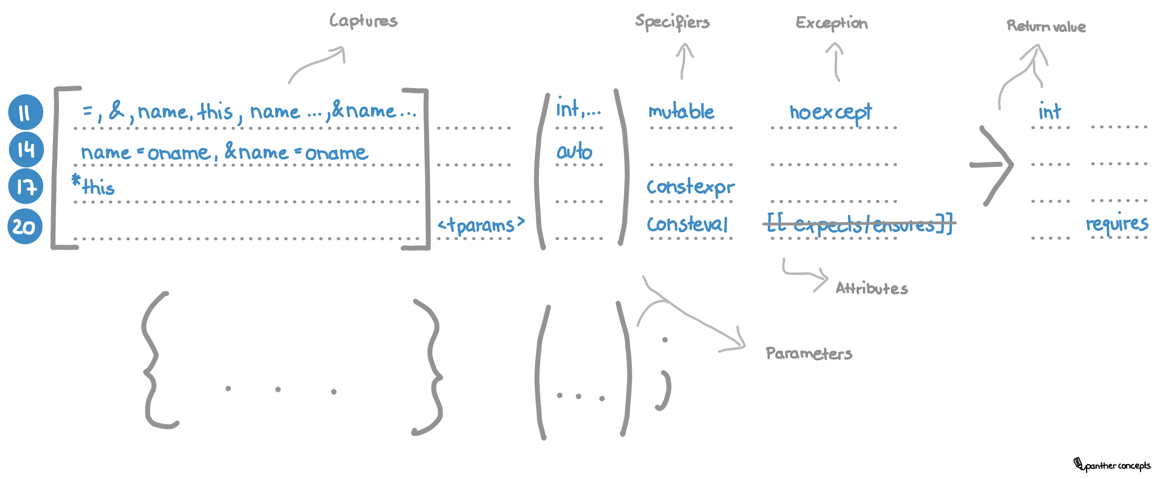
Andreas Fertig
https://www.AndreasFertig.Info
post@AndreasFertig.Info
@Andreas__Fertig

fertig

adjective /ˈfɛrtɪç/

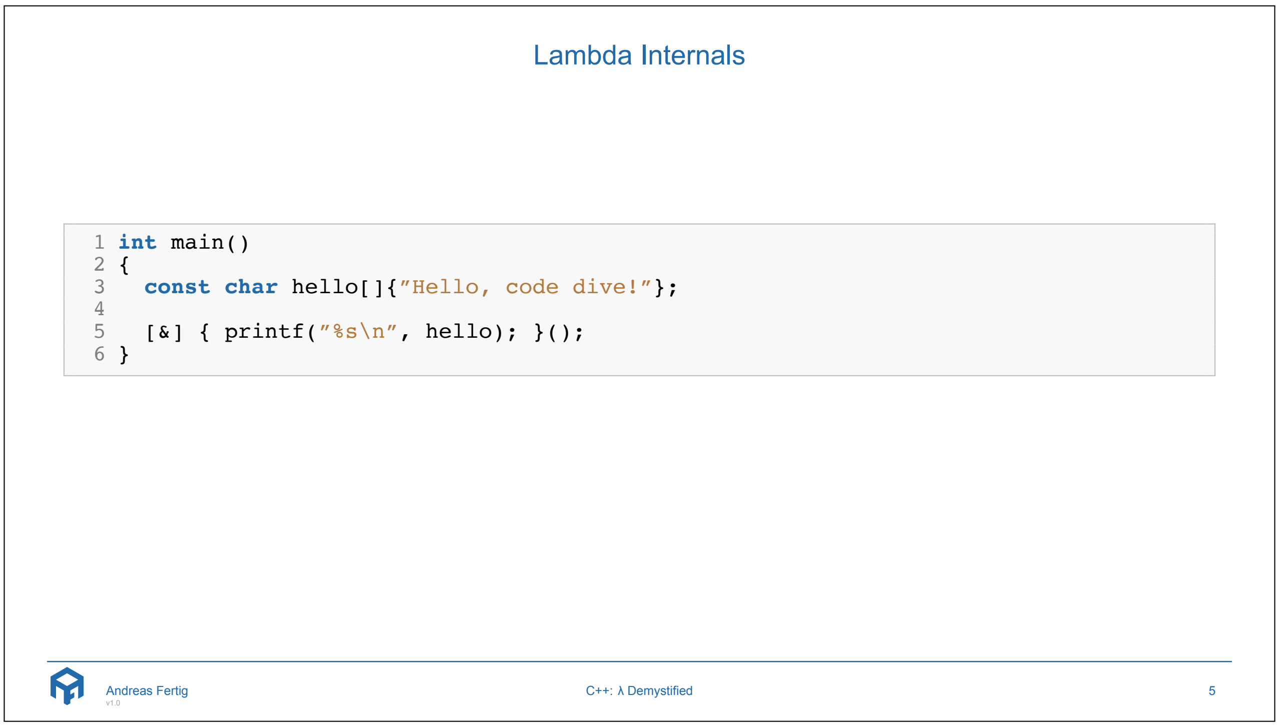
finished
ready
complete
completed

Lambda Evolution



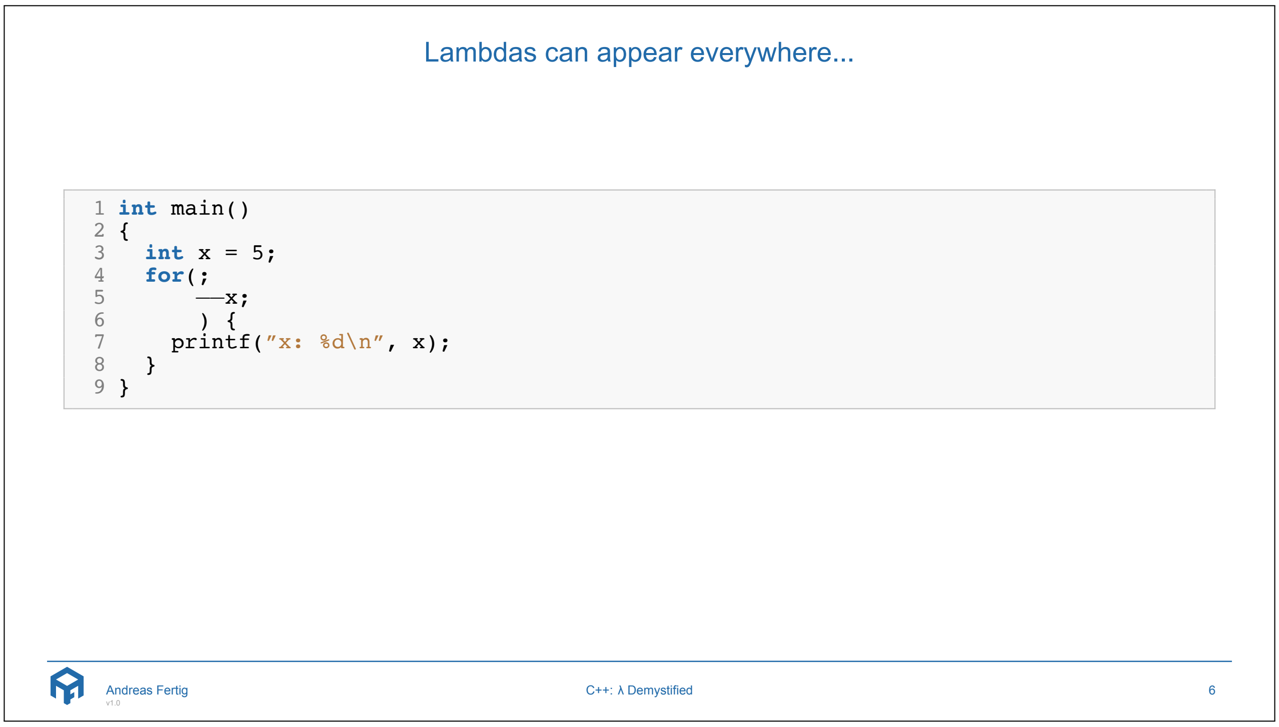
Valid or Not?

```cpp
int main()
{
  (+[]() {})();
}
```



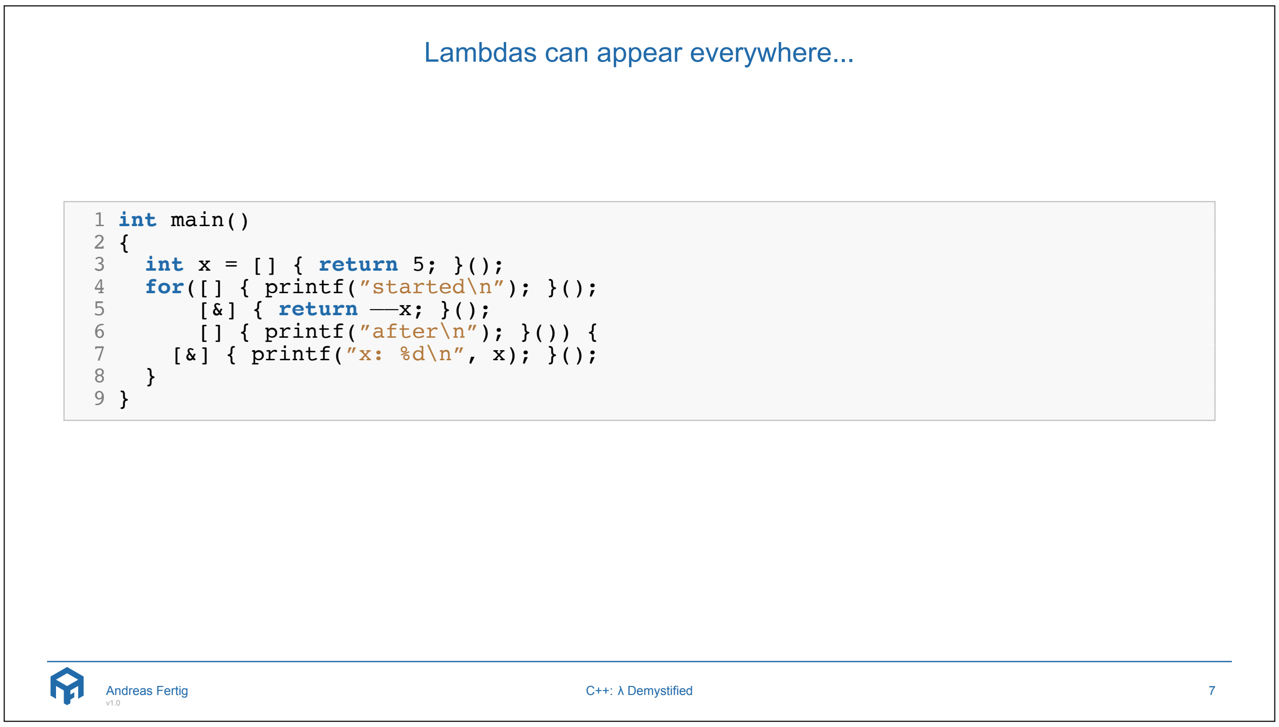
Lambda Internals

```cpp
int main()
{
  const char hello[]{"Hello, code dive!"};

  [&] { printf("%s\n", hello); }();
}
```



Lambdas can appear everywhere...

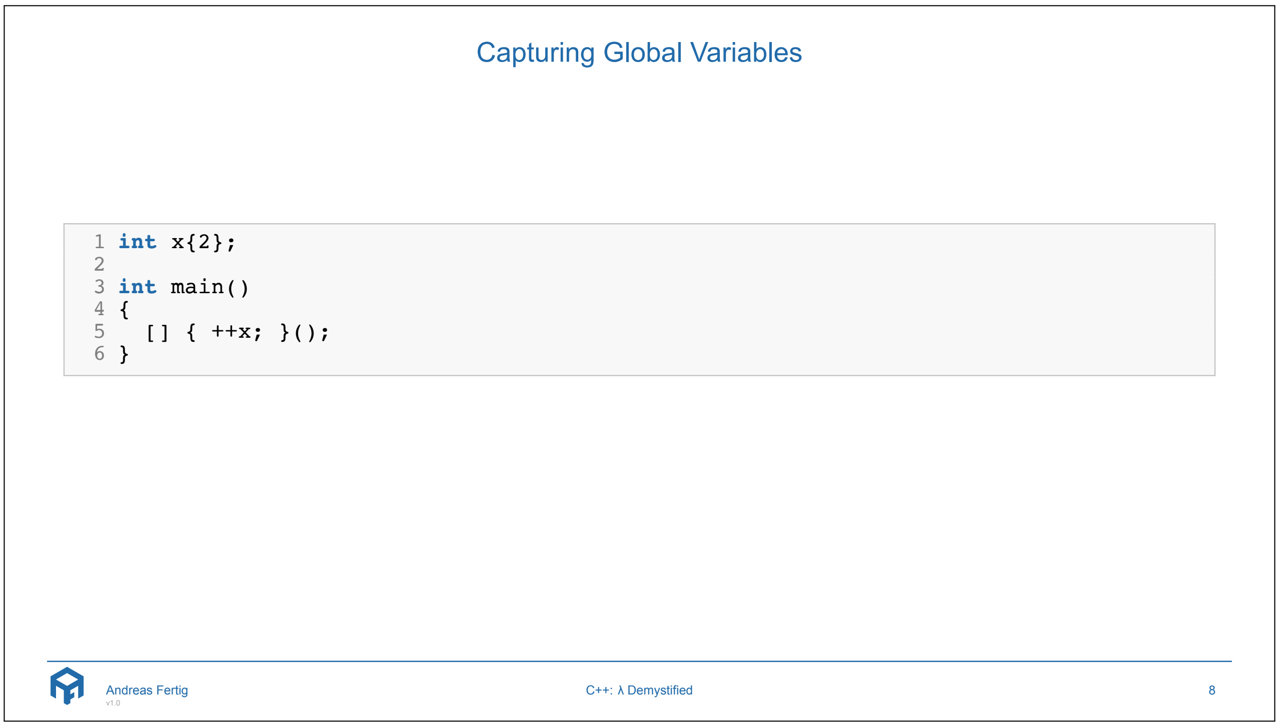
```cpp
int main()
{
  int x = 5;
  for(;
      --x;
      ) {
    printf("x: %d\n", x);
  }
}
```



Lambdas can appear everywhere...

```cpp
int main()
{
  int x = [] { return 5; }();
  for([] { printf("started\n"); }();
      [&] { return --x; }();
      [] { printf("after\n"); }()) {
    [&] { printf("x: %d\n", x); }();
  }
}
```

Andreas Fertig v1.0 C++: λ Demystified 7



Capturing Global Variables

```cpp
int x{2};

int main()
{
  [] { ++x; }();
}
```

Andreas Fertig v1.0 C++: λ Demystified 8

Captureless Lambda and Function Pointer

```cpp
int (*fp)(int, char) = [](int a, char b) { return a + b; };
```

Size of a Lambda

```cpp
int main()
{
  char a{2};
  int  b{3};
  char c{2};

  auto f = [=] {
    a;
    b;
    c;
  };
}
```

Assume a x64 Platform.

