- Toolchain for C++17
  - compilers
    - clang 6.0 or newer
    - gcc 7.0 or newer
    - MSVC 2017 15.8.1 or newer
    - XCode 10.0 (requires OS X 10.13 or newer)
  - CMake
    - 3.8 or newer
  - Before you upgrade, verify with the vendor of any third party library, even boost, that it works with C++17

- Interview Questions / Unique Pointers
  - What is std::unique_ptr?
  - What std containers can store a std::unique_ptr?
  - How do you capture std::unique_ptr in a lambda expression?

- Interview Questions / Unique Pointers
  - What is std::unique_ptr?
    - std::unique_ptr is a replacement for std::auto_ptr which was removed in C++17
    - used to manage a resource or a heap allocated object
    - must own the object it points to
    - it uses move semantics, it can not be copied or assigned
    - std::make_unique added in C++14
      - templated function, creates an object and returns a unique_ptr
      - `std::make_unique<std::string>("Hello C++17");`

- Interview Question / Unique Pointers
  - What std containers can store a std::unique_ptr?
    - container must support elements which are “move only”
    - if the container requires copying elements (like during a resize) it can not store a unique_ptr
    - when a unique_ptr is moved into a container the original ptr transfers ownership to the container
    - most every std container supports move only types, however not every operation can be called
      - `operator=(), assign(), copy constructor, etc`

- Interview Question / Unique Pointers
  - How do you capture std::unique_ptr in a lambda expression?
    - use a generalized lambda capture (from C++14) to “move capture”
    - capture by value requires a copy

```
std::unique_ptr<int> myPtr = std::make_unique<int>(17);
auto lamb = [ capturedPtr = std::move(myPtr) ] ( )
               { return *capturedPtr + 8; };

std::cout << lamb();
```

- Generic Lambdas
  - added in C++14
  - data type for at least one parameter must be auto
  - not every parameter needs to be generic

  - generated “code” results in an operator() which is a template
  - C++20 adds the ability to use the template syntax or auto (or mixed) when defining lambda expressions 

```
auto lamb = [ ] (auto var1, int var2) { return var1 + var2; }
```

- Bonus Round: Is “function template argument deduction” equivalent to “auto type deduction”? Which one is yellow, which one is purple?

- Interview Questions
  - Which ones are valid code, duplicates, or meaningless?

1. const
2. constexpr
3. constexpr const
4. static constexpr const
5. constexpr static const
6. if constexpr
7. constexpr if

- Interview Questions
  - Which ones are valid code, duplicates, or meaningless?
    - const
      - part of the data type
      - some code promises not to change something
      - for a pointer or method
        - location of the keyword const yields a different result
  - What is the worst answer you can give to the following question, “What does const mean?”

- Interview Questions
  - Which ones are valid code, duplicates, or meaningless?
    - constexpr
      - part of the declaration but not part of a data type
      - asks the compiler to evaluate and run code at compile time
      - compiler is not always required to do the work at compile time
      - usually it will, however there is no guarantee or way to check
    - constexpr const
      - For a constexpr method, what changed in C++14?

- Interview Questions
  - For a constexpr method, what changed in C++14?
    - a constexpr method was *always* const qualified in C++11
    - with C++14 this restriction was removed
  - Is “constexpr” the same as a “constant expression”?
    - no they are not

- Interview Questions
  - Which ones are valid code, duplicates, or meaningless?
    - static constexpr const
      - both of these are identical after compiling
    - constexpr static const
      - our preferred ordering
      - returns a const reference to an int
      - static means the method can be called without an object

```
constexpr static const int & someMethod(int var);
```

- Interview Questions
  - Which ones are valid code, duplicates, or meaningless?
    - if constexpr (condition)
      - the condition must be something which can be determined at compile time
      - only one branch is taken, others are discarded at compile time
      - Does the discarded code need to compile?
    - constexpr if
      - original proposal contained this wording, rejected
      - not valid code

- Constexpr Lambda Expression
  - C++14
    - not allowed to have constexpr lambda expression
  -
  - C++17
    - implicitly qualified with constexpr if the body *and* capture clause meet the requirements for constexpr
      - capture list must only contain “literal data types”
    - you can add constexpr however it is meaningless since this will be deduced by the compiler

- Interview Questions / Constexpr Lambda Expression
  - What is a literal data type?
  - Are there good use cases for a constexpr lambda expressions?

- Interview Questions / Constexpr Lambda Expression
  - What is a literal data type?
    - a class or struct which has a trivial destructor
    - has a constructor which can be called at compile time
    - data type which does not require allocation (C++17)
    - important definition to understand since constexpr variables must be literal data types
    - constexpr functions can only work with literal data types

- Interview Questions / Constexpr Lambda Expression
  - Are there good use cases for a constexpr lambda expression?
    - it was inconsistent that lambda expressions were restricted whereas a function object could be constexpr
    - algorithms which take a predicate
    - C++20 is adding constexpr algorithms and some containers
    - constexpr variable initialized from a lambda expression

```
auto sum = [ ](int data)
  { return 5 + data; };

constexpr int result = sum(10);
```

- Interview Questions
  - What is type safety?
  - Is a void * type safe?
  - Is a union type safe?

- Interview Questions
  - What is type safety?
    - computer science states that type safety is the degree to which a computer language discourages or prevents data type errors
    - in C++ it is your responsibility to ensure type safety
    - the appropriate data type must be used for each operation
    - if an operation is type safe then data declared with one type can never hold a value of a different data type

- Interview Questions
  - Is a void * type safe?
    - yes
    - comparison and assignment are really the only operations allowed on a variable declared with the type void *
    - surprisingly these operations are type safe so a program which uses a void * is not automatically unsafe
  - Is a union type safe?
    - no
    - all members of the union share the same memory location
    - only one member of a union can be accessed at a time
    - reading any other member is undefined behavior

- Variant, Visit, Any, Optional
  - std::variant
    - type safe replacement for a union
    - container which holds a single valid alternative
  - std::visit
    - templated function that retrieves the current alternative from an std::variant and then passes it to a “visitor” (function)
  - std::any
    - a container which can store a single value of any data type
  - std::optional
    - a container that stores a single value of the specified T or is empty

- C++17
  - structured bindings
  - fold expressions
  - class template argument deduction

  - std::byte
  - std::string_view
  - [[fallthrough]] attribute

  - inline variables
    - similar to inline functions
  - generalizing range-based for() loops
    - begin and end iterators with different types

❑ Type Traits	April 2018
❑ Lambdas in C++	May 2018
❑ Constexpr Static Const	September 2018
❑ When Your Codebase is Old Enough to Vote	October 2018
❑ C++ ISO Standard	June 2019
❑ Lambdas in Action	August 2019
❑ Any, Optional	September 2019
❑ std::variant	October 2019
❑ std::visit	January 2020

https://www.youtube.com/copperspice

- Videos
  - CppCon 2017: Bryce Adelstein Lelbach “C++17 Features (part 1)”
  - https://www.youtube.com/watch?v=fI2xiUqqH3Q

  - CppCon 2017: Bryce Adelstein Lelbach “C++17 Features (part 2)”
  - https://www.youtube.com/watch?v=qjxBKINAWk0

  - CppCon 2018: Stephen T Lavavej “Class Template Argument Deduction”
  - https://www.youtube.com/watch?v=-H-ut6j1BYU

Where to find CopperSpice

- www.copperspice.com

- ansel@copperspice.com
- barbara@copperspice.com

- source, binaries, documentation files
  - download.copperspice.com

- source code repository
  - github.com/copperspice

- discussion
  - forum.copperspice.com