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

…or equivalently

```cpp
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

```cpp
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?

```cpp
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?

```cpp
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

```cpp
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

Q: What is the output of the program?

```cpp
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

```
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

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

```
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

Q: What is the output of the program?

```cpp
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

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

```cpp
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

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

C++'s Lambda Expression

Remember, lambda expressions are pure syntactic sugar and are equivalent to `struct`s with an appropriate `operator( )( )` overload …

Q: What is the output of the program?

```cpp
#include <iostream>

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

godbolt.org/z/4tAaV5

```cpp
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

```
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

```cpp
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

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

Q: What is the output of the program?

```cpp
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

```cpp
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

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

```cpp
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

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

Stateful Lambdas

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

Stateful Lambdas

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

```cpp
#include <utility>

int main() {
    auto fib = [i=0, j=1]() mutable {
        struct Result {
            int &i, &j;

            auto next() {
                i = std::exchange(j, j + i);
                return *this;
            }
        };
        return Result{.i=i, .j=j}.next();
    };

    auto r = fib();
    r.i = 2; // mutate state
    r.j = 3; // mutate state
    return fib().j; // 5
}
```

godbolt.org/z/gkE6aa

Stateful Lambdas

…or slightly more conveniently:

```cpp
#include <utility>

int main() {
    auto fib = [i=0, j=1]() mutable {
        struct Result {
            int &i, &j;

            auto next(int n = 1) {
                while (n-- > 0) {
                    i = std::exchange(j, j + i);
                }
                return *this;
            }
        };
        return Result{.i=i, .j=j}.next();
    };

    return fib().next(3).j; // 5
}
```

godbolt.org/z/2Um_9Z

Stateful Lambdas

```cpp
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  }
```

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

godbolt.org/z/S8U--M

godbolt.org/z/S8U--M

Use Lambdas in STL algorithm

```cpp
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

```cpp
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

Stop pollution of namespace with helper variables

```cpp
#include <cmath>

int main() {
    auto y = [](auto x) {
        using T = decltype(x);
        T mean = 1.;
        T width = 3.;
        auto norm = 1. / std::sqrt(2. * M_PI);
        auto arg = (x - mean) / width;
        return norm * std::exp(-.5 * arg * arg);
    }(.5);

    return y;
}
```

godbolt.org/z/3-FVEE

Allow variables to be `const`

```cpp
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