Size of a Lambda

```
int main()
{
  char a{2};
  int  b{3};
  char c{2};

  auto s = [=] {
    a;
    c;
    b;
  };
}
```

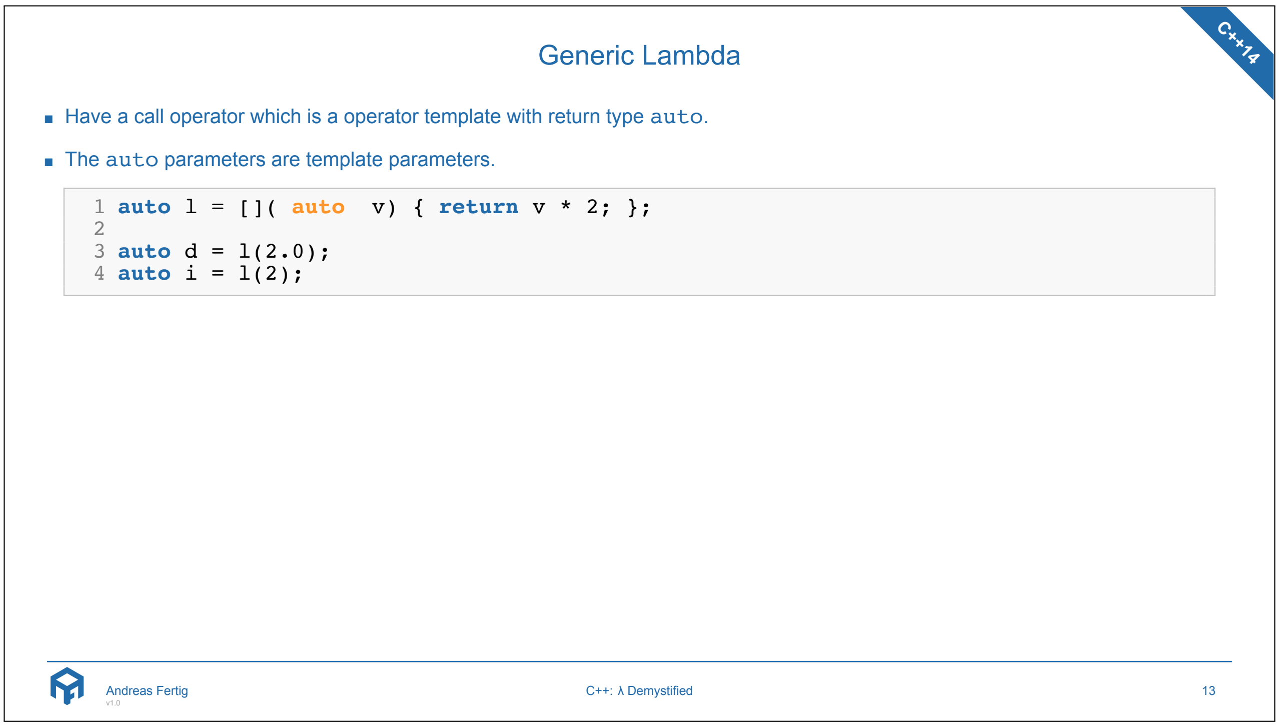
Assume a x64 Platform.

Size of a Lambda

“[...] An implementation may define the closure type differently from what is described below provided this does not alter the observable behavior of the program other than by changing:

(2.1) the size and/or alignment of the closure type,
(2.2) whether the closure type is trivially copyable (10.1), or
(2.3) whether the closure type is a standard-layout class (10.1).
[...]”

— N4800 § 7.5.5.1 p2 [1]



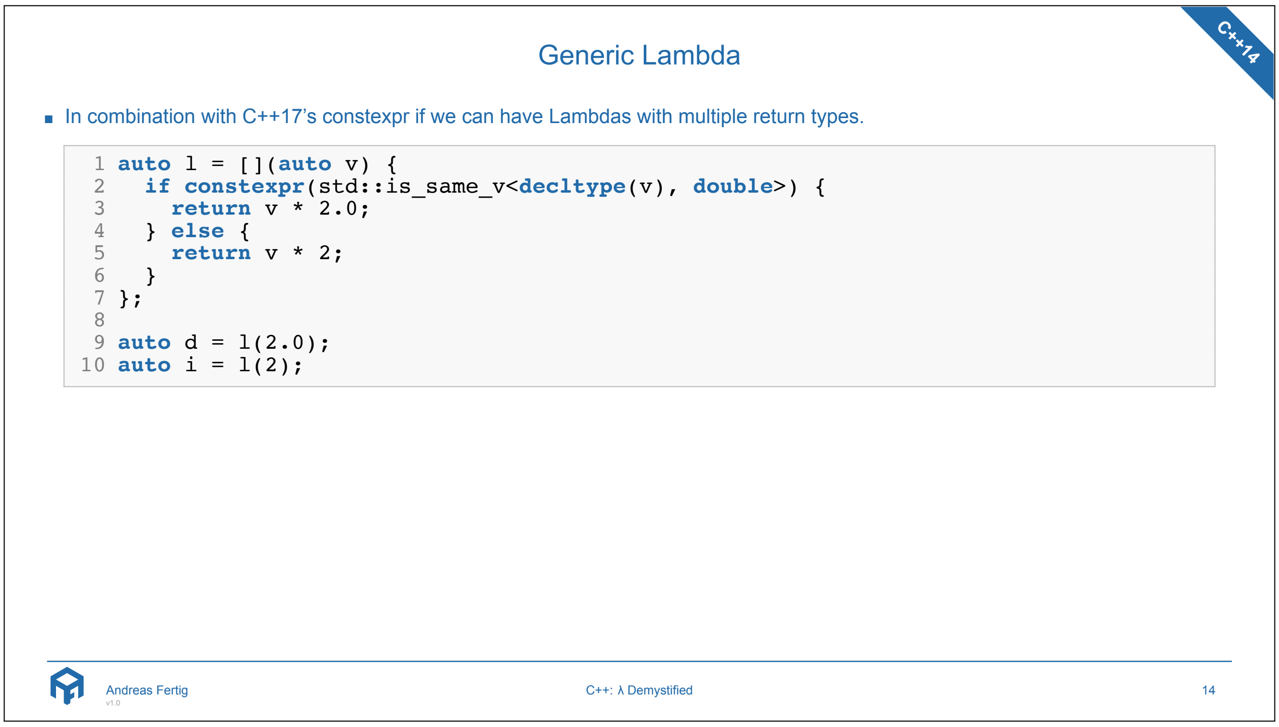
C++14

Generic Lambda

- Have a call operator which is a operator template with return type `auto`.
- The `auto` parameters are template parameters.

```cpp
auto l = []( auto  v) { return v * 2; };

auto d = l(2.0);
auto i = l(2);
```

Andreas Fertig v1.0 C++: λ Demystified 13



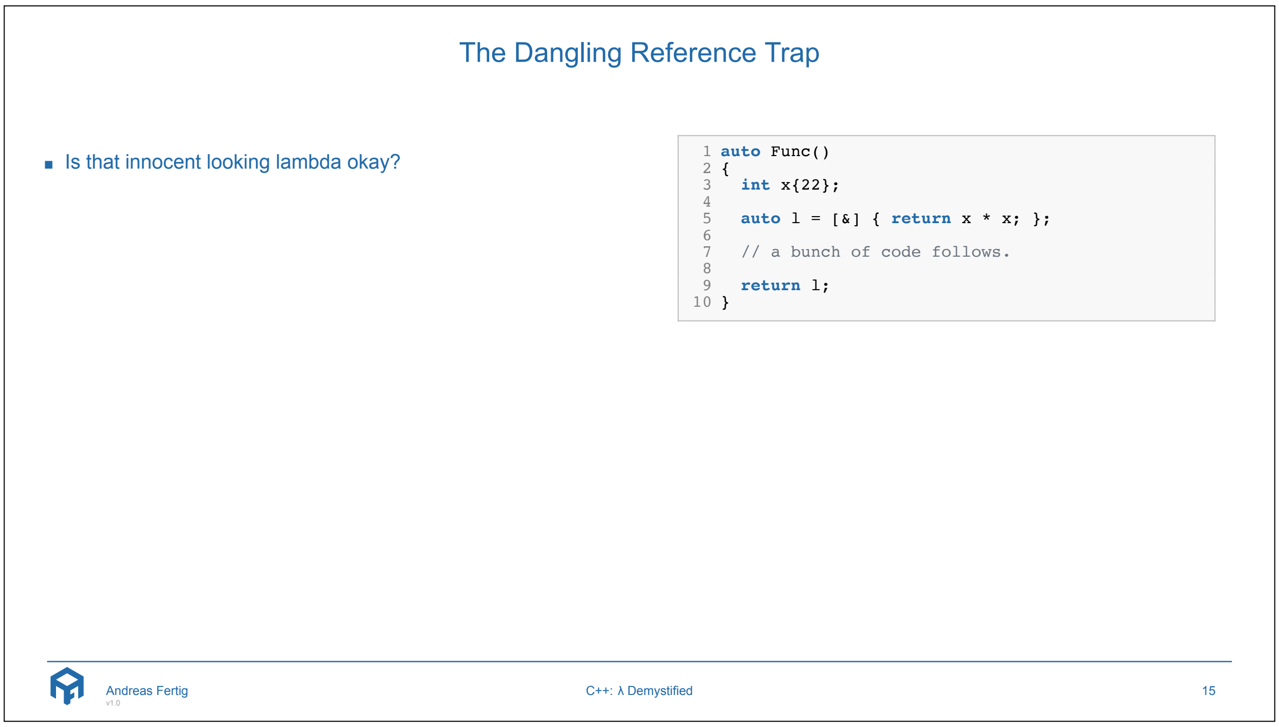
C++14

Generic Lambda

- In combination with C++17's constexpr if we can have Lambdas with multiple return types.

```cpp
auto l = [](auto v) {
  if constexpr(std::is_same_v<decltype(v), double>) {
    return v * 2.0;
  } else {
    return v * 2;
  }
};

auto d = l(2.0);
auto i = l(2);
```

Andreas Fertig v1.0 C++: λ Demystified 14

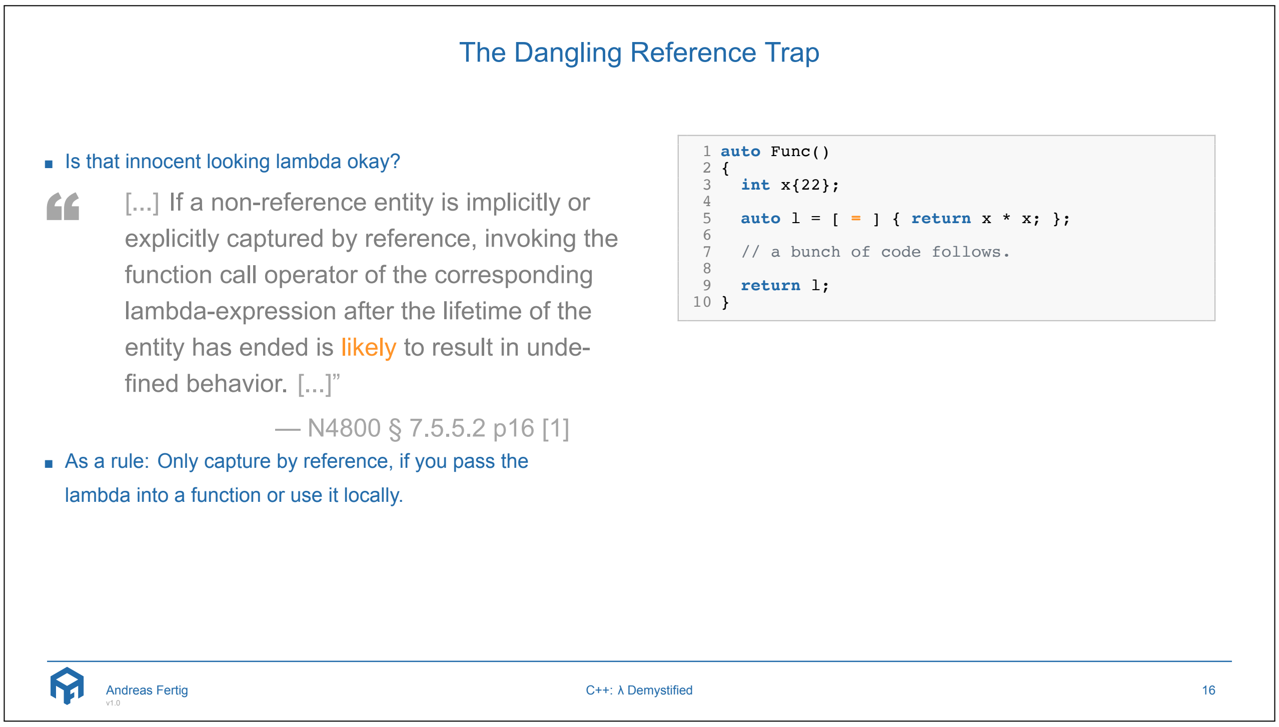


The Dangling Reference Trap

- Is that innocent looking lambda okay?

```
auto Func()
{
  int x{22};

  auto l = [&] { return x * x; };

  // a bunch of code follows.

  return l;
}
```

Andreas Fertig v1.0 C++: λ Demystified 15



The Dangling Reference Trap

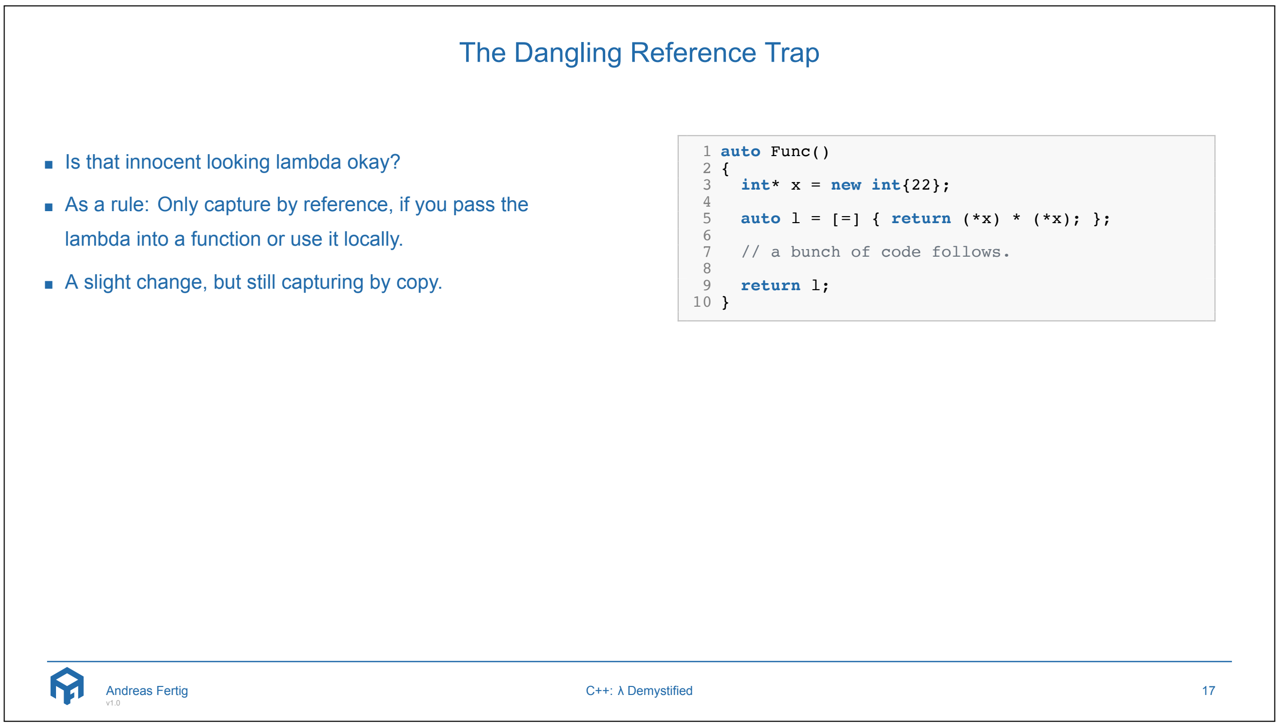
- Is that innocent looking lambda okay?

