C++: λ Demystified

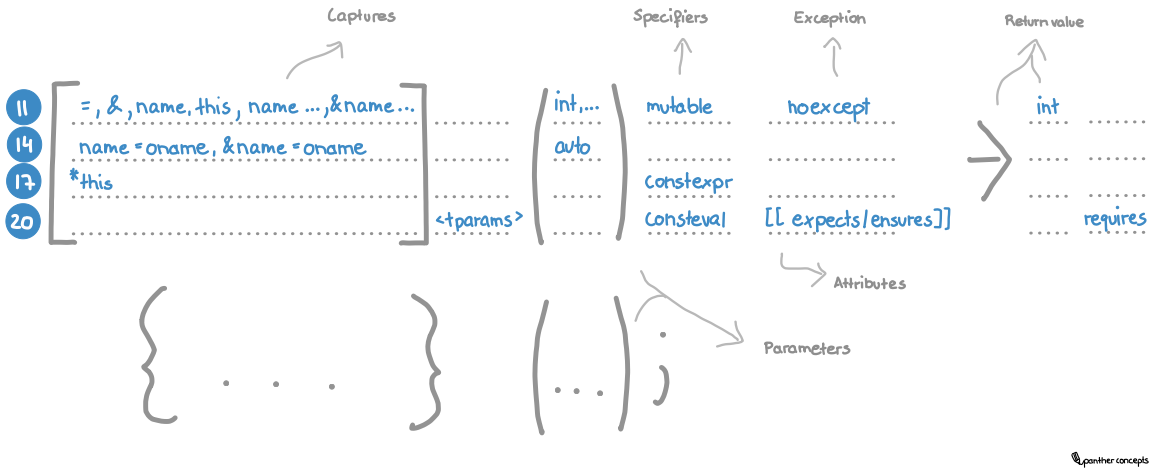
Andreas Fertig
https://www.AndreasFertig.Info
post@AndreasFertig.Info
@Andreas__Fertig

fertig

adjective /ˈfɛrtɪç/

finished
ready
complete
completed

Lambda Evolution



Valid or Not?

```
1 int main()
2 {
3   [] {} = {};
4 }
```

Lambda Internals

```
1 int main()
2 {
3   const char hello[]{"Hello NDC Oslo"};
4
5   [&] { printf("%s\n", hello); }();
6 }
```

Lambdas can appear everywhere...

```
1 int main()
2 {
3   int x = 5;
4   for(;
5       --x;
6       ) {
7     printf("x: %d\n", x);
8   }
9 }
```

Lambdas can appear everywhere...

```
1 int main()
2 {
3   int x = [] { return 5; }();
4   for([] { printf("started\n"); }();
5       [&] { return --x; }();
6       [] { printf("after\n"); }()) {
7     [&] { printf("x: %d\n", x); }();
8   }
9 }
```


Capturing Global Variables

```
1 int x{1};
2
3 int main()
4 {
5   [] { ++x; }();
6 }
```

Captureless Lambda and Function Pointer

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

Size of a Lambda

```
 1 int main()
 2 {
 3   char a{2};
 4   int  b{1};
 5   char c{2};
 6
 7   auto f = [=] {
 8     a;
 9     b;
10     c;
11   };
12 }
```

Assume a x64 Platform.

https://www.AndreasFertig.Info
post@AndreasFertig.Info

Size of a Lambda

```
 1 int main()
 2 {
 3   char a{2};
 4   int  b{1};
 5   char c{2};
 6
 7   auto s = [=] {
 8     a;
 9     c;
10     b;
11   };
12 }
```

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]

Generic Lambda

C++14

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

```
1 auto l = []( auto  v) { return v * 2; };
2
3 auto d = l(2.0);
4 auto i = l(2);
```

