C++ now

What Does it Take to Implement the Standard Library?

Christopher Di Bella

2024

You need to be able to turn this...

```
template<class... Args1, class... Args2>
constexpr pair(piecewise_construct_t,
               tuple<Args1...> first_args, tuple<Args2...> second_args
```

18 *Mandates:*

(18.1) — `is_constructible_v<T1, Args1...>` is `true` and

(18.2) — `is_constructible_v<T2, Args2...>` is `true`

19 *Effects*: Initializes `first` with arguments of types `Args1...` obtained by forwarding the elements of `first_args` and initializes `second` with arguments of types `Args2...` obtained by forwarding the elements of `second_args`. (Here, forwarding an element `x` of type `U` within a `tuple` object means calling `std::forward<U>(x)`.) This form of construction, whereby constructor arguments for `first` and `second` are each provided in a separate `tuple` object, is called *piecewise construction*.

[*Note 2*: If a data member of `pair` is of reference type and its initialization binds it to a temporary object, the program is ill-formed ([class.base.init]).— *end note*]

...into this

```
template<class... Args1, class... Args2>
constexpr pair(piecewise_construct_t,
               tuple<Args1...> args1,
               tuple<Args2...> args2)

  noexcept(is_nothrow_constructible_v<T1, Args1...> and
           is_nothrow_constructible_v<T2, Args2...>)

: first(std::make_from_tuple<T1>(std::move(args1)))
, second(std::make_from_tuple<T2>(std::move(args2)))
{
  static_assert(is_constructible_v<T1, Args1...> and
                is_constructible_v<T2, Args2...>);
}
```

Correctness

(includes reliability, safety, and security)

Good performance

Some things we want as developers

Good diagnostics

Correctness

(includes reliability, safety, and security)

```
1 template<class T, class Alloc = allocator<T>>
2 class vector {
3 public:
4     constexpr int empty() const noexcept
5             // ^ wrong return type
6     {
7         return size() != 0; // wrong return value
8     }
9 };
```

Correctness

(includes reliability, safety, and security)

```
1 auto const v = std::vector{0, 1, 2};
2 return v[4]; // out-of-bounds access
```

Good performance

```
auto const std_vec  = std::vector<int>{0, 1, 2, /* ... */, 1'000};
auto const your_vec = your::vector<int>{0, 1, 2, /* ... */, 1'000};

auto const needle = get_random<int>(0, std_vec.size());

std::ranges::contains(std_vec, needle);  // (1)
std::ranges::contains(your_vec, needle); // (2)
```

(1) should be no slower than (2)

Good diagnostics

```
1 struct S {};
2 S s;
3 std::print("{}", s);
```

How you should feel when you get a diagnostic

```
In file included from source.cpp:1:
In file included from /usr/bin/include/c++/v1/print:41:
In file included from /usr/bin/include/c++/v1/format:202:
/usr/bin/include/c++/v1/__format/format_functions.h:98:30: error: call to deleted constructor of 'formatter<S, char>'
  98 |       formatter<_Tp, _CharT> __f;
     |                              ^
/usr/bin/include/c++/v1/__format/format_functions.h:97:62: note: while substituting into a lambda expression here
  97 |     __parse_ = [](basic_format_parse_context<_CharT>& __ctx) {
     |                                                              ^
/usr/bin/include/c++/v1/__format/format_functions.h:392:25: note: in instantiation of function template specialization 'std::__format::__compile_time_handle<char>::__enable<S>' requested here
 392 |       __handle.template __enable<_Tp>();
     |                         ^
/usr/bin/include/c++/v1/__format/format_functions.h:388:99: note: while substituting into a lambda expression here
 388 |   static constexpr array<__format::__compile_time_handle<_CharT>, sizeof...(_Args)> __handles_{[] {
     |                                                                                                   ^
/usr/bin/include/c++/v1/__format/format_functions.h:372:54: note: in instantiation of static data member 'std::basic_format_string<char, S &>::__handles_' requested here
 372 |                             _Context{__types_.data(), __handles_.data(), sizeof...(_Args)});
     |                                                      ^
source.cpp:8:16: note: in instantiation of function template specialization 'std::basic_format_string<char, S &>::basic_format_string<char[3]>' requested here
   8 |     std::print("{}", s);
     |                ^
/usr/bin/include/c++/v1/__format/formatter.h:36:3: note: 'formatter' has been explicitly marked deleted here
  36 |   formatter()                            = delete;
     |   ^
In file included from source.cpp:1:
In file included from /usr/bin/include/c++/v1/print:41:
In file included from /usr/bin/include/c++/v1/format:202:
/usr/bin/include/c++/v1/__format/format_functions.h:99:28: error: no member named 'parse' in 'std::formatter<S>'
  99 |       __ctx.advance_to(__f.parse(__ctx));
     |                        ~~~ ^
source.cpp:8:16: error: call to consteval function 'std::basic_format_string<char, S &>::basic_format_string<char[3]>' is not a constant expression
   8 |     std::print("{}", s);
     |                ^
/usr/bin/include/c++/v1/__format/format_functions.h:271:7: note: non-constexpr function '__throw_format_error' cannot be used in a constant expression
 271 |       std::__throw_format_error("The argument index value is too large for the number of arguments supplied");
     |       ^
/usr/bin/include/c++/v1/__format/format_functions.h:316:20: note: in call to '__handle_replacement_field<const char *, std::basic_format_parse_context<char>, std::__format::__compile_time_basic_format_context<char>>(&"{}"[1], &"{}"[2], bas
 316 |         __begin  = __format::__handle_replacement_field(__begin, __end, __parse_ctx, __ctx);
     |                    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/bin/include/c++/v1/__format/format_functions.h:371:5: note: in call to '__vformat_to<std::basic_format_parse_context<char>, std::__format::__compile_time_basic_format_context<char>>(basic_format_parse_context<char>{this->__str_, sized
 371 |     __format::__vformat_to(basic_format_parse_context<_CharT>{__str_, sizeof...(_Args)},
     |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 372 |                             _Context{__types_.data(), __handles_.data(), sizeof...(_Args)});
     |                             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
source.cpp:8:16: note: in call to 'basic_format_string<char[3]>("{}")'
   8 |     std::print("{}", s);
     |                ^~~~
/usr/bin/include/c++/v1/__format/format_error.h:38:52: note: declared here
  38 | _LIBCPP_NORETURN inline _LIBCPP_HIDE_FROM_ABI void __throw_format_error(const char* __s) {
     |                                                    ^
In file included from source.cpp:1:
In file included from /usr/bin/include/c++/v1/print:41:
In file included from /usr/bin/include/c++/v1/format:195:
In file included from /usr/bin/include/c++/v1/__format/container_adaptor.h:20:
In file included from /usr/bin/include/c++/v1/__format/range_default_formatter.h:23:
In file included from /usr/bin/include/c++/v1/__format/range_formatter.h:23:
In file included from /usr/bin/include/c++/v1/__format/format_context.h:18:
/usr/bin/include/c++/v1/__format/format_arg_store.h:167:17: error: static assertion failed due to requirement '__arg != __arg_t::__none': the supplied type is not formattable
 167 |   static_assert(__arg != __arg_t::__none, "the supplied type is not formattable");
     |                 ^~~~~~~~~~~~~~~~~~~~~~~
/usr/bin/include/c++/v1/__format/format_arg_store.h:214:54: note: in instantiation of function template specialization 'std::__format::__create_format_arg<std::format_context, S>' requested here
 214 |         basic_format_arg<_Context> __arg = __format::__create_format_arg<_Context>(__args);
     |                                                      ^
/usr/bin/include/c++/v1/__format/format_arg_store.h:249:19: note: in instantiation of function template specialization 'std::__format::__create_packed_storage<std::format_context, S>' requested here
```

