

# Everything you (n)ever wanted to know about C++'s Lambdas

iCSC 2020

Nis Meinert

Rostock University

# Introduction

# What is a Lambda Expression in C++?

cube is a lambda ...

```
1 int main() {  
2     auto cube = [](int x) { return x * x * x; };  
3     return cube(3);  
4 }
```

[godbolt.org/z/zBE2\\_n](http://godbolt.org/z/zBE2_n)

is\_even is a lambda ...

```
1 #include <algorithm>  
2 #include <vector>  
3  
4 int main() {  
5     std::vector<int> xs{1, 2, 3, 4, 5, 6, 7};  
6     auto is_even = [](int x) { return x % 2 == 0; };  
7     return std::count_if(xs.begin(), xs.end(), is_even);  
8 }
```

[godbolt.org/z/V8Xr4f](http://godbolt.org/z/V8Xr4f)

# Syntax

# C++'s Lambda Expression

The simplest (and most boring) lambda

```
1 auto x = []{};
```

...no capturing, takes no parameters and returns nothing

A slightly more “useful” lambda

```
1 int main() {  
2     auto x = [] { return 5; };  
3     return x();  
4 }
```

[godbolt.org/z/DrnSSE](http://godbolt.org/z/DrnSSE)

...is equivalent to

```
1 int main() {  
2     struct {  
3         auto operator()() const {  
4             return 5;  
5         }  
6     } x;  
7     return x();  
8 }
```

[godbolt.org/z/R8qx3Q](http://godbolt.org/z/R8qx3Q)

## Capturing rules

- `[x]`: captures `x` by value
- `[&x]`: captures `x` by reference
- `[=]`: captures all variables (used in the lambda) by value
- `[&]`: captures all variables (used in the lambda) by reference
- `[=, &x]`: captures variables like with `[=]`, but `x` by reference
- `[&, x]`: captures variables like with `[&]`, but `x` by value

# Capturing by value

```
1 int main() {  
2     int i = 1;  
3     auto z = [i](int y) {  
4         return i + y;  
5     }(3);  
6     return z;  
7 }
```

[godbolt.org/z/FVwarE](http://godbolt.org/z/FVwarE)

...or equivalently

```
1 class X {  
2     private:  
3         int i;  
4  
5     public:  
6         X(int i): i(i) {}  
7  
8         int operator()(int y) const {  
9             return i + y;  
10        }  
11    };  
12  
13    // potentially lots of lines of code  
14  
15    int main() {  
16        int i = 1;  
17        auto z = X{i}(3);  
18        return z;  
19    }
```

[godbolt.org/z/SsRwKV](http://godbolt.org/z/SsRwKV)

# Capturing by reference

```
1 int main() {  
2     int i = 1;  
3     auto z = [&i](int y) {  
4         return i + y;  
5     }(3);  
6     return z;  
7 }
```

[godbolt.org/z/xazquF](http://godbolt.org/z/xazquF)

...or equivalently

```
1 class X {  
2     private:  
3         int& i;  
4  
5     public:  
6         X(int& i): i(i) {}  
7  
8         int operator()(int y) /*const*/ {  
9             return i + y;  
10        }  
11    };  
12  
13    // potentially lots of lines of code  
14  
15    int main() {  
16        int i = 1;  
17        auto z = X{i}(3);  
18        return z;  
19    }
```

[godbolt.org/z/3ycaAW](http://godbolt.org/z/3ycaAW)



## Q: What is the output of the program?

```
1 #include <iostream>
2
3 int main() {
4     int i = 1;
5     auto x = [i]() { return ++i; };
6     std::cout << i << x() << i;
7 }
```

[godbolt.org/z/ZwVDE2](https://godbolt.org/z/ZwVDE2)

## Q: What is the output of the program?

```
1 #include <iostream>
2
3 int main() {
4     int i = 1;
5     auto x = [i]() { return ++i; };
6     std::cout << i << x() << i;
7 }
```

[godbolt.org/z/ZwVDE2](http://godbolt.org/z/ZwVDE2)

**error:** cannot assign to a variable captured by copy in a non-mutable lambda

## Q: What is the output of the program?

```
1 #include <iostream>
2
3 int main() {
4     int i = 1;
5     auto x = [i]() mutable { return ++i; };
6     std::cout << i << x() << i;
7 }
```

[godbolt.org/z/Gs995r](https://godbolt.org/z/Gs995r)

```
1 #include <iostream>
2
3 int main() {
4     int i = 1;
5     auto x = [i]() mutable { return ++i; };
6     std::cout << i << x() << i;
7 }
```

[godbolt.org/z/Gs995r](https://godbolt.org/z/Gs995r)