> [...] If a non-reference entity is implicitly or explicitly captured by reference, invoking the function call operator of the corresponding lambda-expression after the lifetime of the entity has ended is likely to result in undefined behavior. [...]”
>
> — N4800 § 7.5.5.2 p16 [1]

- As a rule: Only capture by reference, if you pass the lambda into a function or use it locally.

```
auto Func()
{
  int x{22};

  auto l = [ = ] { return x * x; };

  // a bunch of code follows.

  return l;
}
```

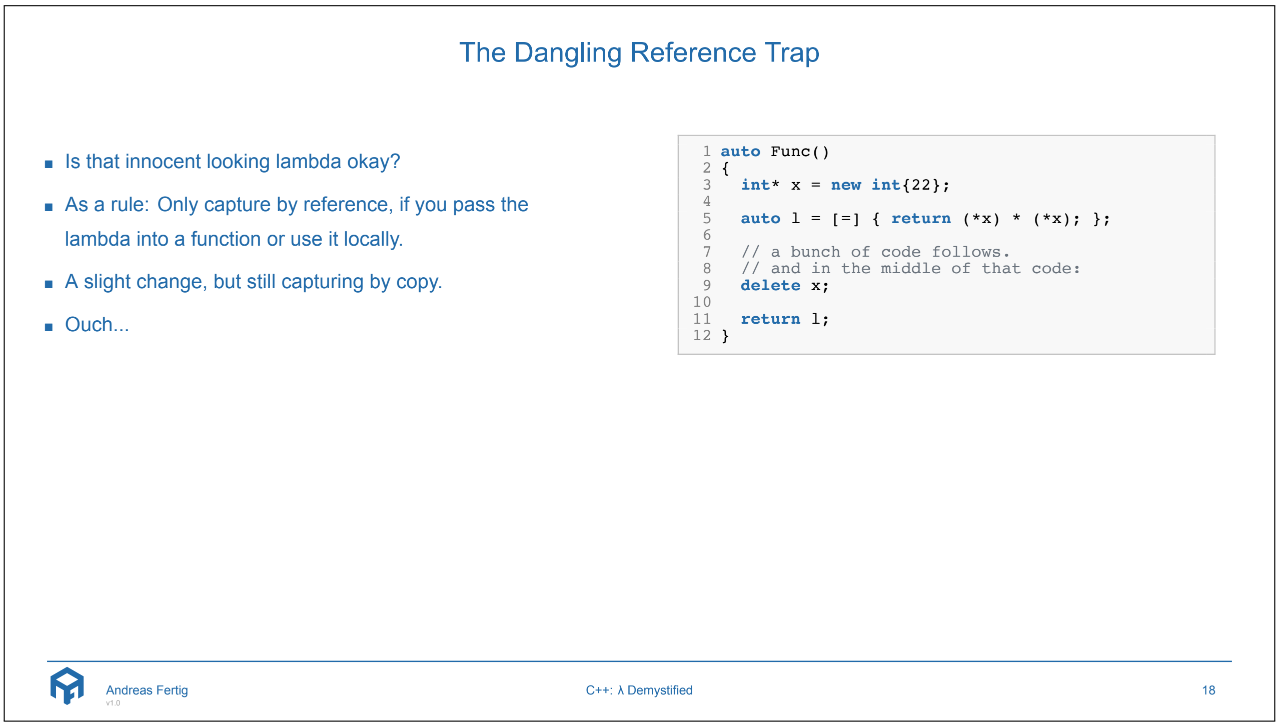
Andreas Fertig v1.0 C++: λ Demystified 16



The Dangling Reference Trap

- Is that innocent looking lambda okay?
- As a rule: Only capture by reference, if you pass the lambda into a function or use it locally.
- A slight change, but still capturing by copy.

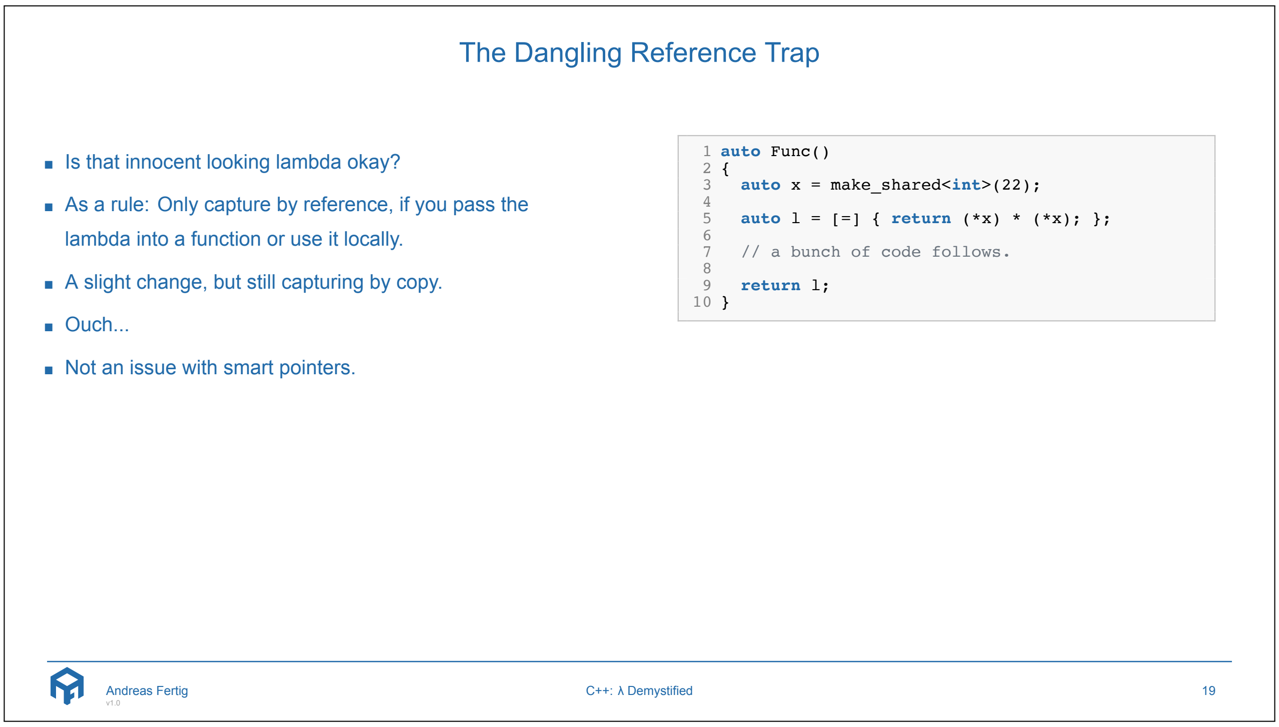
```cpp
auto Func()
{
  int* x = new int{22};

  auto l = [=] { return (*x) * (*x); };

  // a bunch of code follows.

  return l;
}
```



The Dangling Reference Trap

- Is that innocent looking lambda okay?
- As a rule: Only capture by reference, if you pass the lambda into a function or use it locally.
- A slight change, but still capturing by copy.
- Ouch...

```cpp
auto Func()
{
  int* x = new int{22};

  auto l = [=] { return (*x) * (*x); };

  // a bunch of code follows.
  // and in the middle of that code:
  delete x;

  return l;
}
```

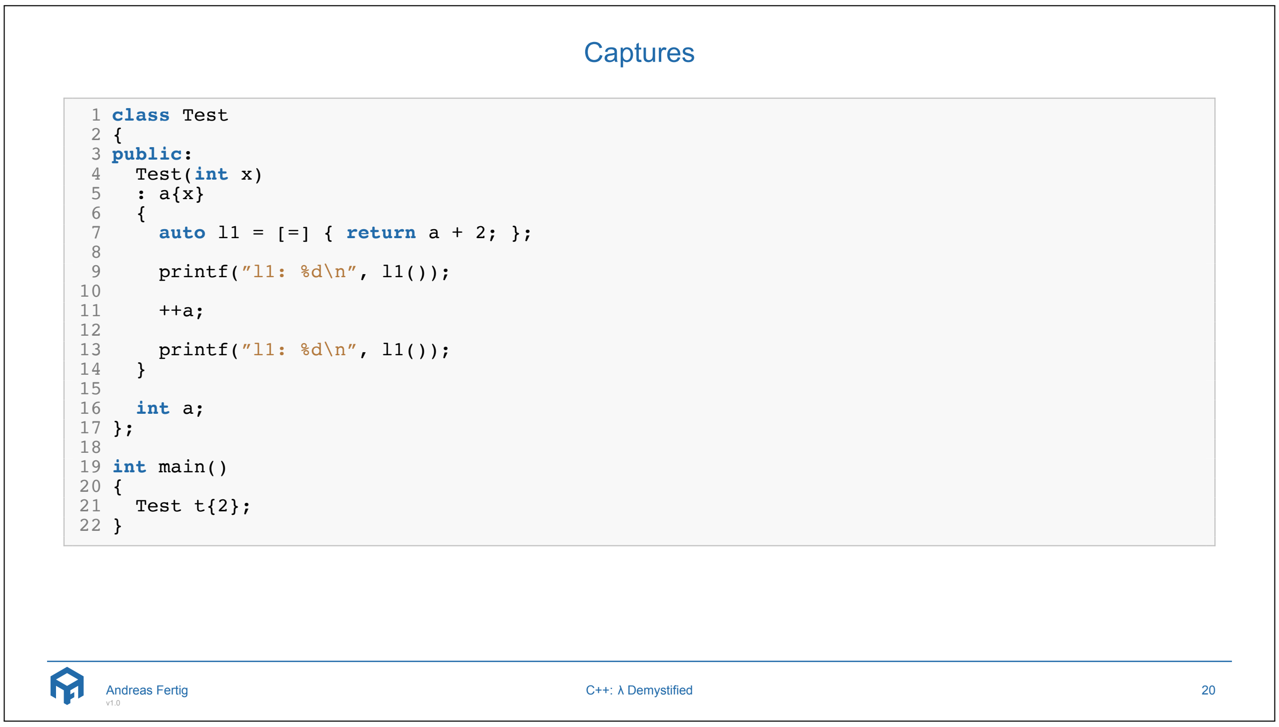


The Dangling Reference Trap

- Is that innocent looking lambda okay?
- As a rule: Only capture by reference, if you pass the lambda into a function or use it locally.
- A slight change, but still capturing by copy.
- Ouch...
- Not an issue with smart pointers.

```cpp
auto Func()
{
  auto x = make_shared<int>(22);

  auto l = [=] { return (*x) * (*x); };

  // a bunch of code follows.

  return l;
}
```

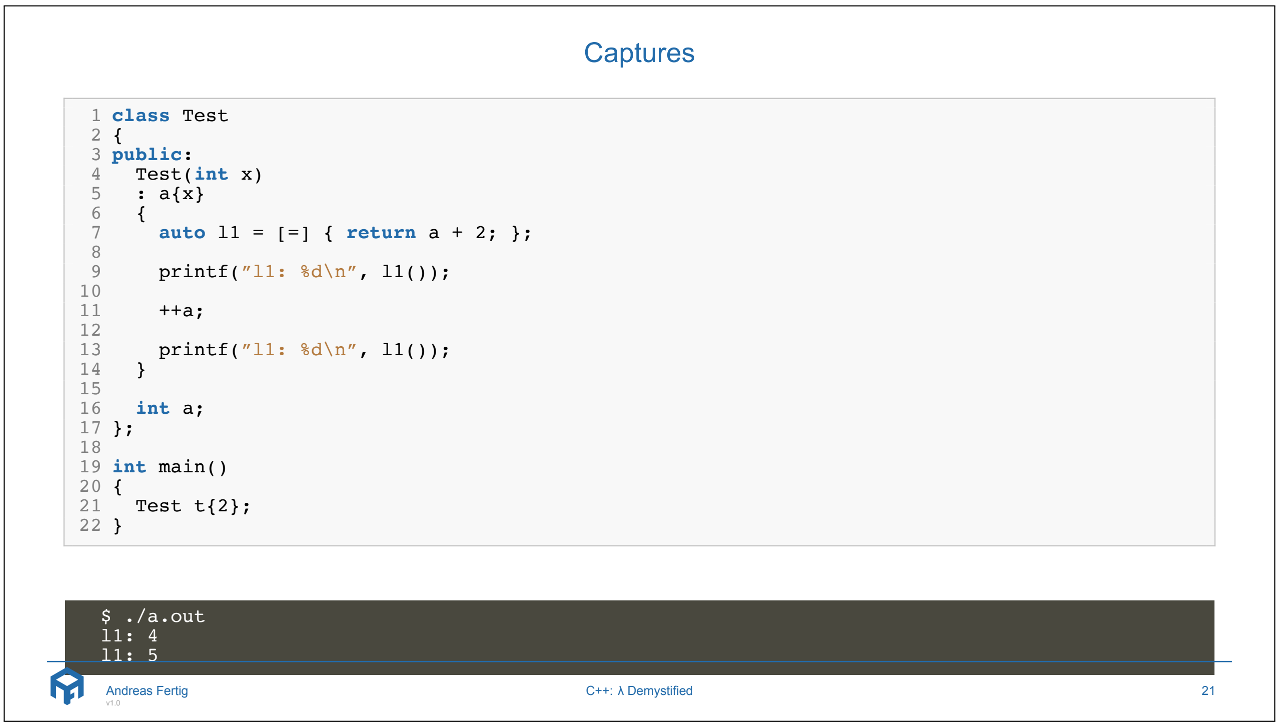
Andreas Fertig v1.0 C++: λ Demystified 19



Captures

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

    printf(”l1: %d\n”, l1());

    ++a;

    printf(”l1: %d\n”, l1());
  }

  int a;
};

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

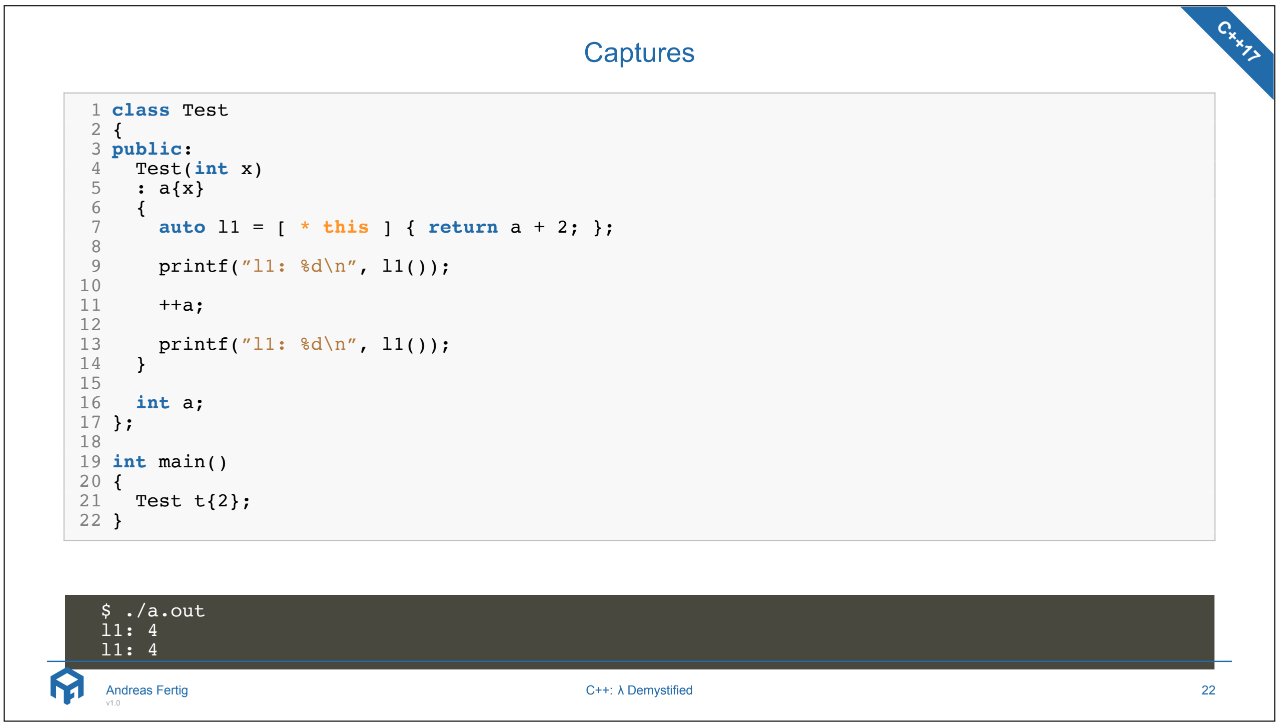
Andreas Fertig v1.0 C++: λ Demystified 20