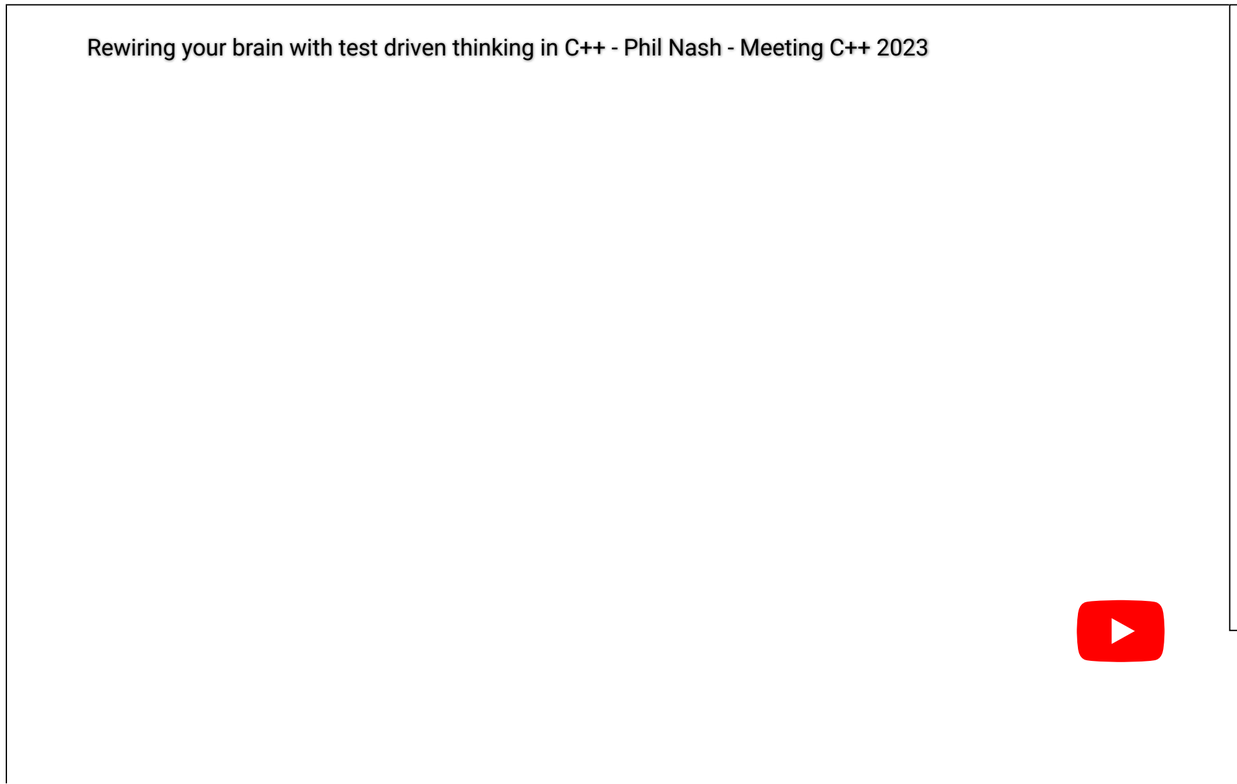
How do we achieve these goals?

ACT I

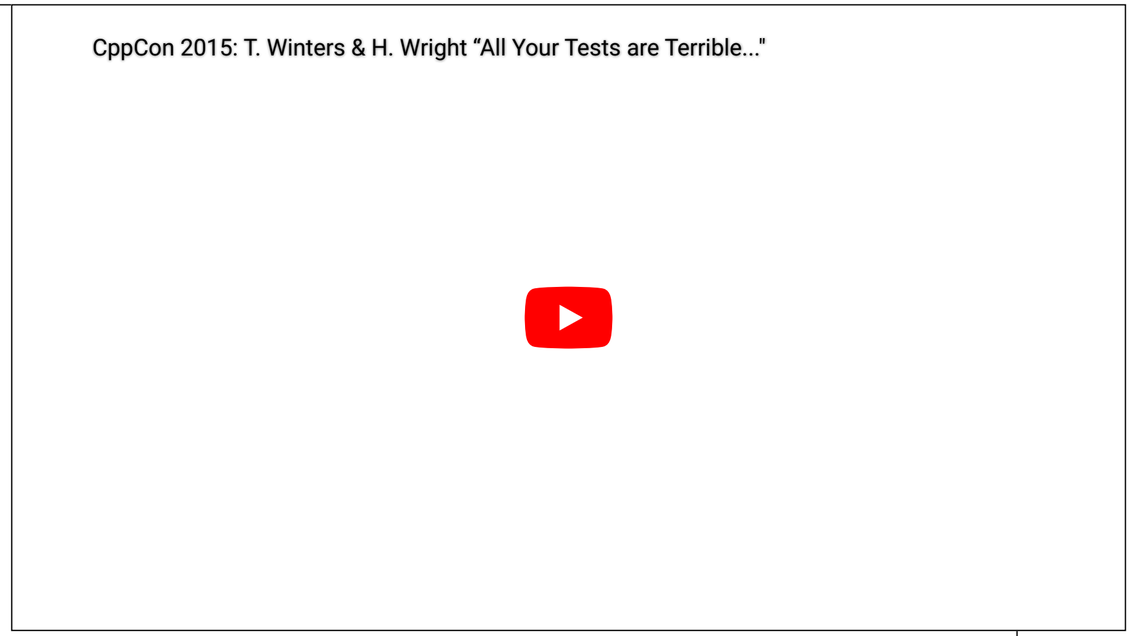
GENERAL ENGINEERING PRACTICES

Unit tests

Rewiring your brain
With Test Driven Thinking (in C++)



All Your Tests are Terrible:
Tales from the Trenches



Unit test problems

```
1 TEST_CASE("std::string is constructible from a string literal")
2 {
3     auto const s = std::string("Water 7");
4     CHECK(s.size() == 7);
5     CHECK(std::strcmp(s.data(), "Water 7") == 0);
6 }
```

A simple unit test solution

```
1  #include <cassert>
2  #include <cstring>
3  #include <string>
4
5  int main(int, char**)
6  {
7      auto const s = std::string("Water 7");
8      assert(std::strcmp(s.data(), "Water 7") == 0);
9      assert(s.size() == 7);
10     return 0;
11 }
```

Correctness

(includes reliability, safety, and security)

```
1 auto const v = std::vector{0, 1, 2};
2 return v[4]; // out-of-bounds access
```

Output with AddressSanitizer

```
=================================================================
==1==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x502000000020 ...
READ of size 4 at 0x502000000020 thread T0
    #0 0x558c7e4ce923 in main /tmp/example.cpp:6:12
    #1 0x7f48e5e29d8f  (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f)
    #2 0x7f48e5e29e3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x29e3f)
    #3 0x558c7e3f63b4 in _start (/tmp/output.s+0x2c3b4)
...
```

Build with `-fsanitize=address` globally, for libc++ and MSVC's STL

Build with `-fsanitize=address -D_GLIBCXX_SANITIZE_VECTOR` globally, for libstdc++

It's our job to help you identify mistakes!

```
1 return *std::optional<int>(); // deref empty optional == logic error
```

```
Program returned: 0
```

Hardened/assertion builds

```
constexpr T& operator*() & noexcept {
  _LIBCPP_ASSERT_VALID_ELEMENT_ACCESS(
    has_value(),
    "optional operator* called on a disengaged value");
    return get();
  }
```

```
.../optional:811: assertion has_value() failed:
                  optional operator* called on a disengaged value
Program terminated with signal: SIGILL
```

libc++ / libstdc++ / MSVC's STL

Is this code okay?

```
auto generator = std::mt19937_64(std::random_device()());
auto distribution = std::uniform_int_distribution<signed char>{};
```

Turns out that it's undefined

```
// [rand.dist.uni.int], class template uniform_int_distribution
template<class IntType = int>
  class uniform_int_distribution;
```

1 Throughout this subclause [rand], the effect of instantiating a template:

...