Generic Lambda

C++14

- Have a call operator which is a operator template with return type `auto`.
- The `auto` parameters are template parameters.
- In combination with C++17's constexpr if we can have Lambdas with multiple return types.

```
1  auto l = [](auto v) {
2    if constexpr(std::is_same_v<decltype(v), double>) {
3      return v * 2.0;
4    } else {
5      return v * 2;
6    }
7  };
8
9  auto d = l(2.0);
10 auto i = l(2);
```

The Dangling Reference Trap

- Is that innocent looking lambda okay?

```
1  auto Func()
2  {
3    int x{22};
4
5    auto l = [&] { return x * x; };
6
7    // a bunch of code follows.
8
9    return l;
10 }
```


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.

```
1  auto Func()
2  {
3    int x{22};
4
5    auto l = [ = ] { return x * x; };
6
7    // a bunch of code follows.
8
9    return l;
10 }
```

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.

```
1  auto Func()
2  {
3    int* x = new int{22};
4
5    auto l = [=] { return (*x) * (*x); };
6
7    // a bunch of code follows.
8
9    return l;
10 }
```


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...

```
1  auto Func()
2  {
3    int* x = new int{22};
4
5    auto l = [=] { return (*x) * (*x); };
6
7    // a bunch of code follows.
8    // and in the middle of that code:
9    delete x;
10
11   return l;
12 }
```

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.

```
1  auto Func()
2  {
3    shared_ptr<int> x = make_shared<int>(22);
4
5    auto l = [=] { return (*x) * (*x); };
6
7    // a bunch of code follows.
8
9    return l;
10 }
```

Captures

```
1  class Test
2  {
3  public:
4    Test(int x)
5    : a{x}
6    {
7      auto l1 = [=] { return a + 2; };
8
9      printf("l1: %d\n", l1());
10
11     ++a;
12
13     printf("l1: %d\n", l1());
14   }
15
16   int a;
17 };
18
19 int main()
20 {
21   Test t{2};
22 }
```

Captures

```
1  class Test
2  {
3  public:
4    Test(int x)
5    : a{x}
6    {
7      auto l1 = [=] { return a + 2; };
8
9      printf("l1: %d\n", l1());
10
11     ++a;
12
13     printf("l1: %d\n", l1());
14   }
15
16   int a;
17 };
18
19 int main()
20 {
21   Test t{2};
22 }
```

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

Captures

C++17

```
1  class Test
2  {
3  public:
4    Test(int x)
5    : a{x}
6    {
7      auto l1 = [ * this ] { return a + 2; };
8
9      printf("l1: %d\n", l1());
10
11     ++a;
12
13     printf("l1: %d\n", l1());
14   }
15
16   int a;
17 };
18
19 int main()
20 {
21   Test t{2};
22 }
```

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

Captures

C++17

```
1  class Test
2  {
3  public:
4    Test(int x)
5    : a{x}
6    {
7      auto l2 = [*this] { return a + 2; };
8    }
9
10   int a;
11   int b;
12 };
```

Captures

C++14

```
1  class Test
2  {
3  public:
4    Test(int x)
5    : a{x}
6    {
7      auto l2 = [al = a] { return al + 2; };
8    }
9
10   int a;
11   int b;
12 };
```

Size of a Lambda - Part II

```
1  class Test
2  {
3  public:
4    Test(int x)
5    : a{x}
6    {
7      const int size = 2;
8
9      auto l2 = [=] {
10       int x[size]{};
11
12       return a + 2;
13     };
14   }
15
16   int a;
17 };
```

Sleeping Lambda

```
1  int main()
2  {
3    std::string foo;
4
5    auto a = [=]    () { printf( "%s\n", foo.c_str()); };
6
7    auto b = [=]    () { };
8
9    auto c = [foo]  () { printf( "%s\n", foo.c_str()); };
10
11   auto d = [foo]  () { };
12
13   auto e = [&foo] () { printf( "%s\n", foo.c_str()); };
14
15   auto f = [&foo] () { };
16 }
```

Assume a x64 Platform.