Captures

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

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

    ++a;

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

  int a;
};

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

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

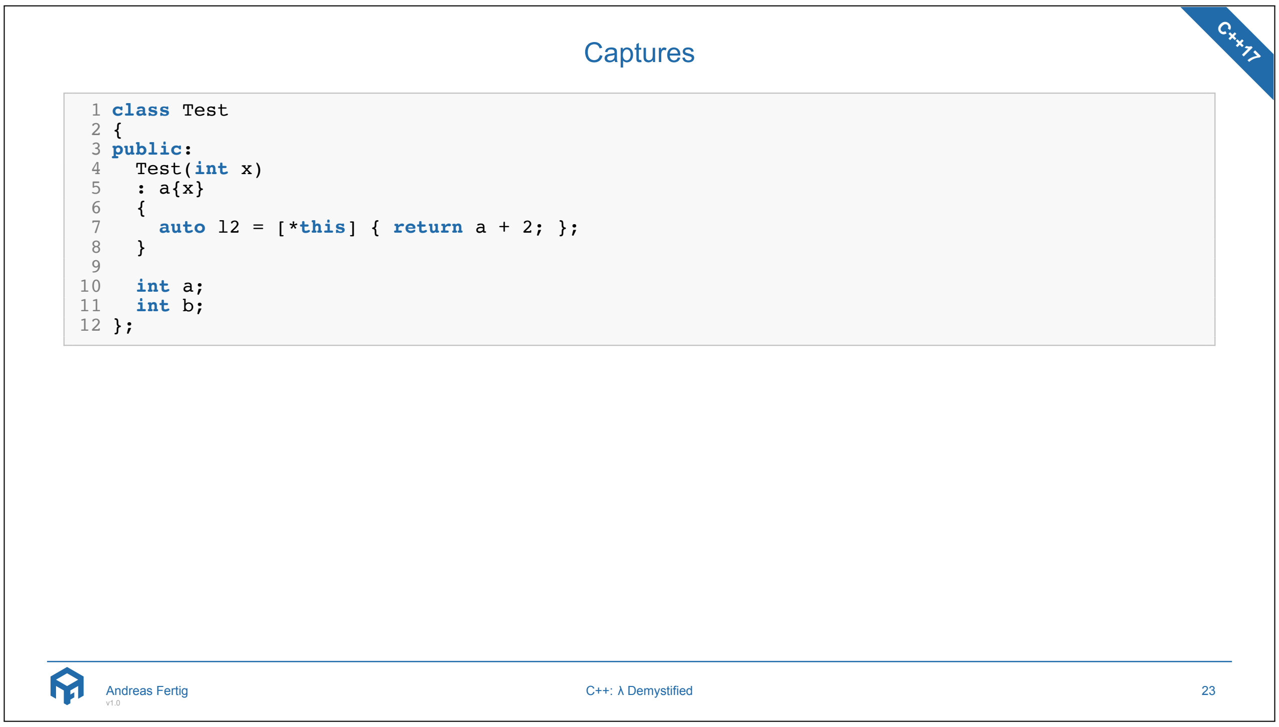


Captures

C++17

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

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

    ++a;

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

  int a;
};

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

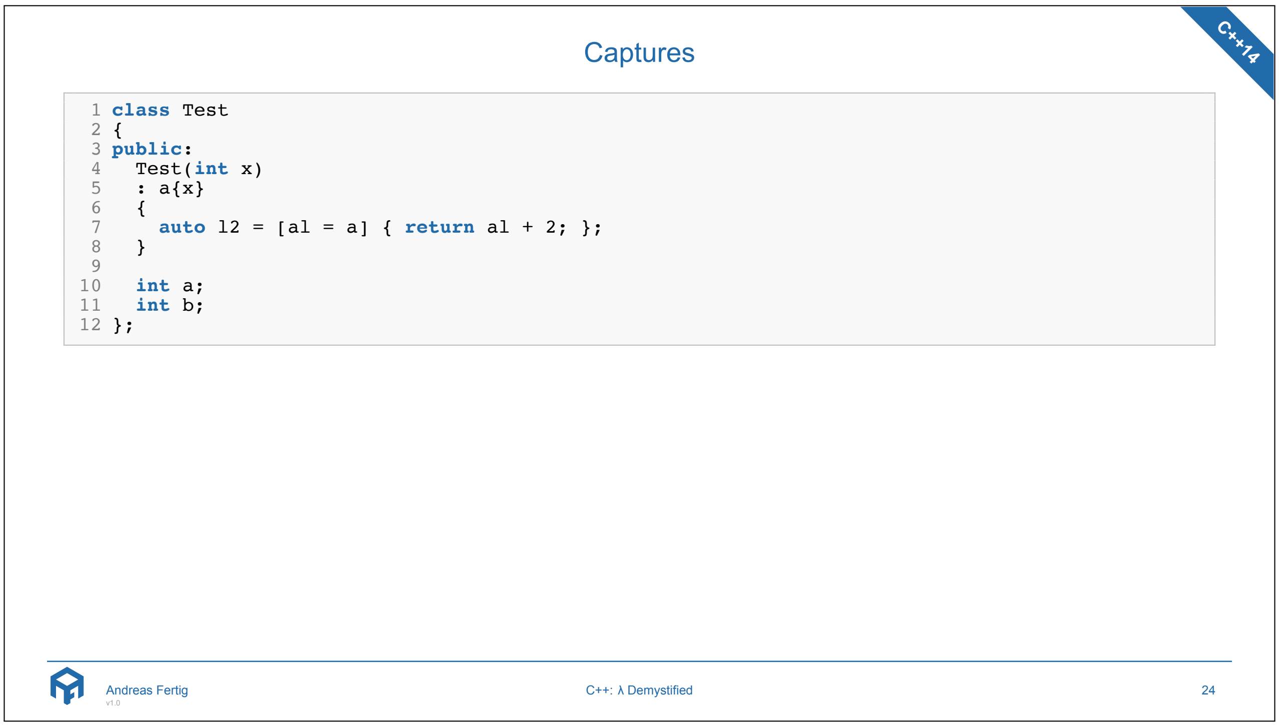
```
$ ./a.out
l1: 4
l1: 4
```



C++17

Captures

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

  int a;
  int b;
};
```



C++14

Captures

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

  int a;
  int b;
};
```

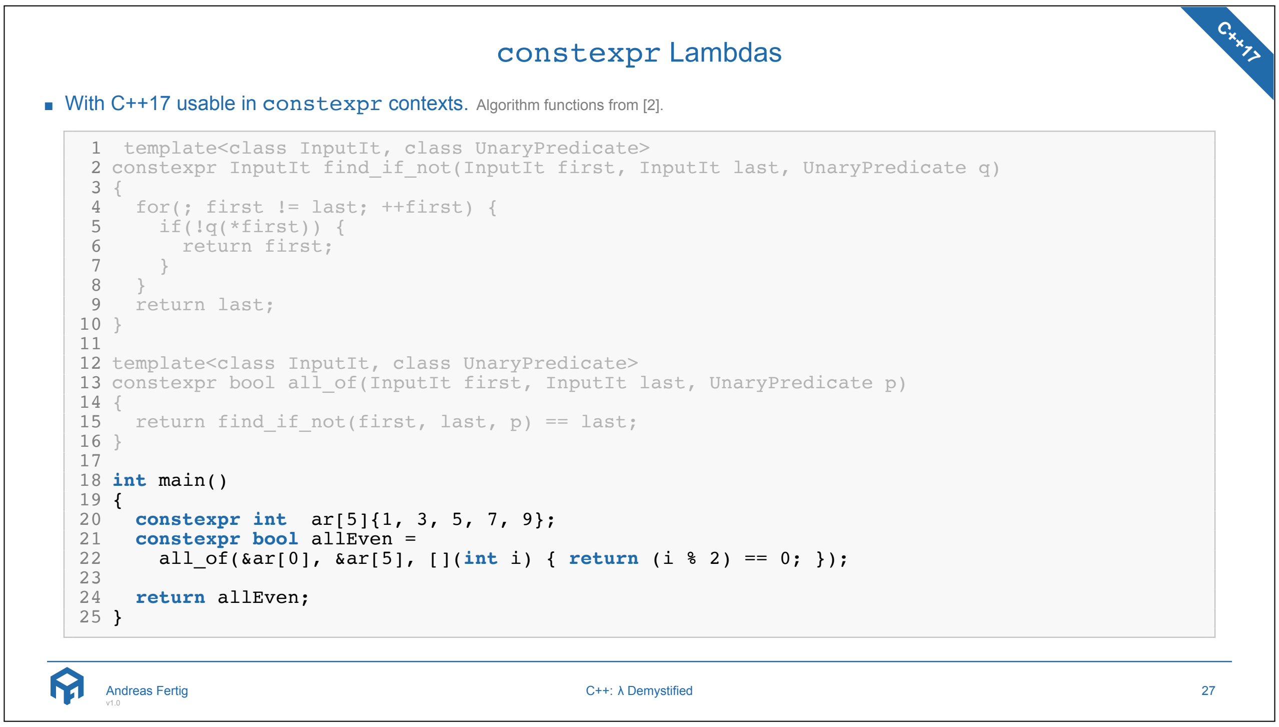
Size of a Lambda - Part II

```cpp
class Test
{
public:
  Test(int x)
  : a{x}
  {
    const int size = 2;

    auto l2 = [=] {
      int x[size]{};

      return a + 2;
    };
  }

  int a;
};
```

Sleeping Lambda

```cpp
int main()
{
  std::string foo;

  auto a = [=]    () { printf( "%s\n", foo.c_str()); };

  auto b = [=]    () { };

  auto c = [foo]  () { printf( "%s\n", foo.c_str()); };

  auto d = [foo]  () { };

  auto e = [&foo] () { printf( "%s\n", foo.c_str()); };

  auto f = [&foo] () { };
}
```

Assume a x64 Platform.

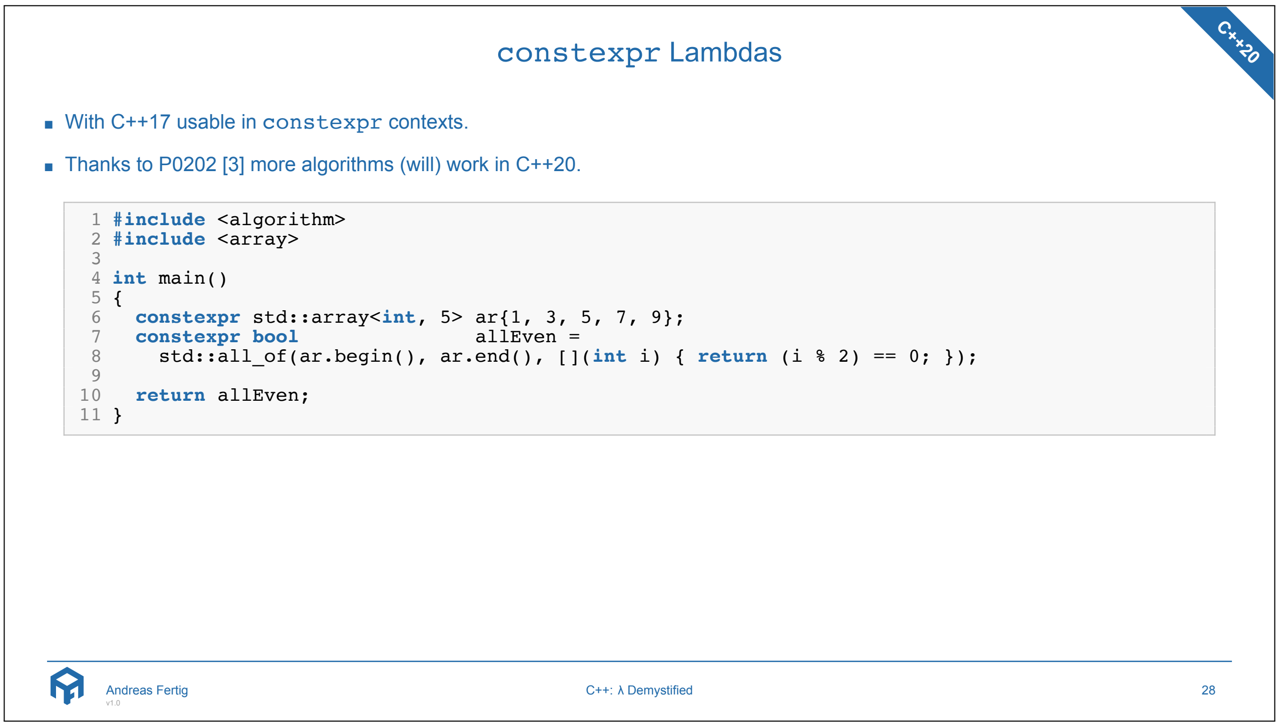


C++17

`constexpr` Lambdas

- With C++17 usable in `constexpr` contexts. Algorithm functions from [2].

```cpp
 template<class InputIt, class UnaryPredicate>
constexpr InputIt find_if_not(InputIt first, InputIt last, UnaryPredicate q)
{
  for(; first != last; ++first) {
    if(!q(*first)) {
      return first;
    }
  }
  return last;
}

template<class InputIt, class UnaryPredicate>
constexpr bool all_of(InputIt first, InputIt last, UnaryPredicate p)
{
  return find_if_not(first, last, p) == last;
}

int main()
{
  constexpr int  ar[5]{1, 3, 5, 7, 9};
  constexpr bool allEven =
    all_of(&ar[0], &ar[5], [](int i) { return (i % 2) == 0; });

  return allEven;
}
```