(1.5) — that has a template type parameter named `IntType` is undefined unless the corresponding template argument is *cv*-unqualified and is one of `short`, `int`, `long`, `long long`, `unsigned short`, `unsigned int`, `unsigned long`, or `unsigned long long`.

Undefined ⇒ badness

3.62 undefined behavior [defns.undefined]

behavior for which this document imposes no requirements

~~Undefined ⇒ badness~~

Undefined ⇒ implementation decides if valid

3.62 undefined behavior [defns.undefined]

behavior for which this document imposes no requirements

[*Note 1*: Undefined behavior may be expected when this document omits any explicit definition of behavior or when a program uses an erroneous construct or erroneous data. Permissible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a *diagnostic message* ([defns.diagnostic])), to terminating a translation or execution (with the issuance of a diagnostic message). Many erroneous program constructs do not engender undefined behavior; they are required to be diagnosed. Evaluation of a constant expression ([expr.const]) never exhibits behavior explicitly specified as undefined in [intro] through [cpp].— end note]

Plan for Hyrum's law

> *With a sufficient number of users of an API,*
> *it does not matter what you promise in the contract:*
> *all observable behaviors of your system*
> *will be depended on by somebody.*

—Hyrum Wright, hyrumslaw.com

Possible responses to Hyrum's law

Prioritise stability over progress

Prioritise progress over stability

Permanent rollback

Breaking any user is unacceptable

Temporary rollback

Users need to fix in a timely manner

Tiered release over three versions

1. deprecate + opt into change
2. opt out of change
3. old behaviour removed

Forward-fix prioritised

Temporary user pain so others have their needs met

Nothing to fix

Not enough users impacted, or change too important to revert

Possible responses to Hyrum's law

Prioritise stability over progress

Prioritise progress over stability

Permanent rollback

Breaking any user is unacceptable

Temporary rollback

Users need to fix in a timely manner

Tiered release over three versions

1. deprecate + opt into change
2. opt out of change
3. old behaviour removed

Forward-fix prioritised

Temporary user pain so others have their needs met

Nothing to fix

Not enough users impacted, or change too important to revert

How do we prevent Hyrum's law?

Simulate user input with fuzzing

```
// Part of the LLVM Project, under the Apache License v2.0 with LLVM E
// See https://llvm.org/LICENSE.txt for license information.
extern "C" int LLVMFuzzerTestOneInput(const std::uint8_t *data,
                                      std::size_t size) {
    std::vector<std::uint8_t> working(data, data + size);
    std::sort(working.begin(), working.end());

    if (!std::is_sorted(working.begin(), working.end()))
        return 1;
    // some more checks...
    return 0;
}
```

libc++ example

Use integration tests to catch Hyrumbugs

Identify who gets broken early.
Build as many packages on as many platforms as you can.

libc++	libstdc++	MSVC's STL
FreeBSD Gentoo	Debian Fedora Gentoo	Handled by the release team

ACT II

GEAR SECOND

Identify optimisation opportunities

- Analyse usage patterns first
- Profilers are a critical tool here
- Benchmark your change

Case study: `std::vector`

A naive implementation

```
1  template<class T>
2  class vector {
3  public:
4    // vector interface here...
5  private:
6
7    T* data_;
8    size_type size_;
9    size_type capacity_;
10 };
```

A good implementation

```
1  template<class T, class Alloc = std::allocator<T>>
2  class vector {
3  public:
4    // vector interface here...
5  private:
6    [[no_unique_address]] Alloc alloc_;
7    pointer data_;
8    size_type size_;
9    size_type capacity_;
10 };
```

How things look upstream

```
template<class T, class Alloc = std::allocator<T>>
class vector {
public:
  // vector interface here...
private:
  [[no_unique_address]] Alloc alloc_;
  pointer begin_;
  pointer end_;
  pointer capacity_;
};
```

only talk about performance when you have

BENCHMARKS

Benchmarks

- Microbenchmarks are kinda like unit tests
- Macrobenchmarks are kinda like integration tests
  - Influenced by link-time optimisation
  - Influenced by profile-guided optimisation

What does any of this have to do with implementing the standard library?

Everything!

What people think it's like

What it's actually like

By AS, 'Path to Mt Kosciuszko', CC BY 2.0, https://www.flickr.com/photos/aschaf/4492775854/in/photostream/

ACT III

UNDERSTAND WHAT YOU'RE GETTING YOURSELF INTO

What *is* the standard library?

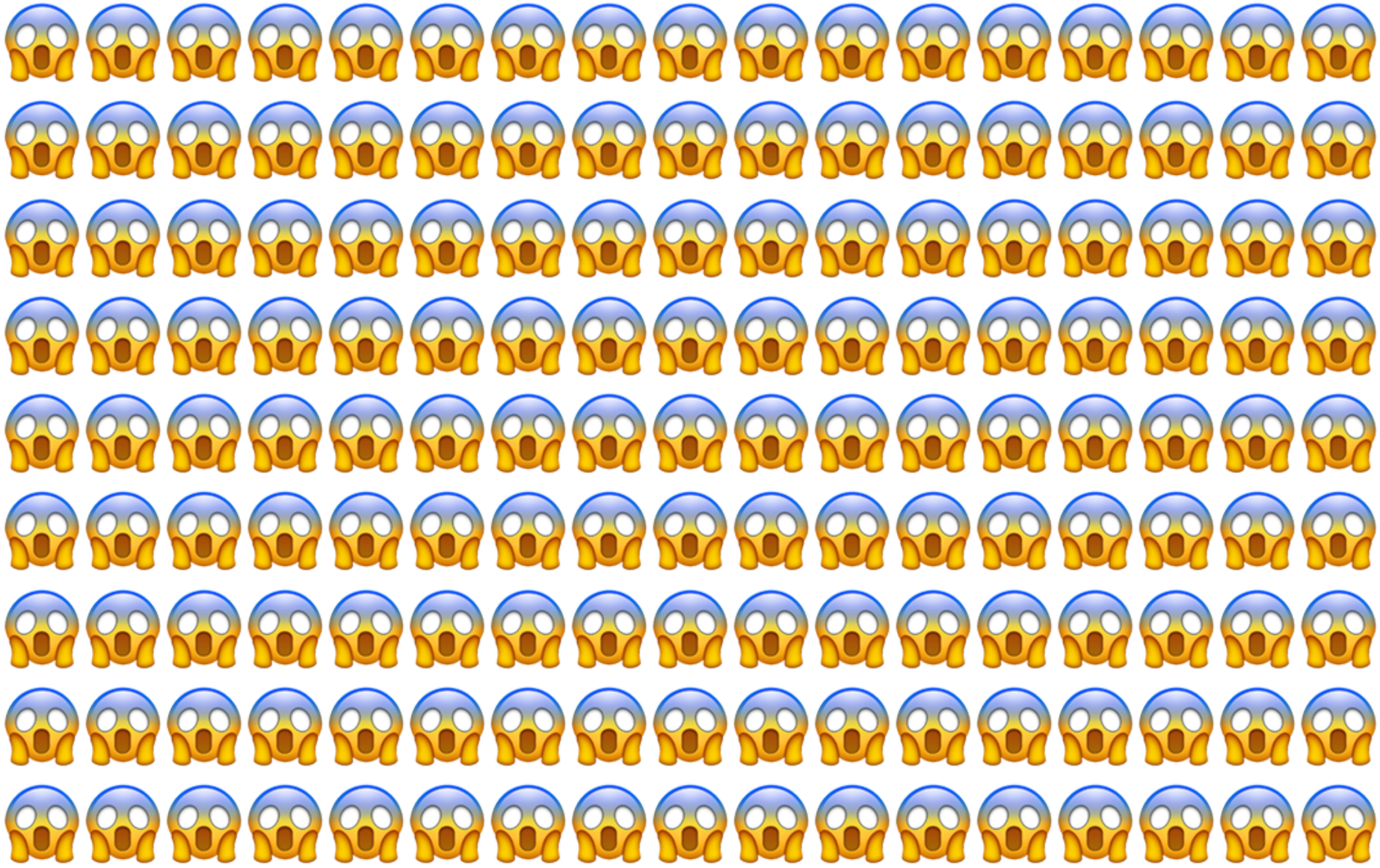
The headers that ship with the compiler (e.g. `<vector>`)

