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

`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

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

…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

C++'s Lambda Expression

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

…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

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

…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

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

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

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

A: 121

```cpp
#include <iostream>

int main() {
    int i = 1;
    auto x = [i]() mutable { return ++i; };
    std::cout << i << x() << i;
}
```

godbolt.org/z/Gs995r

Q: What is the output of the program?

```cpp
#include <iostream>

int main() {
    int i = 1;
    auto x = [&i]() mutable { return ++i; };
    std::cout << i << x() << i;
}
```

godbolt.org/z/9mF5rA

A: 122

```cpp
#include <iostream>

int main() {
    int i = 1;
    auto x = [&i]() mutable { return ++i; };
    std::cout << i << x() << i;
}
```

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

A: 12

```cpp
#include <iostream>

int main() {
    auto x = [i=0]() mutable { return ++i; };
    std::cout << x() << x();
}
```

godbolt.org/z/Fdafh9

Q: What is the output of the program?

```cpp
#include <iostream>
#include <utility>

int main() {
    auto x = [i=0, j=1]() mutable {
        i = std::exchange(j, j + i);
        return i;
    };

    for (int i = 0; i < 5; ++i) {
        std::cout << x();
    }
}
```

godbolt.org/z/eTdadM

(cppreference.com/w/cpp/utility/exchange)

A: 11235

```cpp
#include <iostream>
#include <utility>

int main() {
    auto x = [i=0, j=1]() mutable {
        i = std::exchange(j, j + i);
        return i;
    };

    for (int i = 0; i < 5; ++i) {
        std::cout << x();
    }
}
```

godbolt.org/z/eTdadM

(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

A: 11

```cpp
#include <iostream>

int main() {
    auto x = [] { return 1; };
    auto y = x;
    std::cout << x() << y();
}
```

godbolt.org/z/4tAaV5

Q: What is the output of the program?

```cpp
#include <iostream>

int main() {
    int i = 1;
    int j = 2;
    auto x = [&i, j] { return i + j; };
    i = 4;
    j = 6;
    auto y = x;
    std::cout << x() << y();
}
```

godbolt.org/z/kpH_nT

A: 66

```cpp
#include <iostream>

int main() {
    int i = 1;
    int j = 2;
    auto x = [&i, j] { return i + j; };
    i = 4;
    j = 6;
    auto y = x;
    std::cout << x() << y();
}
```

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

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

error: call to implicitly-deleted copy ctor