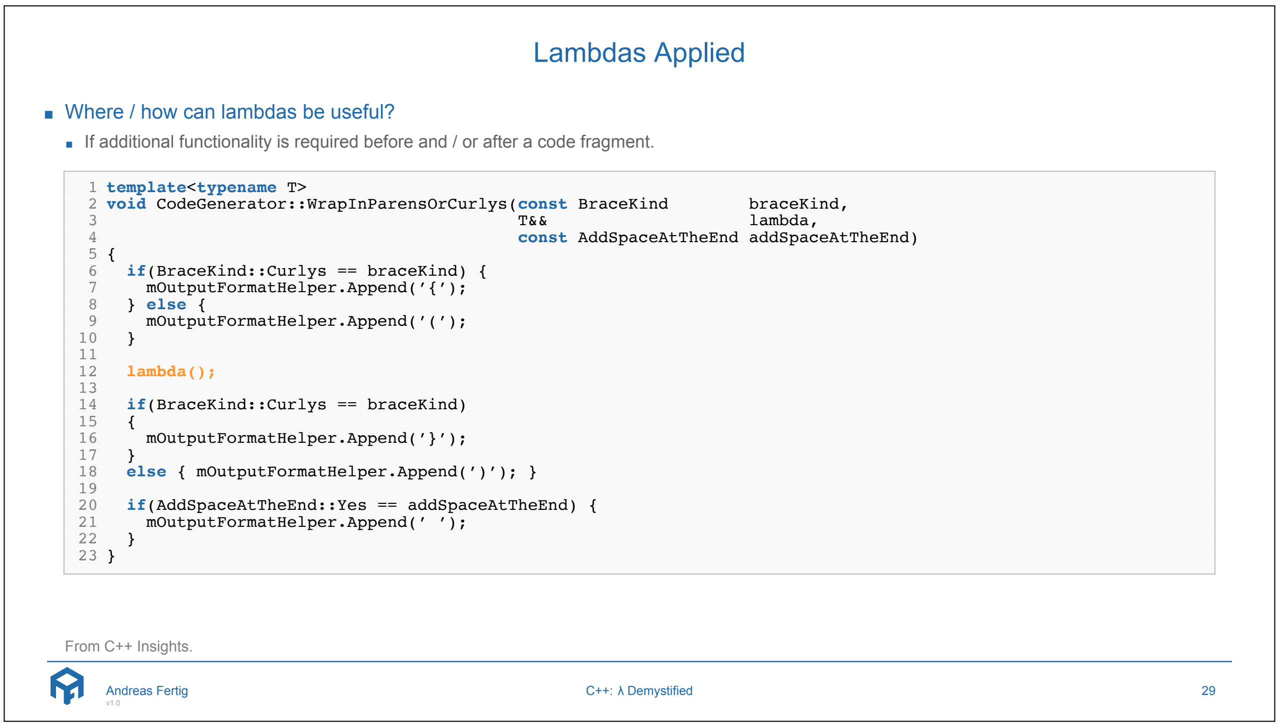


C++20

`constexpr` Lambdas

- With C++17 usable in `constexpr` contexts.
- Thanks to P0202 [3] more algorithms (will) work in C++20.

```cpp
#include <algorithm>
#include <array>

int main()
{
  constexpr std::array<int, 5> ar{1, 3, 5, 7, 9};
  constexpr bool               allEven =
    std::all_of(ar.begin(), ar.end(), [](int i) { return (i % 2) == 0; });

  return allEven;
}
```



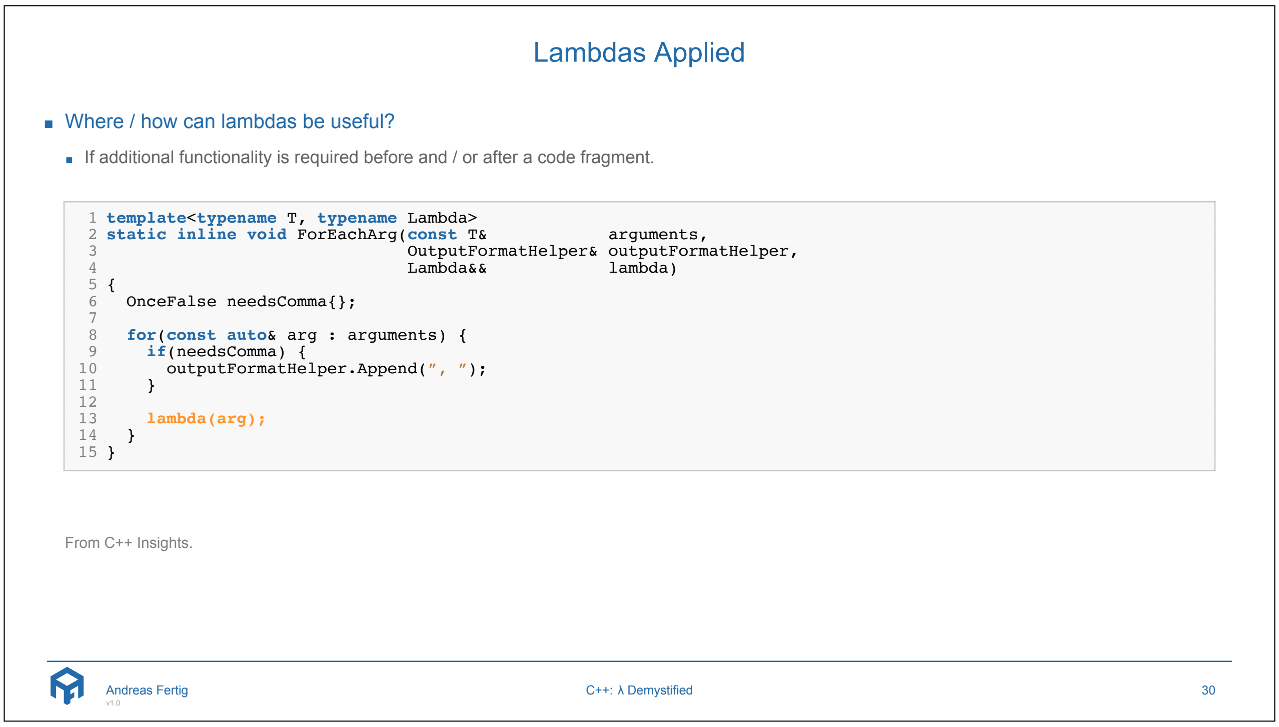
Lambdas Applied

- Where / how can lambdas be useful?
  - If additional functionality is required before and / or after a code fragment.

```cpp
template<typename T>
void CodeGenerator::WrapInParensOrCurlys(const BraceKind        braceKind,
                                         T&&                    lambda,
                                         const AddSpaceAtTheEnd addSpaceAtTheEnd)
{
  if(BraceKind::Curlys == braceKind) {
    mOutputFormatHelper.Append(’{’);
  } else {
    mOutputFormatHelper.Append(’(’);
  }

  lambda();

  if(BraceKind::Curlys == braceKind)
  {
    mOutputFormatHelper.Append(’}’);
  }
  else { mOutputFormatHelper.Append(’)’); }

  if(AddSpaceAtTheEnd::Yes == addSpaceAtTheEnd) {
    mOutputFormatHelper.Append(’ ’);
  }
}
```

From C++ Insights.

Andreas Fertig v1.0 C++: λ Demystified 29



Lambdas Applied

- Where / how can lambdas be useful?
  - If additional functionality is required before and / or after a code fragment.

```cpp
template<typename T, typename Lambda>
static inline void ForEachArg(const T&            arguments,
                              OutputFormatHelper& outputFormatHelper,
                              Lambda&&            lambda)
{
  OnceFalse needsComma{};

  for(const auto& arg : arguments) {
    if(needsComma) {
      outputFormatHelper.Append(", ");
    }

    lambda(arg);
  }
}
```

From C++ Insights.

Andreas Fertig v1.0 C++: λ Demystified 30

Lambdas Applied

- Where / how can lambdas be useful?
  - To achieve more const'ness for variables.
  - Also known as *Immediately-invoked function expression* [4].

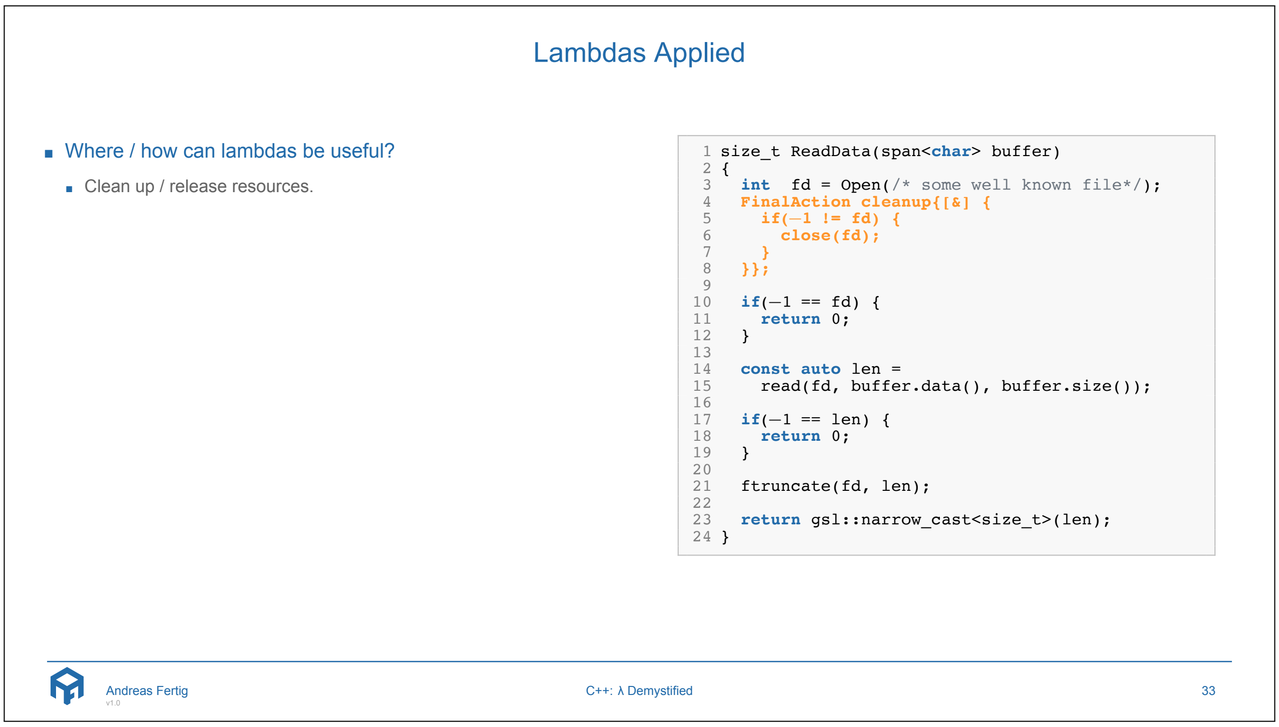
```cpp
const auto name = [&]() -> std::string {
  // Handle a special case where we have a lambda
  // static invoke operator. In that case use the
  // appropriate using retType as return type
  if(const auto* m = dyn_cast_or_null<CXXMethodDecl>(meDecl)) {
    if(const auto* rd = m->getParent(); rd && rd->isLambda()) {
      skipTemplateArgs = true;

      return StrCat("operator ", GetLambdaName(*rd), "::", BuildRetTypeName(*rd));
    }
  }

  return stmt->getMemberNameInfo().getName().getAsString();
}();
```

From C++ Insights.

Lambdas Applied

- Where / how can lambdas be useful?
  - Clean up / release resources.

```cpp
size_t ReadData(span<char> buffer)
{
  int fd = Open(/*some well known file*/);

  if-(1 == fd) {
    return 0;
  }

  const auto len =
    read(fd, buffer.data(), buffer.size());

  if-(1 == len) {
    return 0;
  }

  ftruncate(fd, len);

  close(fd);

  return gsl::narrow_cast<size_t>(len);
}
```

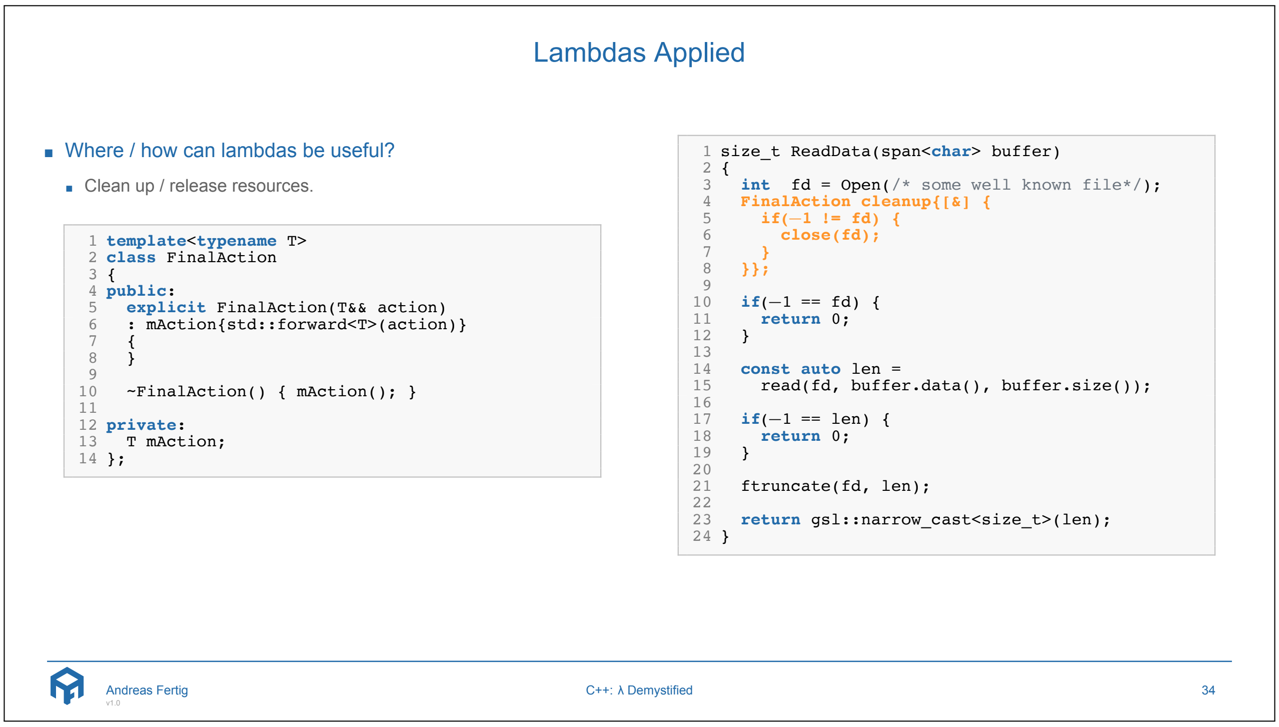


Lambdas Applied

- Where / how can lambdas be useful?
  - Clean up / release resources.

```cpp
size_t ReadData(span<char> buffer)
{
  int  fd = Open(/* some well known file*/);
  FinalAction cleanup{[&] {
    if(-1 != fd) {
      close(fd);
    }
  }};

  if(-1 == fd) {
    return 0;
  }

  const auto len =
    read(fd, buffer.data(), buffer.size());

  if(-1 == len) {
    return 0;
  }

  ftruncate(fd, len);

  return gsl::narrow_cast<size_t>(len);
}
```