The body of text from [library] through [thread] in the C++ standard

It's in `std`

(or libc)

Don't use cppreference



```
template< class... Args1, class... Args2 >
pair( std::piecewise_construct_t,                    (since C++11)
      std::tuple<Args1...> first_args,               (until C++20)
      std::tuple<Args2...> second_args );
```

(9)

```
template< class... Args1, class... Args2 >
constexpr pair( std::piecewise_construct_t,          (since C++20)
                std::tuple<Args1...> first_ar
                std::tuple<Args2...> second_a
```

9) Forwards the elements of `first_args` to the constructor of `first` and forwards the elements of `second_args` to the constructor of `second`. This is the only non-default constructor that can be used to create a pair of non-copyable non-movable types. The program is ill-formed if first or second is a reference and bound to a temporary object.

Standardese

```
template<class... Args1, class... Args2>
constexpr pair(piecewise_construct_t,
               tuple<Args1...> first_args, tuple<Args2...> second_args
```

18 *Mandates:*

(18.1) — `is_constructible_v<T1, Args1...>` is `true` and

(18.2) — `is_constructible_v<T2, Args2...>` is `true`

19 *Effects*: Initializes `first` with arguments of types `Args1...` obtained by forwarding the elements of `first_args` and initializes `second` with arguments of types `Args2...` obtained by forwarding the elements of `second_args`. (Here, forwarding an element `x` of type `U` within a `tuple` object means calling `std::forward<U>(x)`.) This form of construction, whereby constructor arguments for `first` and `second` are each provided in a separate `tuple` object, is called *piecewise construction*.

[*Note 2*: If a data member of `pair` is of reference type and its initialization binds it to a temporary object, the program is ill-formed ([class.base.init]).— *end note*]

Words of power

- The standard gives certain words special meaning.
- These are in prose-text and are *italicised*.
- Some examples:
  - *Constraints*
  - *Mandates*
  - *Preconditions*
  - *Effects*
  - ...

(3.2) — *Mandates*: the conditions that, if not met, render the program ill-formed.

[*Example 2*: An implementation can express such a condition via the *constant-expression* in a *`static_assert`-declaration* ([dcl.pre]). If the diagnostic is to be emitted only after the function has been selected by overload resolution, an implementation can express such a condition via a *constraint-expression* ([temp.constr.decl]) and also define the function as deleted. — *end example*]

(3.1) — *Constraints*: the conditions for the function's participation in overload resolution ([over.match]).

[*Note 1*: Failure to meet such a condition results in the function's silent non-viability. — *end note*]

[*Example 1*: An implementation can express such a condition via a *constraint-expression* ([temp.constr.decl]). — *end example*]

Compare the pair

Based off cppreference

```
1 template<class... Args1, class... Args2>
2 requires is_constructible_v<T1, Args1...> and
3          is_constructible_v<T2, Args2...>
4 pair(piecewise_construct_t, tuple<Args1...> args1, tuple<Args2...> args2)
5 : first(std::make_from_tuple(std::move(args1)))
6 , second(std::make_from_tuple(std::move(args2)))
7 {}
```

Based off standardese

```
1 template<class... Args1, class... Args2>
2 pair(piecewise_construct_t, tuple<Args1...> args1, tuple<Args2...> args2)
3 : first(std::make_from_tuple(std::move(args1)))
4 , second(std::make_from_tuple(std::move(args2)))
5 {
6   static_assert(is_constructible_v<T1, Args1...>);
7   static_assert(is_constructible_v<T2, Args2...>);
8 }
```

Why this matters

```
1 static_assert(std::is_constructible_v<std::pair<int, int>,
2                                       std::piecewise_construct_t,
3                                       std::tuple<int>,
4                                       std::tuple<int*>>);
```

The code needs to compile for backwards-compatibility

```
explicit vector( size_type count,
                 const T& value = T(),
                 const Allocator& alloc = Allocator() );
```
(until C++11)

```
vector( size_type count,
        const T& value,
        const Allocator& alloc = Allocator() );
```
(3)

(since C++11)
(until C++20)

```
constexpr vector( size_type count,
                  const T& value,
                  const Allocator& alloc = Allocator() );
```
(since C++20)

Constructs a new container from a variety of data sources, optionally using a user supplied allocator `alloc`.

...

3) Constructs the container with `count` copies of elements with value `value`.

Complexity

3,4) Linear in `count`

Standardese

```
constexpr vector(size_type n, const T& value,
                 const Allocator& = Allocator());
```

6 *Preconditions:* T is *`Cpp17CopyInsertable`* into `*this`.

7 *Effects:* Constructs a `vector` with n copies of `value`, using the specified allocator.

8 *Complexity:* Linear in n.