constexpr Lambdas

C++17

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

```
1  template<class InputIt, class UnaryPredicate>
2  constexpr InputIt find_if_not(InputIt first, InputIt last, UnaryPredicate q)
3  {
4    for(; first != last; ++first) {
5      if(!q(*first)) {
6        return first;
7      }
8    }
9    return last;
10 }
11
12 template<class InputIt, class UnaryPredicate>
13 constexpr bool all_of(InputIt first, InputIt last, UnaryPredicate p)
14 {
15   return find_if_not(first, last, p) == last;
16 }
17
18 int main()
19 {
20   constexpr int  ar[5]{1, 3, 5, 7, 9};
21   constexpr bool allEven =
22     all_of(&ar[0], &ar[5], [](int i) { return (i % 2) == 0; });
23
24   return allEven;
25 }
```

constexpr Lambdas

C++20

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

```
1  #include <algorithm>
2  #include <array>
3
4  int main()
5  {
6    constexpr std::array<int, 5> ar{1, 3, 5, 7, 9};
7    constexpr bool               allEven =
8      std::all_of(ar.begin(), ar.end(), [](int i) { return (i % 2) == 0; });
9
10   return allEven;
11 }
```

Lambdas Applied

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

```
1  template<typename T>
2  void CodeGenerator::WrapInParensOrCurlys(const BraceKind        braceKind,
3                                           T&&                    lambda,
4                                           const AddSpaceAtTheEnd addSpaceAtTheEnd)
5  {
6    if(BraceKind::Curlys == braceKind) {
7      mOutputFormatHelper.Append('{');
8    } else {
9      mOutputFormatHelper.Append('(');
10   }
11
12   lambda();
13
14   if(BraceKind::Curlys == braceKind)
15   {
16     mOutputFormatHelper.Append('}');
17   }
18   else { mOutputFormatHelper.Append(')'); }
19
20   if(AddSpaceAtTheEnd::Yes == addSpaceAtTheEnd) {
21     mOutputFormatHelper.Append(' ');
22   }
23 }
```

From C++ Insights.

Lambdas Applied

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

```
1  template<typename T, typename Lambda>
2  static inline void ForEachArg(const T&            arguments,
3                                OutputFormatHelper& outputFormatHelper,
4                                Lambda&&            lambda)
5  {
6    OnceFalse needsComma{};
7
8    for(const auto& arg : arguments) {
9      if(needsComma) {
10       outputFormatHelper.Append(", ");
11     }
12
13     lambda(arg);
14   }
15 }
```

From C++ Insights.

Lambdas Applied

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

```
1  const auto name = [&]() -> std::string {
2    // Handle a special case where we have a lambda
3    // static invoke operator. In that case use the
4    // appropriate using retType as return type
5    if(const auto* m = dyn_cast_or_null<CXXMethodDecl>(meDecl)) {
6      if(const auto* rd = m->getParent(); rd && rd->isLambda()) {
7        skipTemplateArgs = true;
8
9        return StrCat("operator ", GetLambdaName(*rd), "::", BuildRetTypeName(*rd));
10     }
11   }
12
13   return stmt->getMemberNameInfo().getName().getAsString();
14 }();
```

From C++ Insights.

Lambdas Applied

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

```
1  size_t ReadData(span<char> buffer)
2  {
3    int fd = Open(/* some well known file*/);
4
5    if-(1 == fd) {
6      return 0;
7    }
8
9    const auto len =
10     read(fd, buffer.data(), buffer.size());
11
12   if-(1 == len) {
13     return 0;
14   }
15
16   ftruncate(fd, len);
17
18   close(fd);
19
20   return gsl::narrow_cast<size_t>(len);
21 }
```

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);
}
```


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::move(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);
}
```