Andreas Fertig v1.0 C++: λ Demystified 33



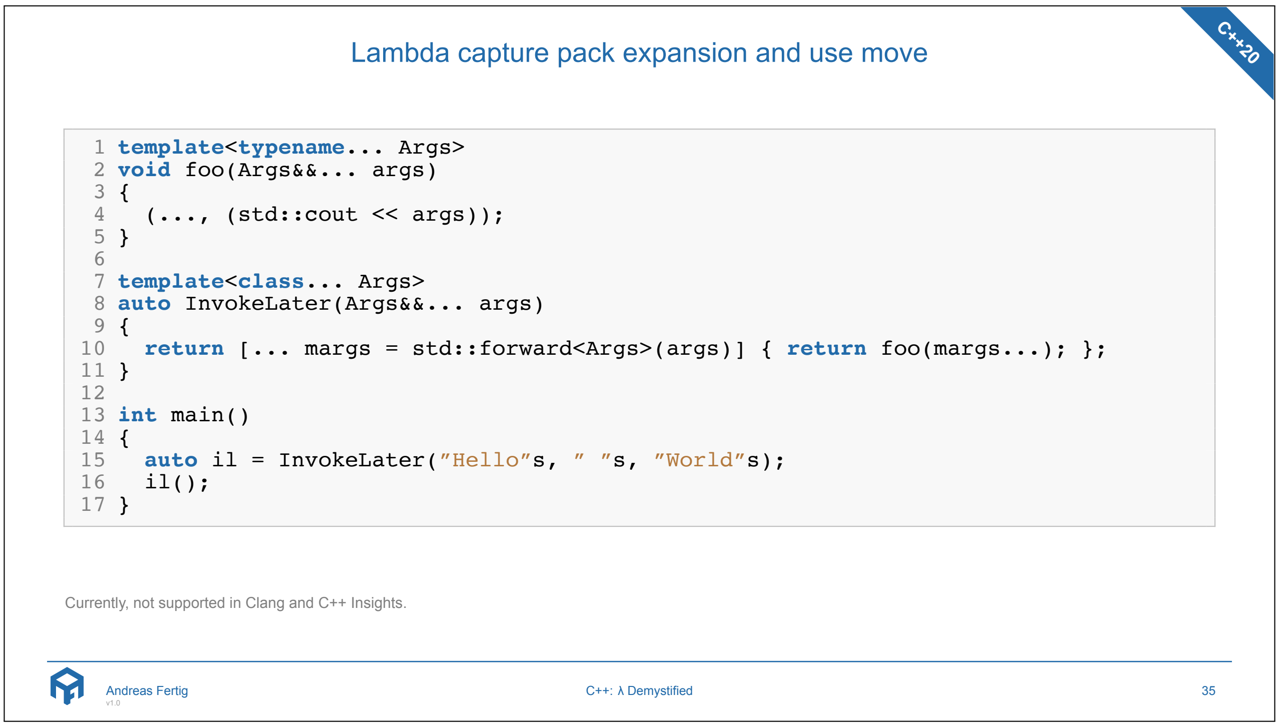
Lambdas Applied

- Where / how can lambdas be useful?
  - Clean up / release resources.

```cpp
template<typename T>
class FinalAction
{
public:
  explicit FinalAction(T&& action)
  : mAction{std::forward<T>(action)}
  {
  }

  ~FinalAction() { mAction(); }

private:
  T mAction;
};
```

```cpp
size_t ReadData(span<char> buffer)
{
  int  fd = Open(/* some well known file*/);
  FinalAction cleanup{[&] {
    if(-1 != fd) {
      close(fd);
    }
  }};

  if(-1 == fd) {
    return 0;
  }

  const auto len =
    read(fd, buffer.data(), buffer.size());

  if(-1 == len) {
    return 0;
  }

  ftruncate(fd, len);

  return gsl::narrow_cast<size_t>(len);
}
```

Andreas Fertig v1.0 C++: λ Demystified 34

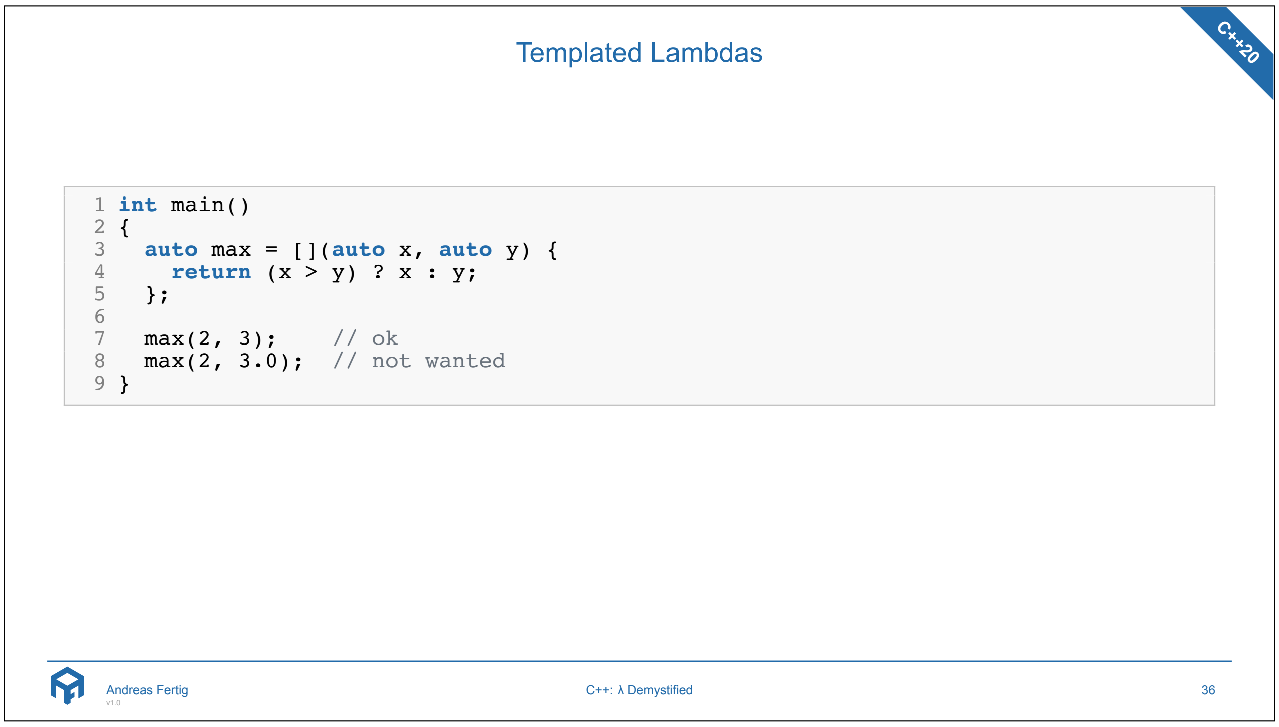


C++20

Lambda capture pack expansion and use move

```cpp
template<typename... Args>
void foo(Args&&... args)
{
  (..., (std::cout << args));
}

template<class... Args>
auto InvokeLater(Args&&... args)
{
  return [... margs = std::forward<Args>(args)] { return foo(margs...); };
}

int main()
{
  auto il = InvokeLater("Hello"s, " "s, "World"s);
  il();
}
```

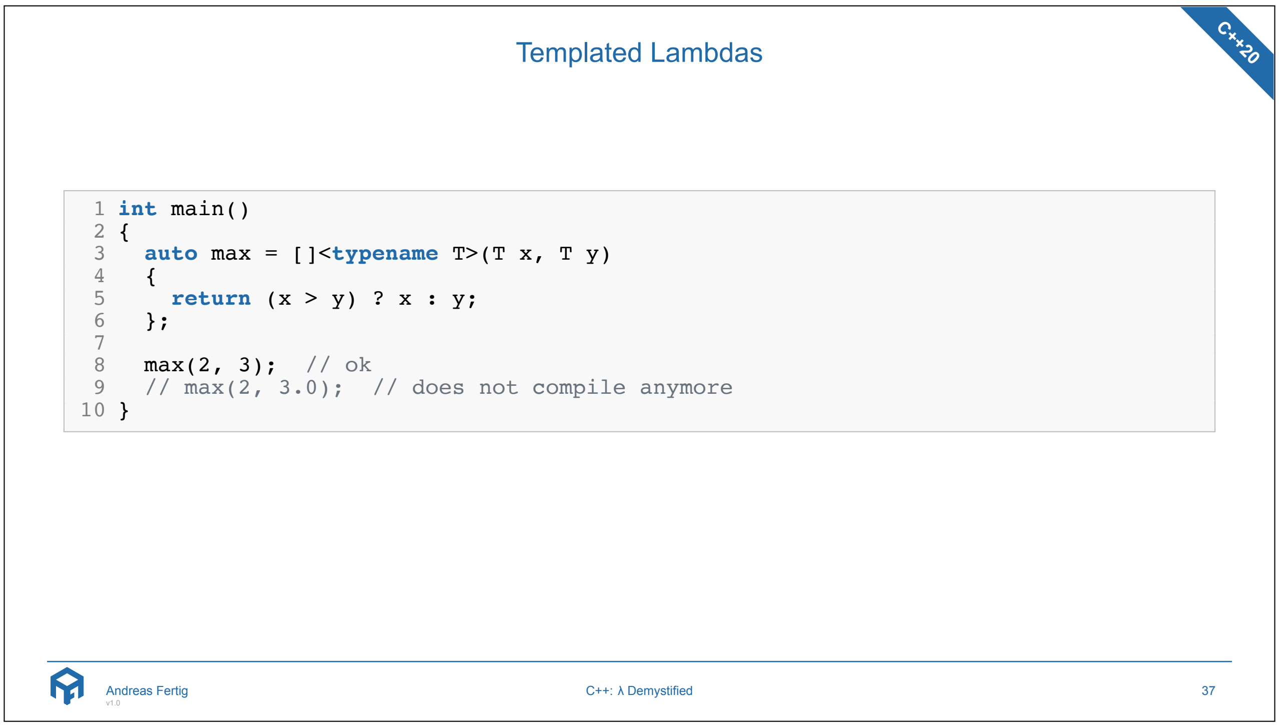
Currently, not supported in Clang and C++ Insights.



C++20

Templated Lambdas

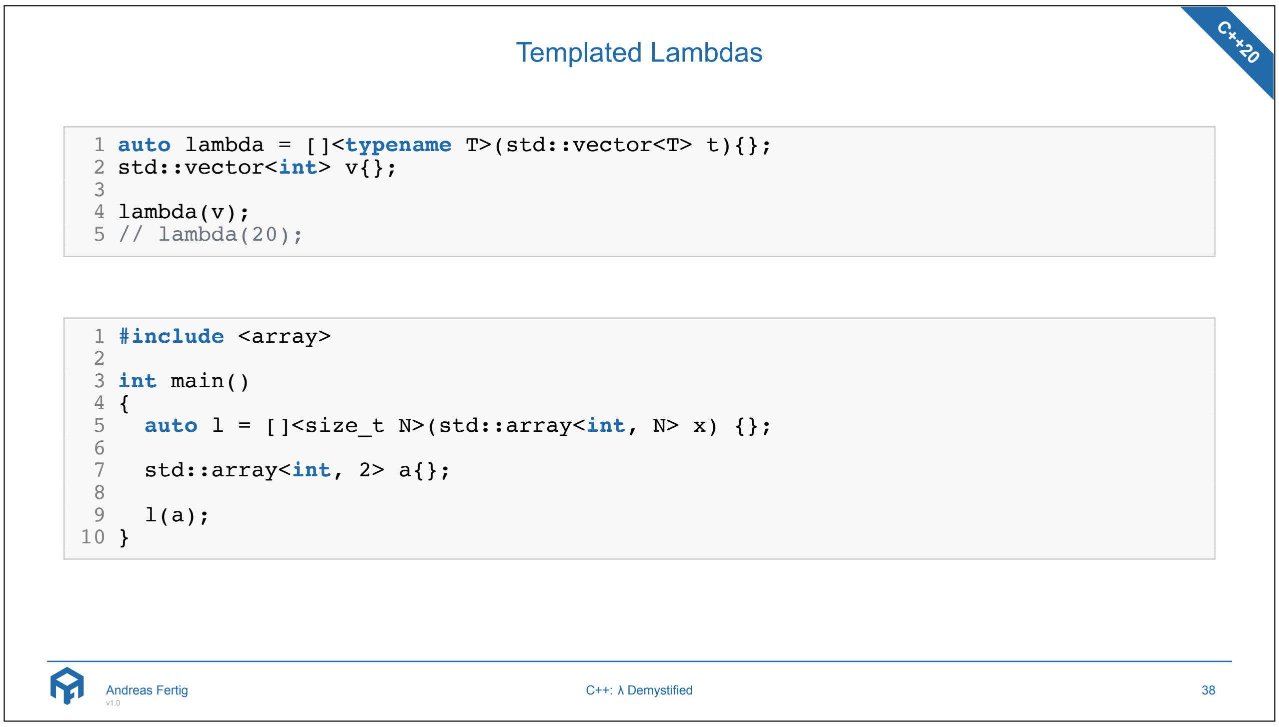
```cpp
int main()
{
  auto max = [](auto x, auto y) {
    return (x > y) ? x : y;
  };

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



Templated Lambdas

C++20

```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
}
```

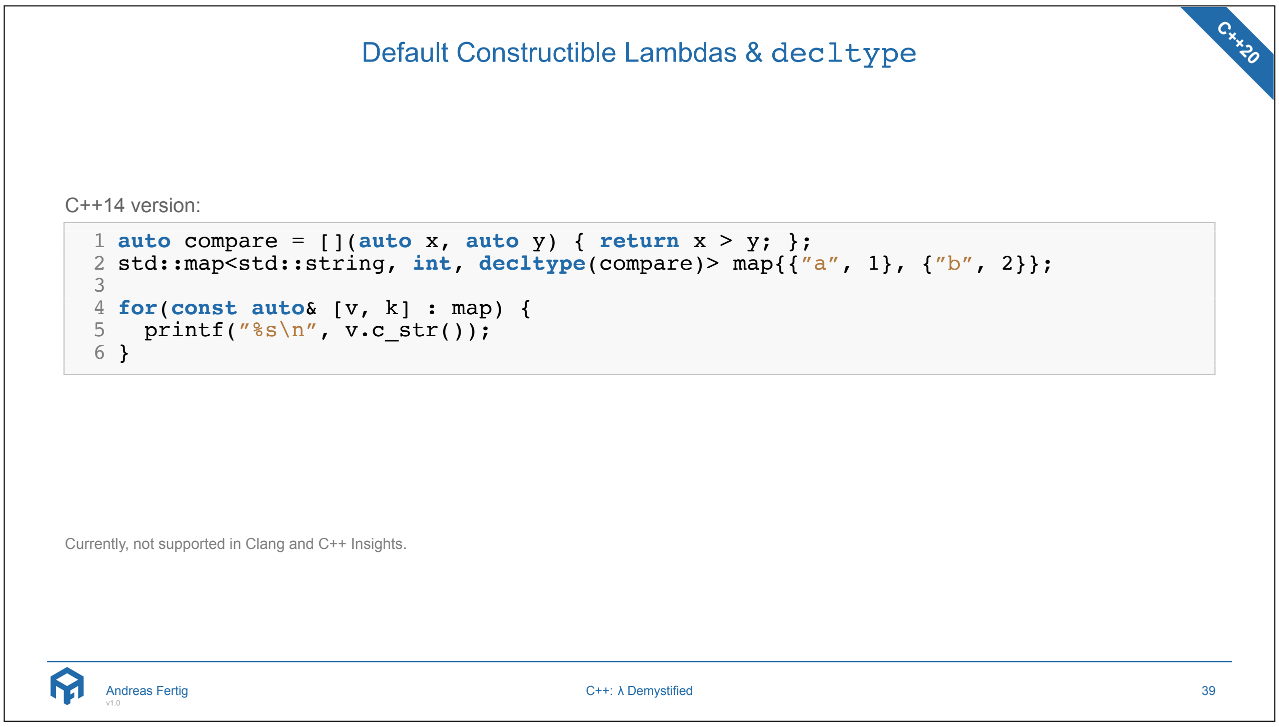


Templated Lambdas

C++20