## Q: What is the output of the program?

```
1 #include <iostream>
2
3 int main() {
4     int i = 1;
5     auto x = [&i]() mutable { return ++i; };
6     std::cout << i << x() << i;
7 }
```

[godbolt.org/z/9mF5rA](https://godbolt.org/z/9mF5rA)

```
1 #include <iostream>
2
3 int main() {
4     int i = 1;
5     auto x = [&i]() mutable { return ++i; };
6     std::cout << i << x() << i;
7 }
```

[godbolt.org/z/9mF5rA](https://godbolt.org/z/9mF5rA)

## Q: What is the output of the program?

```
1 #include <iostream>
2
3 int main() {
4     auto x = [i=0]() mutable { return ++i; };
5     std::cout << x() << x();
6 }
```

[godbolt.org/z/Fdafh9](https://godbolt.org/z/Fdafh9)

```
1 #include <iostream>
2
3 int main() {
4     auto x = [i=0]() mutable { return ++i; };
5     std::cout << x() << x();
6 }
```

[godbolt.org/z/Fdafh9](http://godbolt.org/z/Fdafh9)



## Q: What is the output of the program?

```
1 #include <iostream>
2 #include <utility>
3
4 int main() {
5     auto x = [i=0, j=1]() mutable {
6         i = std::exchange(j, j + i);
7         return i;
8     };
9
10    for (int i = 0; i < 5; ++i) {
11        std::cout << x();
12    }
13 }
```

[godbolt.org/z/eTdadM](http://godbolt.org/z/eTdadM)

[cppreference.com/w/cpp/utility/exchange](http://cppreference.com/w/cpp/utility/exchange)

```
1 #include <iostream>
2 #include <utility>
3
4 int main() {
5     auto x = [i=0, j=1]() mutable {
6         i = std::exchange(j, j + i);
7         return i;
8     };
9
10    for (int i = 0; i < 5; ++i) {
11        std::cout << x();
12    }
13 }
```

[godbolt.org/z/eTdadM](http://godbolt.org/z/eTdadM)

[cppreference.com/w/cpp/utility/exchange](http://cppreference.com/w/cpp/utility/exchange)

# C++'s Lambda Expression

Remember, lambda expressions are pure syntactic sugar and are equivalent to structs with an appropriate `operator()()` overload ...

## Q: What is the output of the program?

```
1 #include <iostream>
2
3 int main() {
4     auto x = [] { return 1; };
5     auto y = x;
6     std::cout << x() << y();
7 }
```

[godbolt.org/z/4tAaV5](https://godbolt.org/z/4tAaV5)

```
1 #include <iostream>
2
3 int main() {
4     auto x = [] { return 1; };
5     auto y = x;
6     std::cout << x() << y();
7 }
```

[godbolt.org/z/4tAaV5](https://godbolt.org/z/4tAaV5)

## Q: What is the output of the program?

```
1  #include <iostream>
2
3  int main() {
4      int i = 1;
5      int j = 2;
6      auto x = [&i, j] { return i + j; };
7      i = 4;
8      j = 6;
9      auto y = x;
10     std::cout << x() << y();
11 }
```

[godbolt.org/z/kpH\\_nT](https://godbolt.org/z/kpH_nT)

```
1  #include <iostream>
2
3  int main() {
4      int i = 1;
5      int j = 2;
6      auto x = [&i, j] { return i + j; };
7      i = 4;
8      j = 6;
9      auto y = x;
10     std::cout << x() << y();
11 }
```

[godbolt.org/z/kpH\\_nT](https://godbolt.org/z/kpH_nT)

## Q: What is the output of the program?

```
1 #include <iostream>
2 #include <memory>
3
4 int main() {
5     auto x = [i=std::make_unique<int>(1)] { return *i; };
6     auto y = x;
7     std::cout << x () << y();
8 }
```

[godbolt.org/z/V37Rmg](https://godbolt.org/z/V37Rmg)



## Q: What is the output of the program?

```
1 #include <iostream>
2 #include <memory>
3
4 int main() {
5     auto x = [i=std::make_unique<int>(1)] { return *i; };
6     auto y = x;
7     std::cout << x () << y();
8 }
```

[godbolt.org/z/V37Rmg](http://godbolt.org/z/V37Rmg)

**error:** call to implicitly-deleted copy ctor

# Stateful Lambdas

## Q: What is the output of the program?

```
1 #include <iostream>
2
3 int main() {
4     auto x = [i=0]() mutable { return ++i; };
5     auto y = x;
6     x();
7     x();
8     y();
9     y();
10    std::cout << x();
11 }
```