Lambda capture pack expansion and use move

C++20

```
1  template<typename... Args>
2  void foo(Args&&... args)
3  {
4    (..., (std::cout << args));
5  }
6
7  template<class... Args>
8  auto InvokeLater(Args&&... args)
9  {
10   return [... margs = std::forward<Args>(args)] { return foo(margs...); };
11 }
12
13 int main()
14 {
15   auto il = InvokeLater("Hello"s, " "s, "World"s);
16   il();
17 }
```

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

Templated Lambdas

C++20

```
1  int main()
2  {
3    auto max = [](auto x, auto y) {
4      return (x > y) ? x : y;
5    };
6
7    max(2, 3);    // ok
8    max(2, 3.0);  // not wanted
9  }
```

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

Templated Lambdas

C++20

```
 1 int main()
 2 {
 3   auto max = []<typename T>(T x, T y)
 4   {
 5     return (x > y) ? x : y;
 6   };
 7
 8   max(2, 3);  // ok
 9   // max(2, 3.0);  // does not compile anymore
10 }
```

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

Templated Lambdas

C++20

```
1 auto lambda = []<typename T>(std::vector<T> t){};
2 std::vector<int> v{};
3
4 lambda(v);
5 // lambda(20);
```

```
 1 #include <array>
 2
 3 int main()
 4 {
 5   auto l = []<size_t N>(std::array<int, N> x) {};
 6
 7   std::array<int, 2> a{};
 8
 9   l(a);
10 }
```

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

Templated Lambdas

C++20

Do it with C++14:

```
1  #include <array>
2
3  int main()
4  {
5    auto l = [](std::array<auto, 2> x) {};
6
7    std::array<int, 2> a{};
8
9    l(a);
10 }
```

Does it compile?

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

Default Constructible Lambdas & `decltype`

C++20

C++14 version:

```
1 auto compare = [](auto x, auto y) { return x > y; };
2 std::map<std::string, int, decltype(compare)> map{{"a", 1}, {"b", 2}};
3
4 for(const auto& [v, k] : map) {
5   printf("%s\n", v.c_str());
6 }
```

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

Default Constructible Lambdas & `decltype`

C++20

```
1 std::map<std::string, int, decltype([](auto x, auto y) { return x > y; })> map{
2   {"a", 1},
3   {"b", 2}};
4
5 for(const auto& [v, k] : map) {
6   printf("%s\n", v.c_str());
7 }
```

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

Lambda Overuse

- Can we have an overuse of lambdas?

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

Lambda Overuse

- Can we have an overuse of lambdas?

```
1 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]
- ...

}

I am Fertig.

https://cppinsights.io

Available online:

https://www.AndreasFertig.Info

Images by Franziska Panter:

https://panther-concepts.de

Used Compilers

- Compilers used to compile (most of) the examples.
  - g++ (GCC) 9.0.1 20190421 (experimental)
  - clang version 9.0.0 (https://github.com/llvm-mirror/clang.git 6fddf7789c74ae74d695dd571024915ad319db1b) (https://github.com/llvm-mirror/llvm.git 92ce94fff3978cb5f48d14f2e0ac4fe0035ea7c5)

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., “life 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:

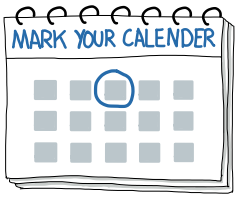
3: Franziska Panter

48: Franziska Panter

Upcoming Events

- *C++: λ Demystified*, NDC { Oslo }, June 17 2019
- *C++: λ Demystified*, NDC TechTown, September 04 2019
- *C++ Insights: See your source code with the eyes of a compiler*, NDC { TechTown }, September 05 2019
- *C++1x für eingebettete Systeme kompakt*, Seminar QA Systems, November 14 2018
- *C++ Templates - die richtige Dosis kompakt*, Seminar QA Systems, November 15 2018

To keep in the loop, periodically check my *Talks and Training* (`https://andreasfertig.info/talks.html`) page.



About Andreas Fertig

Andreas is a freelance 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 deals.
He also develops macOS applications and is the author of cppinsights.io.