```cpp
auto lambda = []<typename T>(std::vector<T> t){};
std::vector<int> v{};

lambda(v);
// lambda(20);
```

```cpp
#include <array>

int main()
{
  auto l = []<size_t N>(std::array<int, N> x) {};

  std::array<int, 2> a{};

  l(a);
}
```



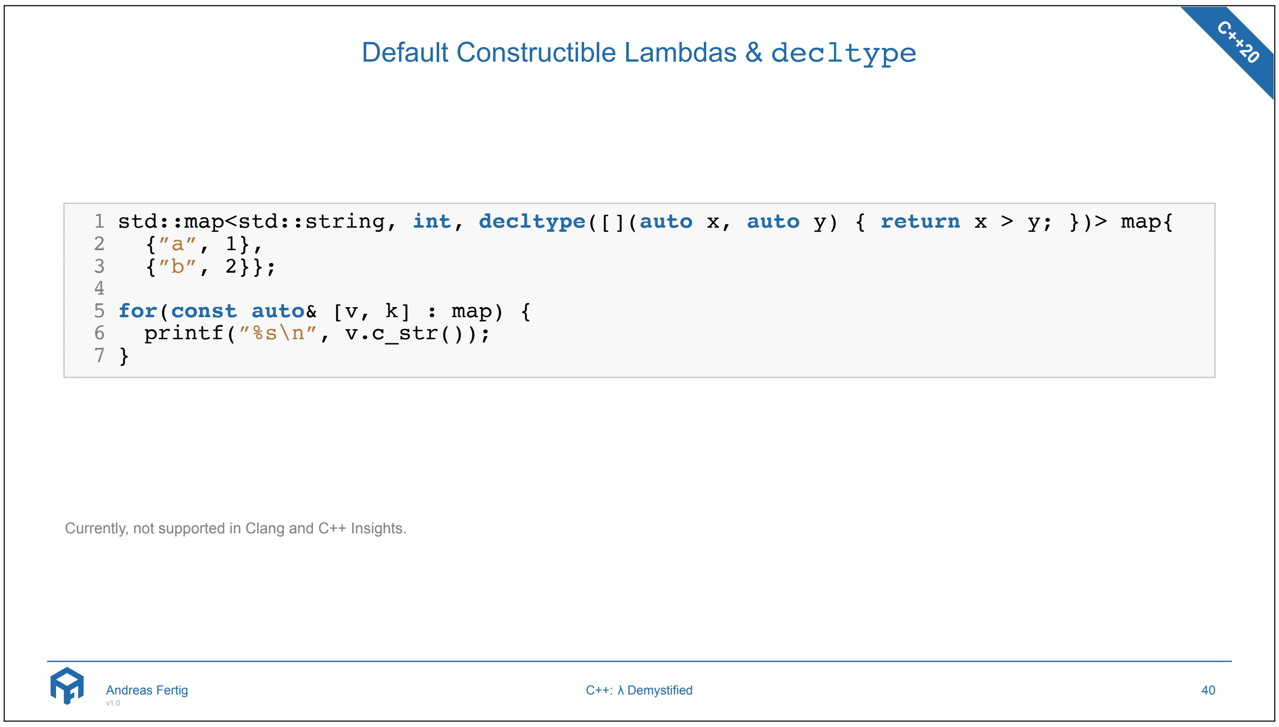
Default Constructible Lambdas & decltype

C++20

C++14 version:

```cpp
auto compare = [](auto x, auto y) { return x > y; };
std::map<std::string, int, decltype(compare)> map{{"a", 1}, {"b", 2}};

for(const auto& [v, k] : map) {
  printf("%s\n", v.c_str());
}
```

Currently, not supported in Clang and C++ Insights.

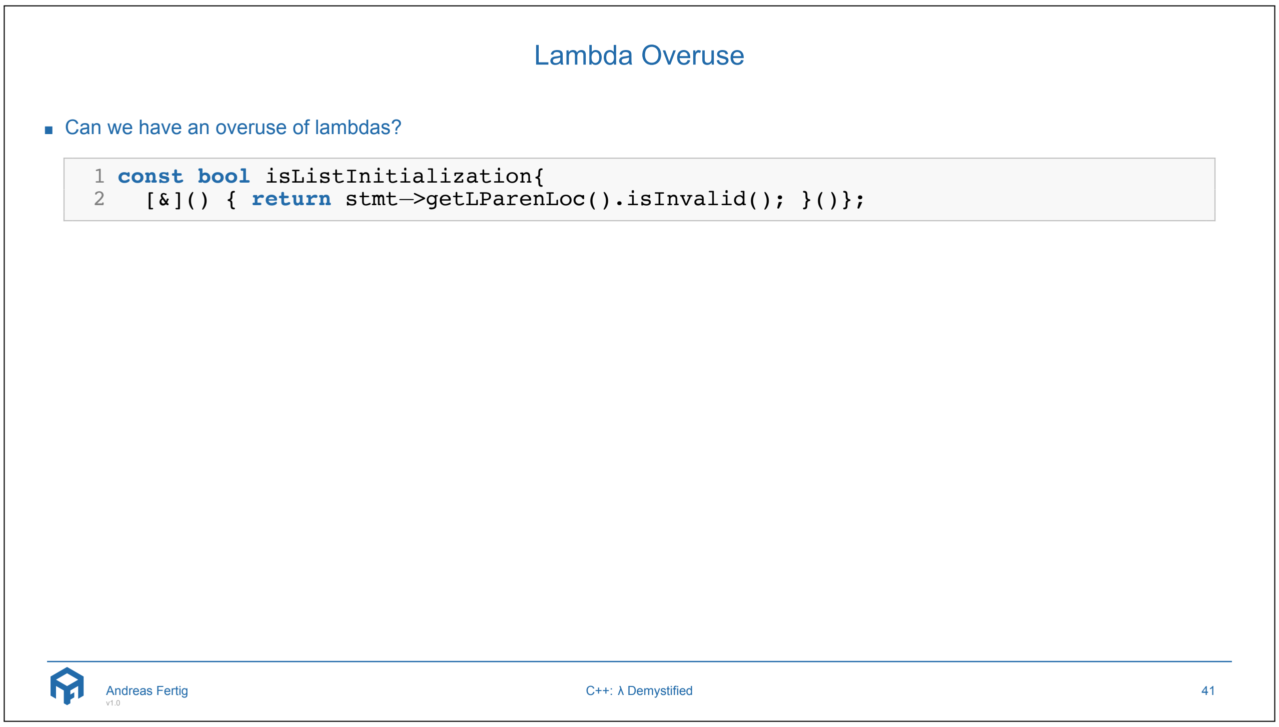


Default Constructible Lambdas & decltype

C++20

```cpp
std::map<std::string, int, decltype([](auto x, auto y) { return x > y; })> map{
  {"a", 1},
  {"b", 2}};