[godbolt.org/z/pm7BXk](https://godbolt.org/z/pm7BXk)

```
1 #include <iostream>
2
3 int main() {
4     auto x = [i=0]() mutable { return ++i; };
5     auto y = x;
6     x();
7     x();
8     y();
9     y();
10    std::cout << x();
11 }
```

[godbolt.org/z/pm7BXk](https://godbolt.org/z/pm7BXk)

## Q: What is the output of the program?

```
1 #include <iostream>
2
3 int main() {
4     auto x = [] { static int i = 0; return ++i; };
5     auto y = x;
6     x();
7     x();
8     y();
9     y();
10    std::cout << x();
11 }
```

[godbolt.org/z/XjTDvt](https://godbolt.org/z/XjTDvt)

```
1 #include <iostream>
2
3 int main() {
4     auto x = [] { static int i = 0; return ++i; };
5     auto y = x;
6     x();
7     x();
8     y();
9     y();
10    std::cout << x();
11 }
```

[godbolt.org/z/XjTDvt](https://godbolt.org/z/XjTDvt)

(\* undefined in a threaded context, since `static` is not thread-safe!)

## Fibonacci (again):

```
1 #include <iostream>
2
3 int main() {
4     auto fib = [i=0, j=1]() mutable {
5         struct Result {
6             int &i, &j;
7
8             auto next() {
9                 i = std::exchange(j, j + i);
10                return *this;
11            }
12        };
13        return Result{.i=i, .j=j}.next();
14    };
15
16    fib().next().next().next(); // mutate state
17    return fib().i;
18 }
```

[godbolt.org/z/SJqdtk](https://godbolt.org/z/SJqdtk)

# Stateful Lambdas

Let us now try to interact with the state of the Lambda ...

```
1 #include <utility>
2
3 int main() {
4     auto fib = [i=0, j=1]() mutable {
5         struct Result {
6             int &i, &j;
7
8             auto next() {
9                 i = std::exchange(j, j + i);
10                return *this;
11            }
12        };
13        return Result{.i=i, .j=j}.next();
14    };
15
16    auto r = fib();
17    r.i = 2; // mutate state
18    r.j = 3; // mutate state
19    return fib().j; // 5
20 }
```

[godbolt.org/z/gkE6aa](https://godbolt.org/z/gkE6aa)



# Stateful Lambdas

...or slightly more conveniently:

```
1 #include <utility>
2
3 int main() {
4     auto fib = [i=0, j=1]() mutable {
5         struct Result {
6             int &i, &j;
7
8             auto next(int n = 1) {
9                 while (n-- > 0) {
10                     i = std::exchange(j, j + i);
11                 }
12                 return *this;
13             }
14         };
15         return Result{.i=i, .j=j}.next();
16     };
17
18     return fib().next(3).j; // 5
19 }
```

[godbolt.org/z/2Um\\_9Z](http://godbolt.org/z/2Um_9Z)

# Stateful Lambdas

```
1 #include <utility>
2
3 int main() {
4     auto fib = [i=0, j=1]() mutable {
5         struct Result {
6             int &i, &j;
7
8             auto next(int n = 1) {
9                 while (n-- > 0) {
10                     i = std::exchange(j, j + i);
11                 }
12                 return *this;
13             }
14         };
15         return Result{.i=i, .j=j}.next();
16     };
17
18     return fib().next(10).j; // 144
19 }
```

[godbolt.org/z/S8U--M](https://godbolt.org/z/S8U--M)

```
# g92 -O3
| main:
19|     mov eax, 144
19|     ret
```

[godbolt.org/z/S8U--M](https://godbolt.org/z/S8U--M)

# Best Practices

# Use Lambdas in STL algorithm

```
1 #include <algorithm>
2 #include <vector>
3
4 std::vector<int> get_ints();
5
6 int main() {
7     auto ints = get_ints();
8     auto in_range = [](int x) { return x > 0 && x < 10; };
9     return *std::find_if(ints.begin(), ints.end(), in_range);
10 }
```

[godbolt.org/z/qYp7NU](https://godbolt.org/z/qYp7NU)

# Use Lambdas in STL algorithm

```
1 #include <algorithm>
2 #include <vector>
3
4 std::vector<int> get_ints();
5
6 int main() {
7     auto ints = get_ints();
8     return *std::find_if(ints.begin(), ints.end(),
9                          [](int x) { return x > 0 && x < 10; });
10 }
```

[godbolt.org/z/J7cccJ](http://godbolt.org/z/J7cccJ)

# Stop pollution of namespace with helper variables

```
1 #include <cmath>
2
3 int main() {
4     auto y = [](auto x) {
5         using T = decltype(x);
6         T mean = 1.;
7         T width = 3.;
8         auto norm = 1. / std::sqrt(2. * M_PI);
9         auto arg = (x - mean) / width;
10        return norm * std::exp(-.5 * arg * arg);
11    }(.5);
12
13    return y;
14 }
```