~~Don't use cppreference~~

Only consult the standard when implementing C++

Great for user code

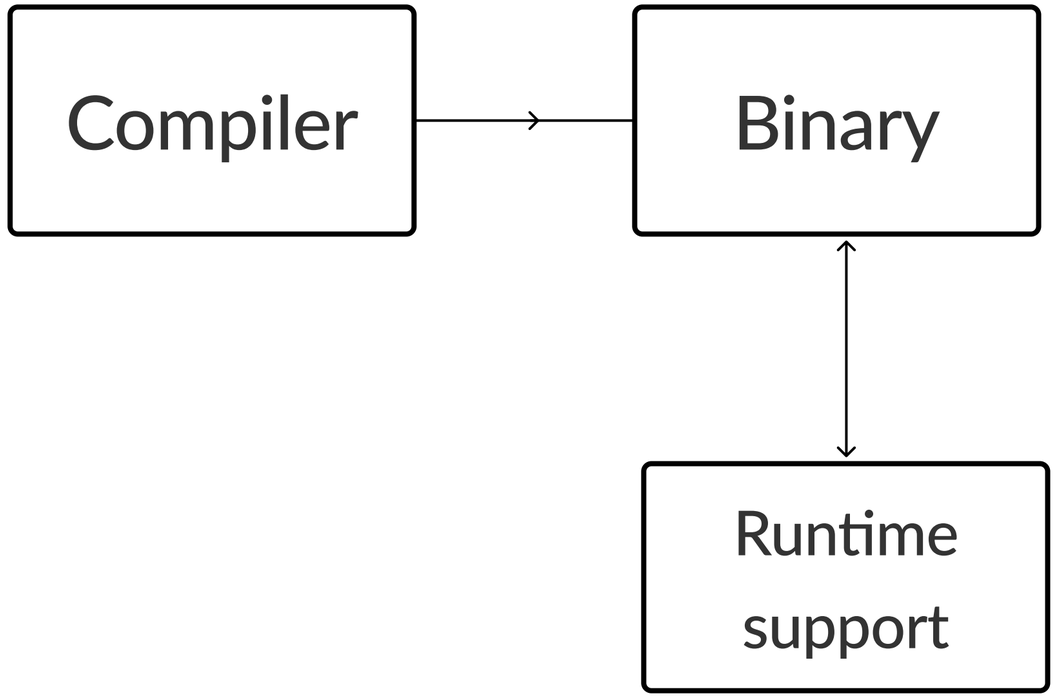
cppreference
MSDN
Blogs, books, etc.

Required for compiler/stdlib work

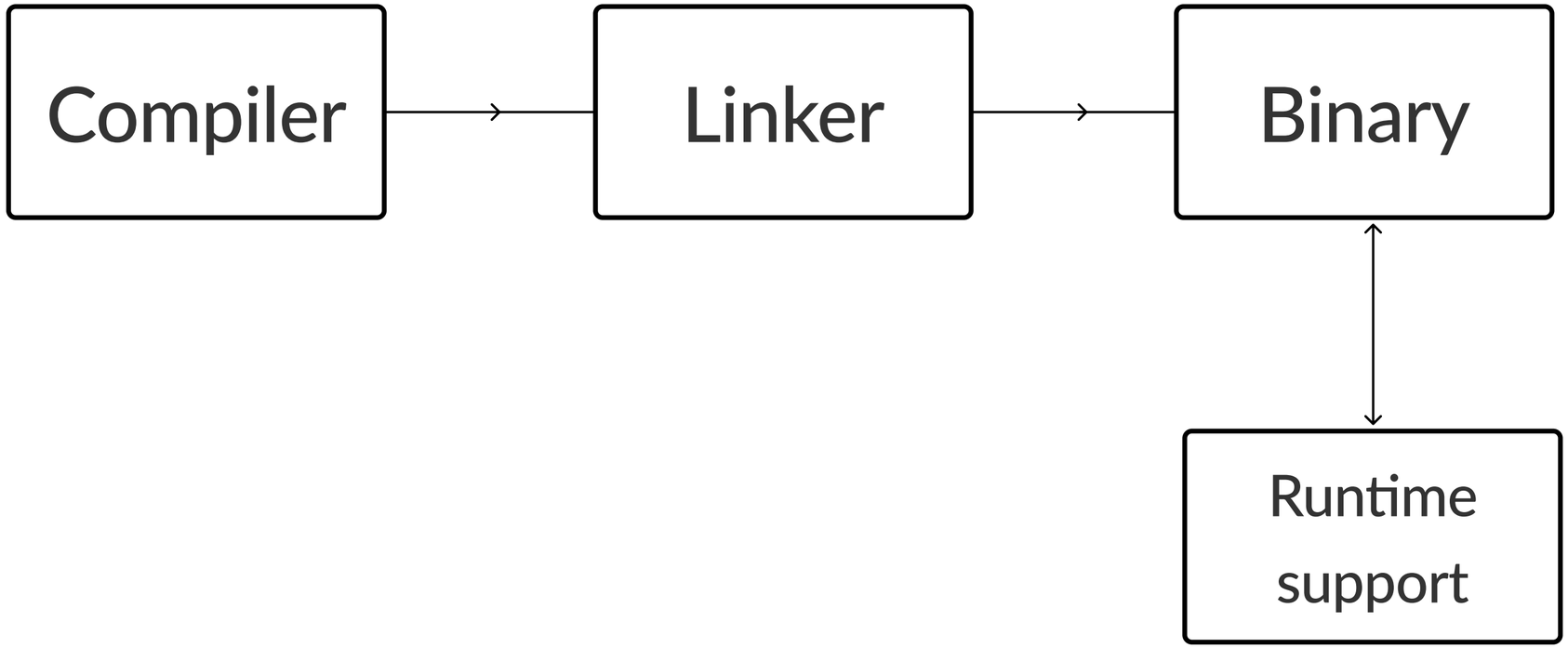
C++ working paper
Hana's search engine

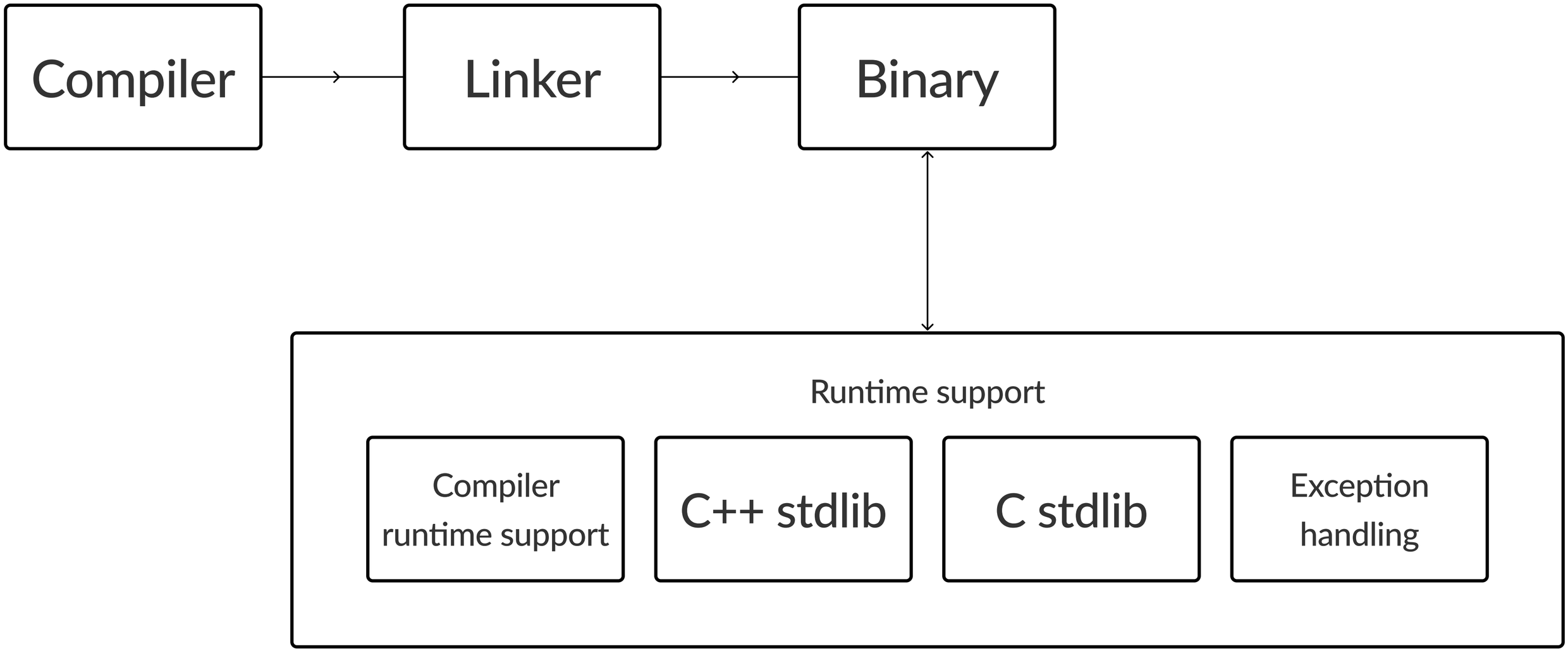
The same applies to reviewing contributions.