for(const auto& [v, k] : map) {
  printf("%s\n", v.c_str());
}
```

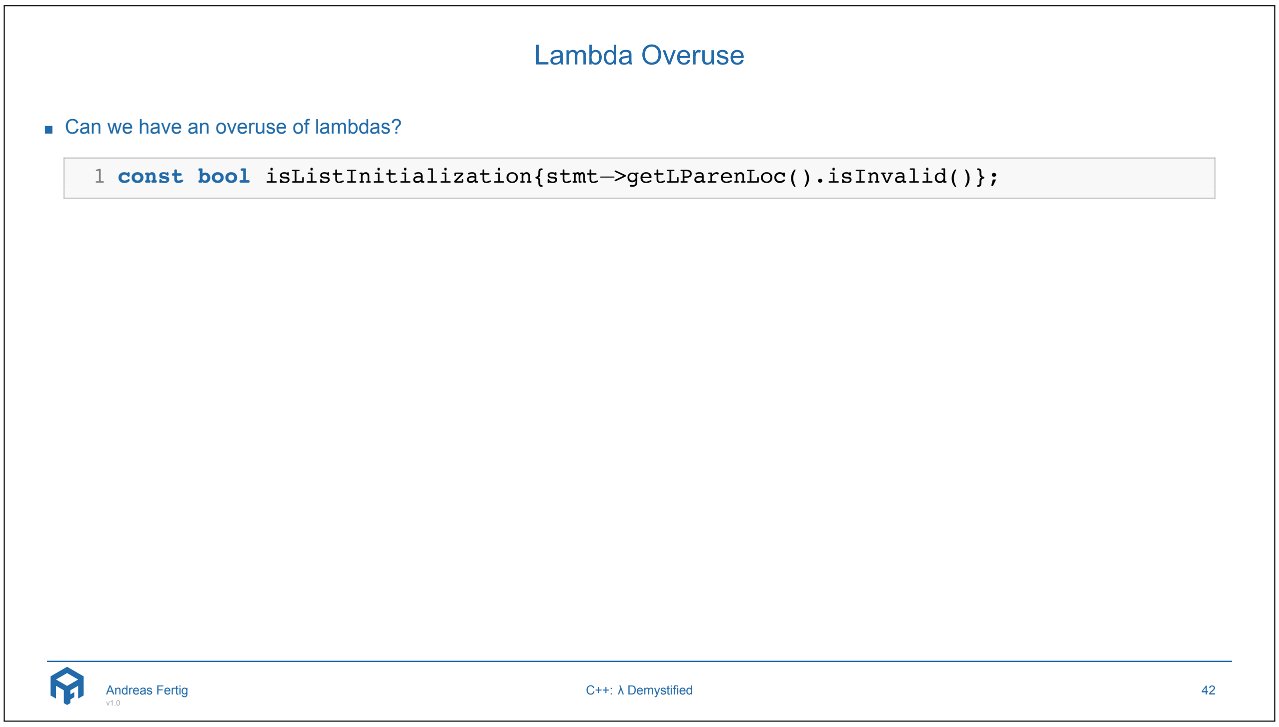
Currently, not supported in Clang and C++ Insights.



Lambda Overuse

- Can we have an overuse of lambdas?

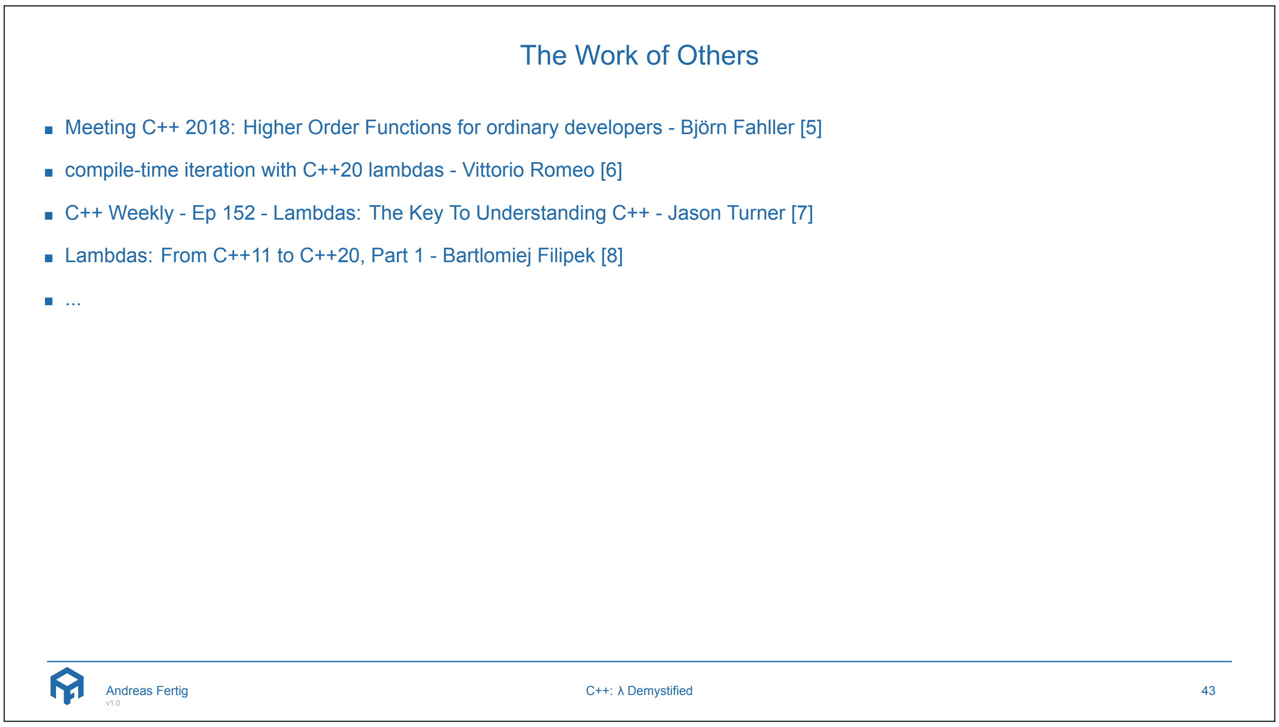
```cpp
const bool isListInitialization{
  [&]() { return stmt->getLParenLoc().isInvalid(); }()};
```



Lambda Overuse

- Can we have an overuse of lambdas?

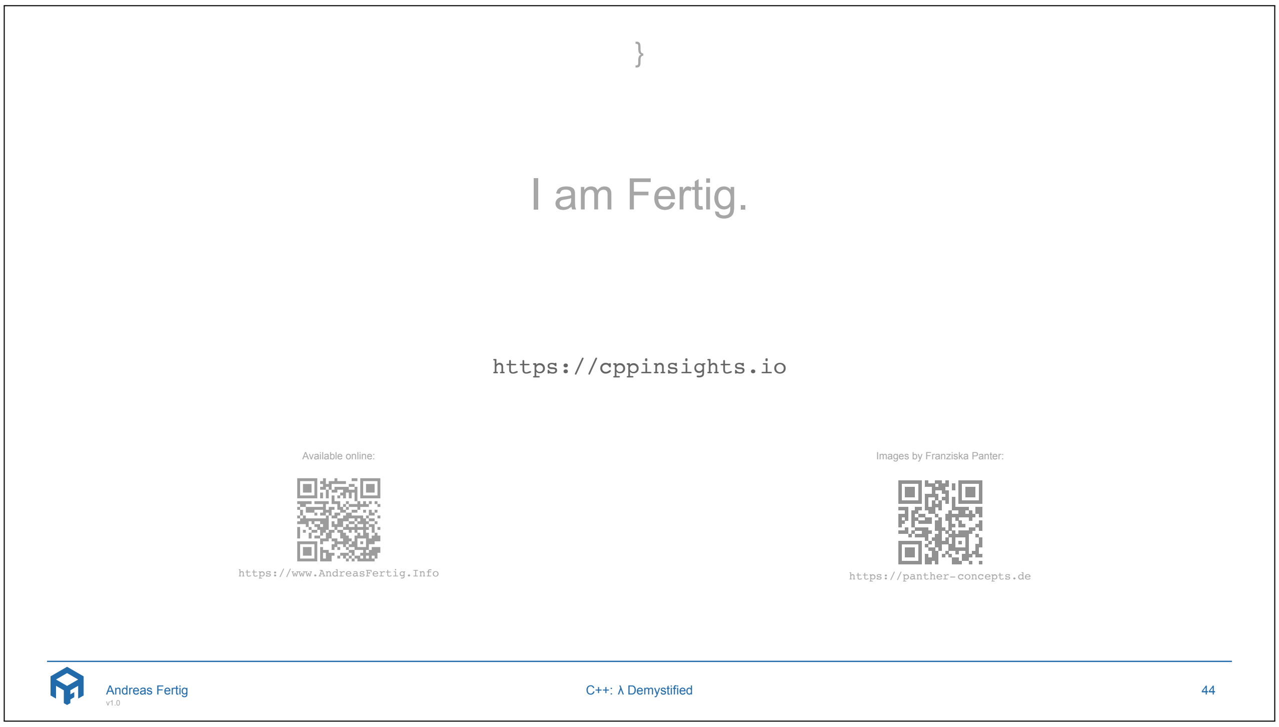
```cpp
const bool isListInitialization{stmt->getLParenLoc().isInvalid()};
```



The Work of Others

- Meeting C++ 2018: Higher Order Functions for ordinary developers - Björn Fahller [5]
- compile-time iteration with C++20 lambdas - Vittorio Romeo [6]
- C++ Weekly - Ep 152 - Lambdas: The Key To Understanding C++ - Jason Turner [7]
- Lambdas: From C++11 to C++20, Part 1 - Bartlomiej Filipek [8]
- ...

Andreas Fertig v1.0 C++: λ Demystified 43



}

I am Fertig.

`https://cppinsights.io`

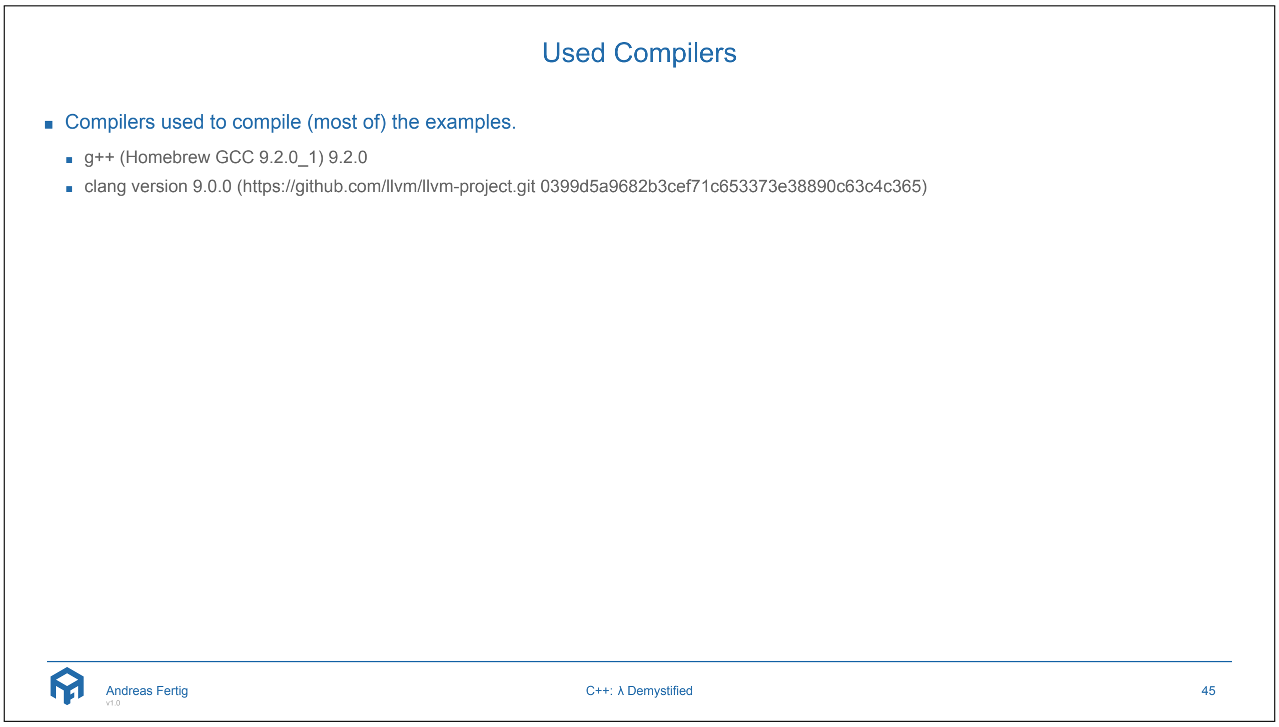
Available online:

https://www.AndreasFertig.Info

Images by Franziska Panter:

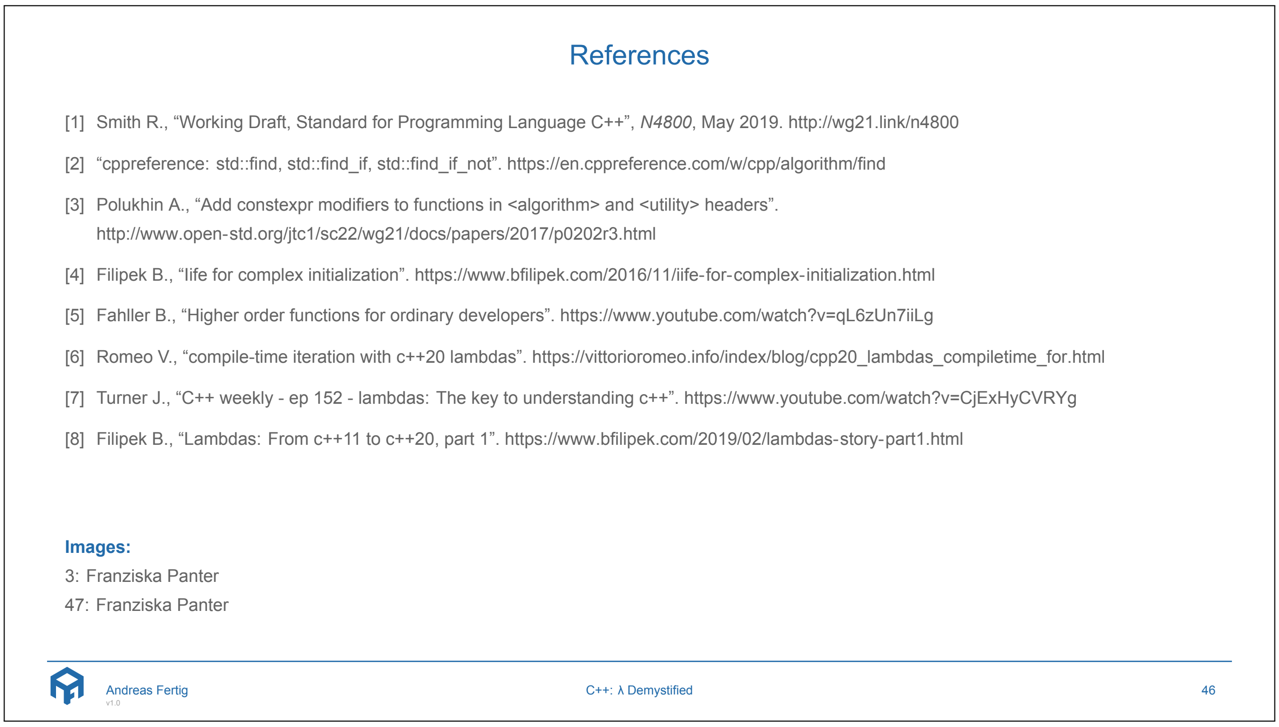
https://panther-concepts.de

Andreas Fertig v1.0 C++: λ Demystified 44



Used Compilers

- Compilers used to compile (most of) the examples.
  - g++ (Homebrew GCC 9.2.0_1) 9.2.0
  - clang version 9.0.0 (https://github.com/llvm/llvm-project.git 0399d5a9682b3cef71c653373e38890c63c4c365)



References

[1] Smith R., “Working Draft, Standard for Programming Language C++”, *N4800*, May 2019. http://wg21.link/n4800

[2] “cppreference: std::find, std::find_if, std::find_if_not”. https://en.cppreference.com/w/cpp/algorithm/find

[3] Polukhin A., “Add constexpr modifiers to functions in <algorithm> and <utility> headers”. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0202r3.html

[4] Filipek B., “Iife for complex initialization”. https://www.bfilipek.com/2016/11/iife-for-complex-initialization.html

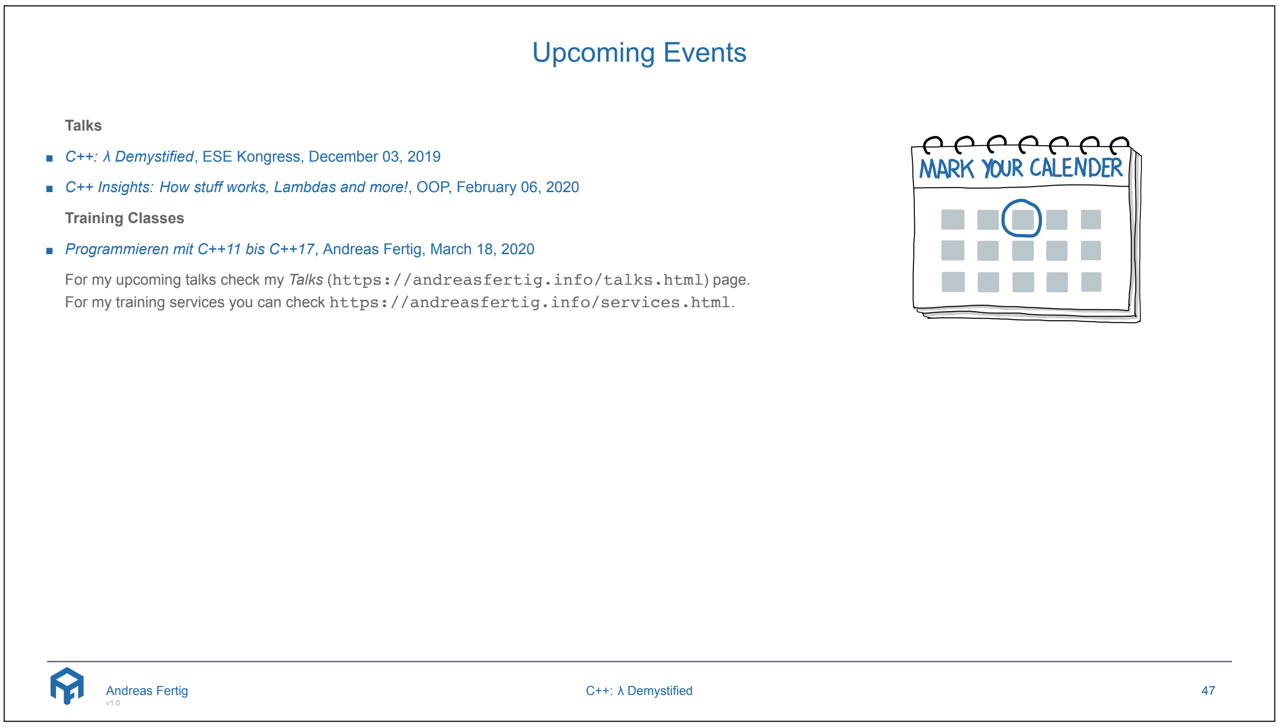
[5] Fahller B., “Higher order functions for ordinary developers”. https://www.youtube.com/watch?v=qL6zUn7iiLg

[6] Romeo V., “compile-time iteration with c++20 lambdas”. https://vittorioromeo.info/index/blog/cpp20_lambdas_compiletime_for.html

[7] Turner J., “C++ weekly - ep 152 - lambdas: The key to understanding c++”. https://www.youtube.com/watch?v=CjExHyCVRYg

[8] Filipek B., “Lambdas: From c++11 to c++20, part 1”. https://www.bfilipek.com/2019/02/lambdas-story-part1.html

Images:
3: Franziska Panter
47: Franziska Panter



Upcoming Events

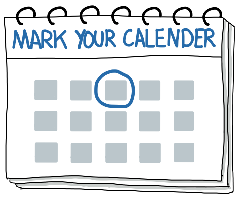
Talks

- *C++: λ Demystified*, ESE Kongress, December 03, 2019
- *C++ Insights: How stuff works, Lambdas and more!*, OOP, February 06, 2020

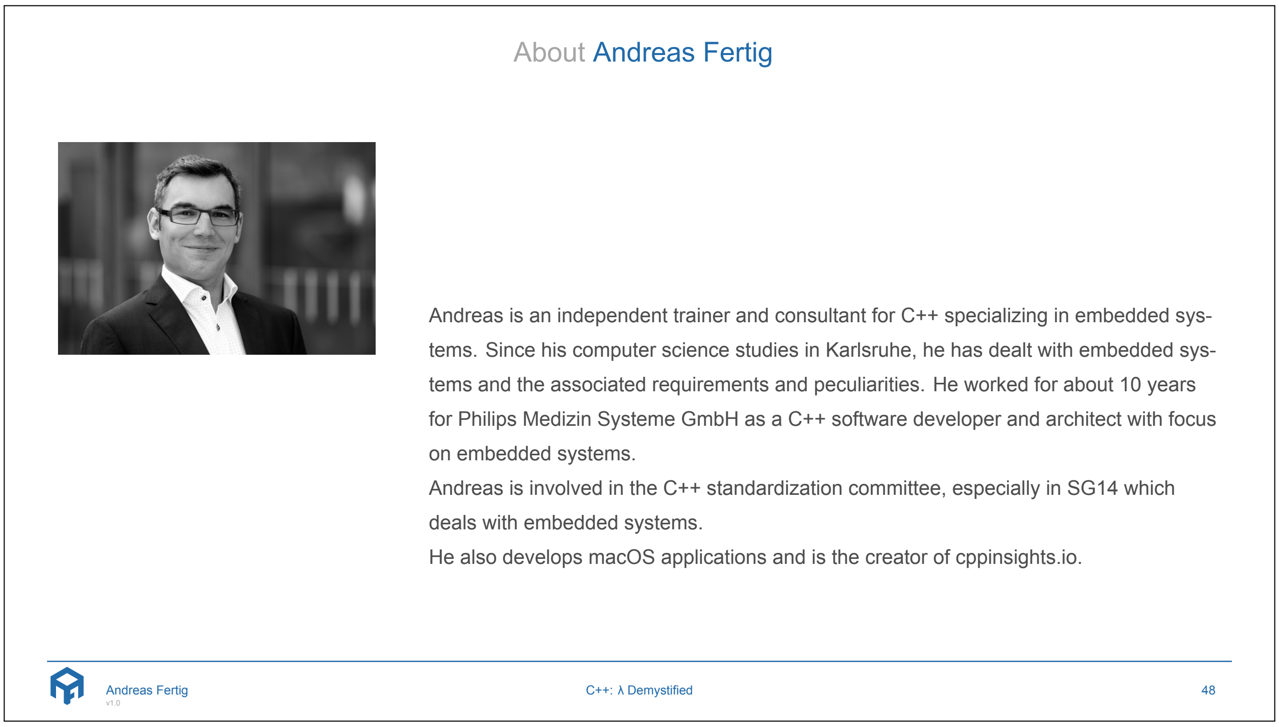
Training Classes

- *Programmieren mit C++11 bis C++17*, Andreas Fertig, March 18, 2020

For my upcoming talks check my *Talks* (`https://andreasfertig.info/talks.html`) page.
For my training services you can check `https://andreasfertig.info/services.html`.



Andreas Fertig v1.0 C++: λ Demystified 47



About Andreas Fertig

Andreas is an independent trainer and consultant for C++ specializing in embedded systems. Since his computer science studies in Karlsruhe, he has dealt with embedded systems and the associated requirements and peculiarities. He worked for about 10 years for Philips Medizin Systeme GmbH as a C++ software developer and architect with focus on embedded systems.
Andreas is involved in the C++ standardization committee, especially in SG14 which deals with embedded systems.
He also develops macOS applications and is the creator of cppinsights.io.

Andreas Fertig v1.0 C++: λ Demystified 48