[godbolt.org/z/3-FVEE](http://godbolt.org/z/3-FVEE)

# Allow variables to be const

```
1 #include <vector>
2
3 std::vector<int> get_ints();
4
5 int main() {
6     auto ints = get_ints();
7     const auto sum = [&ints] {
8         int acc = 0;
9         for (auto &x: ints) acc += x;
10        return acc;
11    }();
12
13    return sum;
14 }
```

[godbolt.org/z/B9UDnG](https://godbolt.org/z/B9UDnG)

# Avoid default capture modes

Below, there is a dangling pointer lurking in the wings ...

```
1 void add_filter() {  
2     auto divisor = get_magic_number();  
3     filters.emplace_back([&](int x) { return % divisor == 0; });  
4 }
```

This error becomes more obvious, when explicit capturing is used:

```
1 void add_filter() {  
2     auto divisor = get_magic_number();  
3     filters.emplace_back([&divisor](int x) { return % divisor == 0; });  
4 }
```



# Avoid default capture modes

Mitigation of copy & paste bugs:

```
1 auto divisor = get_magic_number();  
2 std::find_if(container.begin(),  
3             container.end(),  
4             [&divisor](int x) { return x % divisor == 0; });
```

`[&divisor]` indicates that there is an *external* dependency and it is not enough to “just copy” the lambda function if needed elsewhere.

(off-topic: check out this interesting article about copy & paste bugs in real world applications: “The Last Line Effect” by the PVS-Studio team, [www.viva64.com/en/b/0260/](http://www.viva64.com/en/b/0260/))

# Avoid default capture modes

Does the following implementation looks fine?

```
1 struct Widget {  
2     int divisor = 2;  
3  
4     void add_filter() const {  
5         filters.emplace_back(=[](int x) { return x % divisor == 0; });  
6     }  
7 };
```

...given a sufficient implementation of `filters`

# Avoid default capture modes

Does the following implementation looks fine?

```
1 struct Widget {  
2     int divisor = 2;  
3  
4     void add_filter() const {  
5         filters.emplace_back(=[](int x) { return x % divisor == 0; });  
6     }  
7 };
```

...given a sufficient implementation of `filters`

**No! Horrible code!**

Capturing only applies to non-static local variables. Why does this work?

# Avoid default capture modes

Capturing only applies to non-`static` local variables. Why does this work?

```
1 Widget::add_filter() const {  
2     filters.emplace_back(=[](int x) { return x % divisor == 0; });  
3 }
```

...but this fails

```
1 Widget::add_filter() const {  
2     filters.emplace_back([](int x) { return x % divisor == 0; });  
3 }
```

...and this also

```
1 Widget::add_filter() const {  
2     filters.emplace_back([divisor](int x) { return x % divisor == 0; });  
3 }
```

# Avoid default capture modes

There is no local variable `divisor`! But what happens is the following

```
1 Widget::add_filter() const {  
2     filters.emplace_back(=[](int x) {  
3         return x % divisor == 0;  
4     }));  
5 }
```

copies (implicitly) this pointer (until C++17), i.e.

```
1 Widget::add_filter() const {  
2     auto copy_of_this = this;  
3     filters.emplace_back([copy_of_this](int x) {  
4         return x % copy_of_this->divisor == 0;  
5     }));  
6 }
```

...welcome to the world of undefined behavior, when `Widget` goes out of scope!

# Avoid default capture modes

Default capturing by value can be misleading and gives the impression that a lambda is self-contained:

```
1 static auto divisor = 1;  
2 filters.emplace_back([=](int x) { return x % divisor == 0; });  
3 ++divisor;
```

Above, `divisor` is not copied! (as one may have guessed seeing `[=]`)

# Stop using `std::bind`

## Stop using `std::bind`

...and prefer lambda expression, since

- this increases readability,
- lambdas are much more flexible,
- `std::bind` can potentially introduce additional overhead at run-time, whereas lambdas are default `constexpr`

# Stop using `std::function`

## Stop using `std::function`

- `std::function` add multiple copies of passed object (consider using drop-in replacements such as *delegates*<sup>\*</sup>)
- may cause heap allocation
- is just a wrapper ...

...deduce type of lambda via `auto` or template deduction, **if possible** (cf. exercise)

<sup>\*</sup>[codereview.stackexchange.com/questions/14730/impossibly-fast-delegate-in-c11](https://codereview.stackexchange.com/questions/14730/impossibly-fast-delegate-in-c11)



# Inheriting from Lambdas