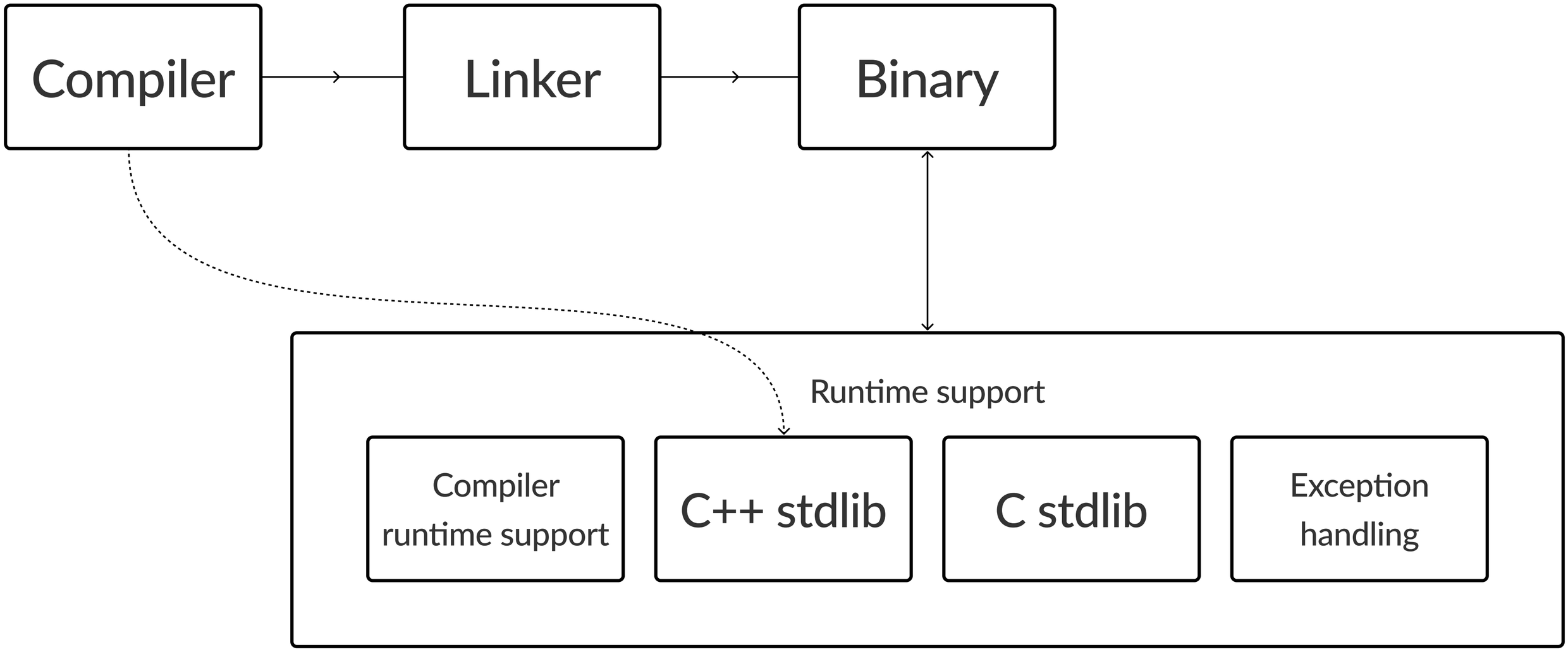
What is the C++ implementation?



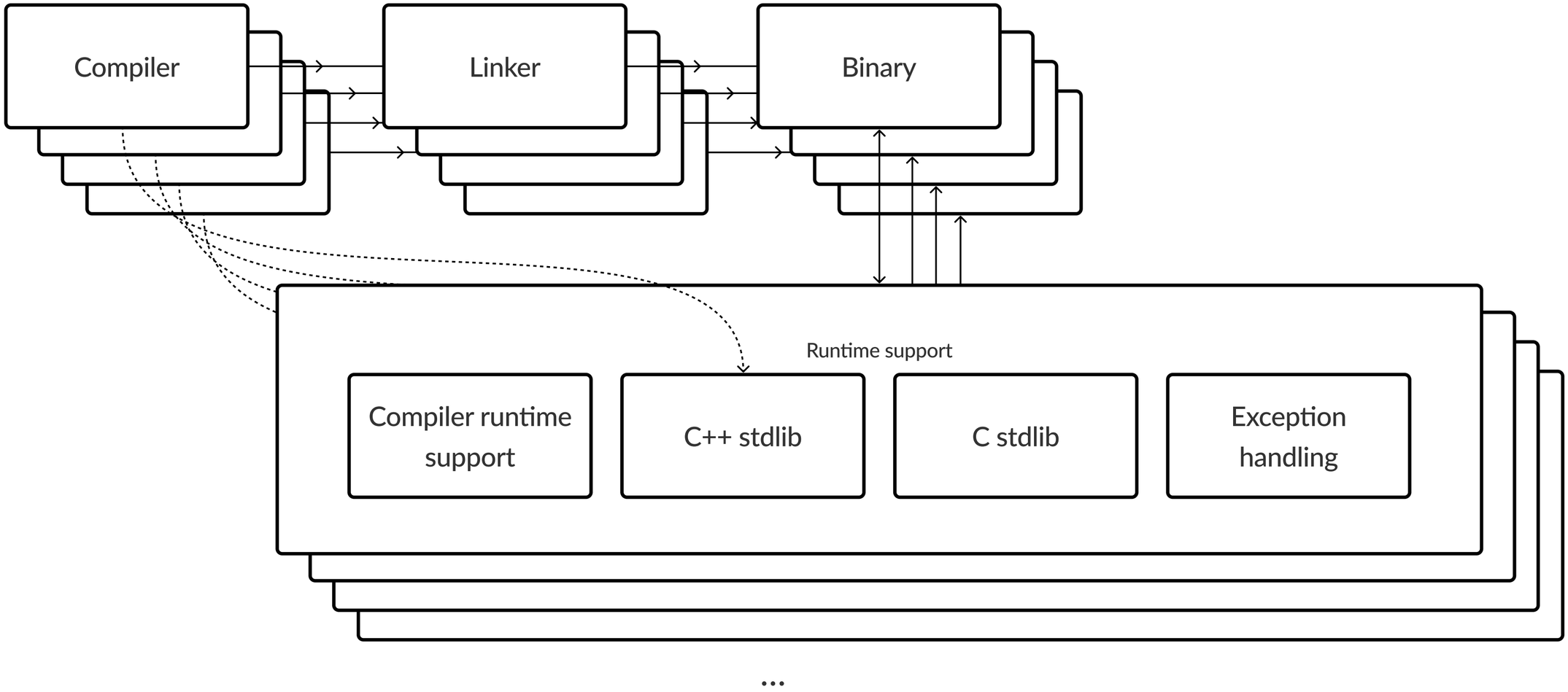
What is the C++ implementation?



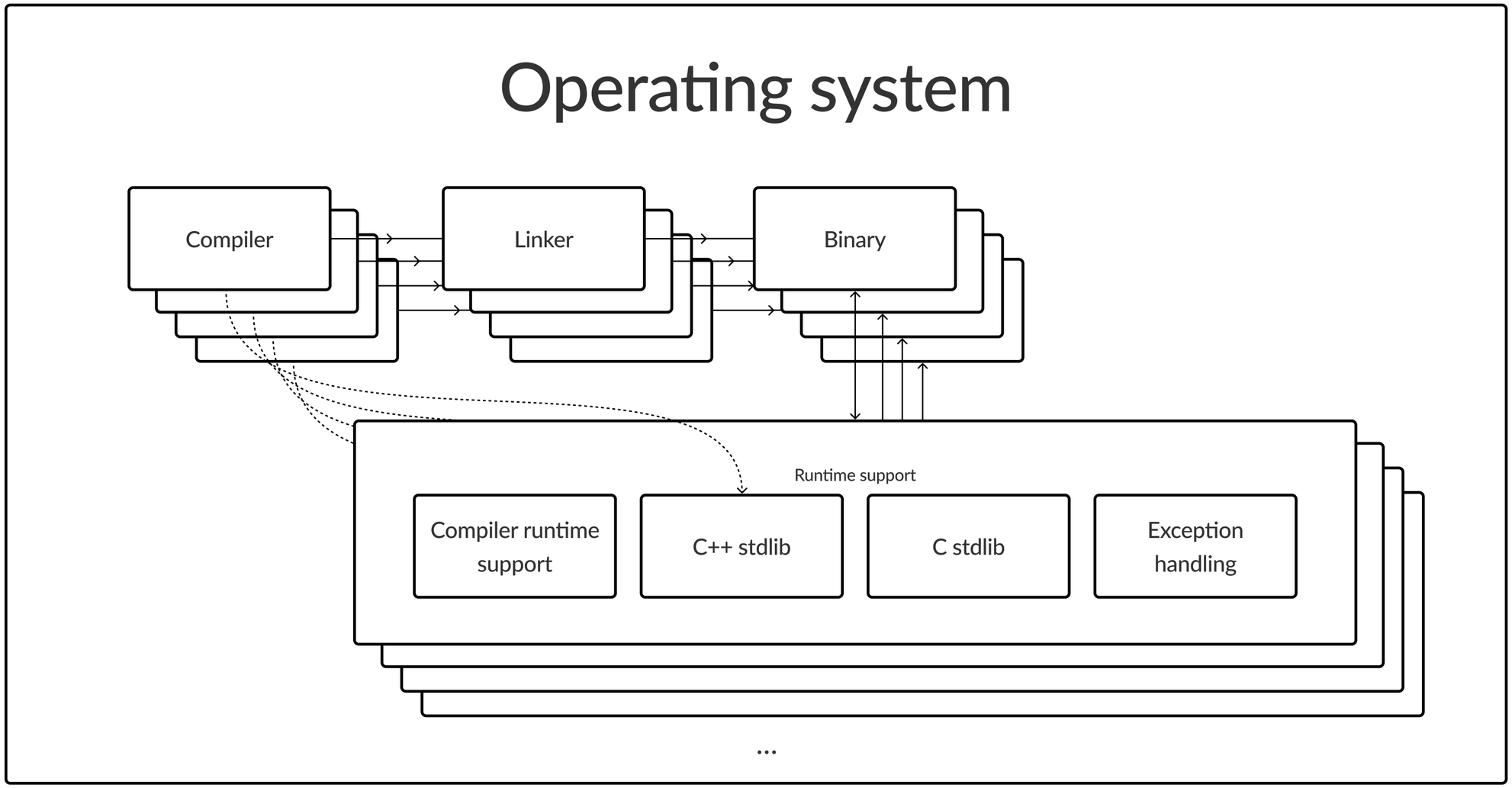

Compiler
Linker
Binary
Runtime support
Compiler runtime support
C++ stdlib
C stdlib
Exception handling


Compiler
Linker
Binary
Runtime support
Compiler runtime support
C++ stdlib
C stdlib
Exception handling

Operating system



Hardware


Operating system
Compiler
Linker
Binary
Runtime support
Compiler runtime support
C++ stdlib
C stdlib
Exception handling
...

Standard library implementers are compiler developers

```
$ apt install clang-18 libc++-18-dev libc++abi-18-dev
```

Direct input from the compiler can be unavoidable

General language features (e.g. modules for `module std;`)

`std::is_trivially_copyable_v<T>`

`std::is_constant_evaluated()`

`std::source_location`

All atomic operations

Coroutines

Direct input from the compiler can be optimal

Clang-supported

`std::move`, etc.
`std::make_integer_sequence`
`std::printf`'s interface

Hopefully soon

`std::invoke`
`std::tuple`'s guts
`std::format`'s interface

Most of the standard library should still be implemented in C++ code

Algorithms
Containers
Iterators
Ranges
I/O logic
Stacktrace support
Algebraic data types

ACT IV

THE EXCLUSIVITY OF THE STANDARD LIBRARY

Remember: we want good diagnostics!

```
1 struct S {};
2 S s;
3 std::print("{}", s);
```

```
In file included from source.cpp:1:
In file included from /usr/include/c++/14.0.1/print:41:
/usr/include/c++/14.0.1/format:4030:3: error: static assertion failed due to requirement 'is_default_constructible_v<std::formatter<S, char>>': std::formatter must be specialized for each type being formatted
 4030 |         (is_default_constructible_v<formatter<_Args, _CharT>> && ...),
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/c++/14.0.1/format:4174:4: note: in instantiation of template class 'std::__format::_Checking_scanner<char, S>' requested here
 4174 |           __scanner(_M_str);
      |           ^
source.cpp:8:16: note: in instantiation of function template specialization 'std::basic_format_string<char, S &>::basic_format_string<char[3]>' requested here
    8 |           std::print("{}", s);
      |                      ^
source.cpp:8:16: error: call to consteval function 'std::basic_format_string<char, S &>::basic_format_string<char[3]>' is not a constant expression
    8 |           std::print("{}", s);
      |                      ^
In file included from source.cpp:1:
In file included from /usr/include/c++/14.0.1/print:41:
/usr/include/c++/14.0.1/format:3711:31: error: no matching constructor for initialization of 'basic_format_arg<basic_format_context<_Sink_iter<char>, char>>'
 3711 |           basic_format_arg<_Context> __arg(__v);
      |                                      ^     ~~~
/usr/include/c++/14.0.1/format:3722:12: note: in instantiation of function template specialization 'std::__format::_Arg_store<std::basic_format_context<std::__format::_Sink_iter<char>, char>, std::basic_format_arg<std::basic_format_context<st
 3722 |         : _M_args{_S_make_elt(__a)...}
      |                   ^
/usr/include/c++/14.0.1/format:3772:14: note: in instantiation of function template specialization 'std::__format::_Arg_store<std::basic_format_context<std::__format::_Sink_iter<char>, char>, std::basic_format_arg<std::basic_format_context<st
 3772 |       return _Store(__fmt_args...);
      |              ^
/usr/include/c++/14.0.1/print:116:12: note: in instantiation of function template specialization 'std::print<S &>' requested here
  116 |     { std::print(stdout, __fmt, std::forward<_Args>(__args)...); }
      |            ^
source.cpp:8:10: note: in instantiation of function template specialization 'std::print<S &>' requested here
    8 |           std::print("{}", s);
      |                ^
/usr/include/c++/14.0.1/format:3197:11: note: candidate constructor (the implicit copy constructor) not viable: no known conversion from 'S' to 'const basic_format_arg<basic_format_context<_Sink_iter<char>, char>>' for 1st argument
 3197 |     class basic_format_arg
      |           ^~~~~~~~~~~~~~~
/usr/include/c++/14.0.1/format:3197:11: note: candidate constructor (the implicit move constructor) not viable: no known conversion from 'S' to 'basic_format_arg<basic_format_context<_Sink_iter<char>, char>>' for 1st argument
 3197 |     class basic_format_arg
      |           ^~~~~~~~~~~~~~~
/usr/include/c++/14.0.1/format:3458:2: note: candidate template ignored: constraints not satisfied [with _Tp = S]
 3458 |         basic_format_arg(_Tp& __v) noexcept
      |         ^
/usr/include/c++/14.0.1/format:3456:11: note: because '__format::__formattable_with<S, std::basic_format_context<std::__format::_Sink_iter<char>, char> >' evaluated to false
 3456 |         requires __format::__formattable_with<_Tp, _Context>
      |                  ^
/usr/include/c++/14.0.1/format:2503:9: note: because 'std::formatter<S>' does not satisfy 'semiregular'
 2503 |       = semiregular<_Formatter>
      |         ^
/usr/include/c++/14.0.1/concepts:280:27: note: because 'std::formatter<S>' does not satisfy 'copyable'
  280 |     concept semiregular = copyable<_Tp> && default_initializable<_Tp>;
      |                           ^
/usr/include/c++/14.0.1/concepts:275:24: note: because 'std::formatter<S>' does not satisfy 'copy_constructible'
  275 |     concept copyable = copy_constructible<_Tp> && movable<_Tp>
      |                        ^
/usr/include/c++/14.0.1/concepts:179:9: note: because 'std::formatter<S>' does not satisfy 'move_constructible'
  179 |       = move_constructible<_Tp>
      |         ^
/usr/include/c++/14.0.1/concepts:174:7: note: because 'constructible_from<std::formatter<S>, std::formatter<S> >' evaluated to false
  174 |     = constructible_from<_Tp, _Tp> && convertible_to<_Tp, _Tp>;
      |       ^
/usr/include/c++/14.0.1/concepts:160:30: note: because 'is_constructible_v<std::formatter<S>, std::formatter<S> >' evaluated to false
  160 |       = destructible<_Tp> && is_constructible_v<_Tp, _Args...>;
      |                              ^
/usr/include/c++/14.0.1/format:3258:7: note: candidate constructor not viable: requires 0 arguments, but 1 was provided
 3258 |       basic_format_arg() noexcept : _M_type(__format::_Arg_none) { }
      |       ^
```

Not great, but could be worse...

```
example.cpp
C:/data/msvc/14.39.33321-Pre/include\format(687): error C2672: 'std::_Format_arg_traits<_Context>::_Type_eraser': no matching overloaded function found
        with
        [
            _Context=std::format_context
        ]
C:/data/msvc/14.39.33321-Pre/include\format(684): note: could be 'auto std::_Format_arg_traits<_Context>::_Type_eraser(void)'
        with
        [
            _Context=std::format_context
        ]
C:/data/msvc/14.39.33321-Pre/include\format(687): note: the associated constraints are not satisfied
C:/data/msvc/14.39.33321-Pre/include\format(847): note: the concept 'std::_Formattable_with<S,std::format_context,std::formatter<S,_CharT>>' evaluated to false
        with
        [
            _CharT=char
        ]
C:/data/msvc/14.39.33321-Pre/include\format(661): note: the concept 'std::semiregular<std::formatter<S,_CharT>>' evaluated to false
        with
        [
            _CharT=char
        ]
C:/data/msvc/14.39.33321-Pre/include\concepts(255): note: the concept 'std::copyable<std::formatter<S,_CharT>>' evaluated to false
        with
        [
            _CharT=char
        ]
C:/data/msvc/14.39.33321-Pre/include\concepts(248): note: the concept 'std::copy_constructible<std::formatter<S,_CharT>>' evaluated to false
        with
        [
            _CharT=char
        ]
C:/data/msvc/14.39.33321-Pre/include\concepts(170): note: the concept 'std::move_constructible<std::formatter<S,_CharT>>' evaluated to false
        with
        [
            _CharT=char
        ]
C:/data/msvc/14.39.33321-Pre/include\concepts(105): note: the concept 'std::constructible_from<std::formatter<S,_CharT>,std::formatter<S,_CharT>>' evaluated to false
        with
        [
            _CharT=char
        ]
C:/data/msvc/14.39.33321-Pre/include\concepts(95): note: the constraint was not satisfied
C:/data/msvc/14.39.33321-Pre/include\format(687): note: the template instantiation context (the oldest one first) is
<source>(8): note: see reference to function template instantiation 'std::basic_format_string<char,S &>::basic_format_string<char[3]>(const _Ty (&))' being compiled
        with
        [
            _Ty=char [3]
        ]
C:/data/msvc/14.39.33321-Pre/include\format(3802): note: see reference to class template instantiation 'std::__p2286::_Format_checker<_CharT,S>' being compiled
        with
        [
            _CharT=char
        ]
C:/data/msvc/14.39.33321-Pre/include\format(3579): note: while compiling class template member function 'std::__p2286::_Format_checker<_CharT,S>::_Format_checker(std::basic_string_view<char,std::char_traits<char>>) noexcept'
        with
        [
            _CharT=char
```

... a lot worse

```
source.cpp:8:10: error: no member named 'print' in namespace 'std'
    8 |     std::print("{}", s);
      |     ~~~~~^
```

Can we do better?

```
1 struct S {};
2 auto const s = S{};
3 std::printf("%d", s);
```

```
source.cpp:8:23: warning: format specifies type 'int' but the argument has type 'const S' [-Wformat]
   8 |     std::printf("%d", s);
     |                  ~~   ^
```

What do you think?

```
1 struct S {};
2 auto const s = S{};
3 std::print("{}", s);
```

```
source.cpp:8:22: error: type 'S' does not have a format specifier
   8 |     std::print("{}", s);
     |                  ~~   ^
source.cpp:8:23: note: a partial or explicit specialization for class template 'std::formatter<S, char>
```

Ideal scenario in the current framework?

```
source.cpp:8:22: error: type 'S' does not have a format specifier
   8 |     std::print("{}", s);
     |                  ~~   ^
source.cpp:8:23: note: a partial or explicit specialization for class template 'std::formatter<S, char>
     |     // sample suggestion
     |     template<>
     |     struct std::formatter<S, char> {
     |       template<class ParseContext>
     |       constexpr typename ParseContext::iterator parse(ParseContext &c) {
     |         return c.begin();
     |       }
     |
     |       template<class FormatContext>
     |       typename FormatContext::iterator format(S s, FormatContext &c) const {
     |         return std::ranges::copy("Shishishi!", c.out()).out;
     |       }
     |     };
```

Compiler extensions are good

for the standard library to use on your behalf

ACT V

WE WOULD BE HONOURED
IF YOU WOULD JOIN US

The compiler and runtime are inextricably linked

and this is a good thing

Our engineering practices are the same as yours

A lot of tenacity

Join our crew!

Onboarding pages:

- libc++
- libstdc++
- MSVC's STL

https://slides.cjdb.xyz

C++ now

What Does it Take to Implement the Standard Library?

Christopher Di Bella

2024