C++26 Preview

Jeff Garland

Status of c++26 - it'll ship on time

- it is an immutable property of committee process
  - 'train model' - what is ready ships
  - there is a plan: http://wg21.link/P0592
- much committee work has gone virtual
  - subgroup meetings on going every week
  - 3 virtual plenary sessions / year
  - opened the door to participation!

Status of c++26 - it'll ship on time

- it is an immutable property of committee process
  - 'train model' - what is ready ships
  - there is a plan: http://wg21.link/P0592
- much committee work has gone virtual
  - subgroup meetings on going every week
  - 3 virtual plenary sessions / year
  - opened the door to participation!

- c++26 schedule
  - 2022-02 meeting (Kent Wa) was last c++ 2023 meeting
  - 2025-02 –> c++26 feature freeze – **just months away!**
  - 2025-06 –> c++26 draft ships
  - June to Feb 2026– national body/iso review and bug fixing
  - late 2026 –> final iso approval

Priorities for c++26

- bug fixes - always a high priority
- concurrency support `std::execution` (P2300)
- ~~networking~~
- reflection, ~~contracts~~, ~~pattern matching~~
- additions to c++20/23 features `ranges`, `format`
- mop up c++23 stuff that missed the train
- really - there's so much more
- despite the above 26 will be large
- cppreference 26

Priorities for this talk

- discuss **the smaller features**
- focus more on **everyday programmer** features
- new collections, new ranges, language features

Outline of the talk I

- Language & Library
  - debugging
  - structured bindings
- Language
  - Templates
  - Misc
  - ~~Contracts~~
  - ~~Reflection~~

Outline of the talk II

- Library:
  - string processing
  - format additions
  - containers
  - ranges
  - utilities
  - general math support
  - constexpr all the things
  - ~~concurrency~~
  - ~~simd~~
  - ~~linear algebra and `mdspan`~~

Language and Library

Debugging

P2741 User-generated `static_assert` messages

- pass a string to `static_assert`
- build-time diagnostics

```
1 static_assert(sizeof(S) == 1,
2                std::format("Unexpected sizeof: expected 1, got {}", sizeof(S)))
```

P2741 User-generated `static_assert` messages

- pass a string to `static_assert`
- build-time diagnostics

```
_assert(sizeof(S) == 1,
        std::format("Unexpected sizeof: expected 1, got {}", sizeof(S)))
```

P2573 = delete("should have a reason");

```
1 void newapi();
2 void oldapi() = delete("oldapi() is outdated and been removed - newapi().");
3
4 template<typename T>
5 struct A {/* ... */};
6 template<typename T>
7 A<T> factory(const T&) {/* process lvalue */}
8 template<typename T>
9 A<T> factory(const T&&) = delete("Using rvalue may result in dangling reference");
```

- relevant blog post
  https://quuxplusone.github.io/blog/2021/10/17/equals-delete-means/

P2573 = delete("should have a reason");

```cpp
void newapi();
void oldapi() = delete("oldapi() is outdated and been removed - newapi().");

template<typename T>
struct A {/* ... */};
template<typename T>
A<T> factory(const T&) {/* process lvalue */}
template<typename T>
A<T> factory(const T&&) = delete("Using rvalue may result in dangling reference");
```

- relevant blog post https://quuxplusone.github.io/blog/2021/10/17/equals-delete-means/

P2264 Make assert() macro user friendly for C and C++

```
1 //new definition
2 #define assert(...) ((void)0)
3
4 assert(x > 0 , "x was not greater than zero"); //user error specification
```

Library - Debugging Support

- new header <debugger>
- Enables tooling
- lots of existing implementations
- improvements to assert macro

```
#include <debugging>

int main()
{
   std::breakpoint_if_debugging(); //stop if in debugger
}
```

`<debugging>` support details

- P2546 Debugging Support and
- P2810 `is_debugger_present` `is_replaceable`

Functions	Description
breakpoint	pauses the running program
`breakpoint_if_debugging`	calls breakpoint if `std::is_debugger_present` returns true
`is_debugger_present`	checks whether a program is running under the control of a debugger

Structured Bindings

P0963 Structured binding declaration as a condition

- use as conditions in if, while, for, and switch statements
- success evaluation **prior** to destructuring
- blog: https://mariusbancila.ro/blog/2024/09/06/whats-new-in-c26-part-1/

```
1  //before (needs P2497 update to to_chars)
2  if (auto result = std::to_chars(p, last, 42)) {
3      // okay, use char pointer
4      auto [ptr, _] = result;
5  } else {
6      // handle errors
7      auto [_, ec] = result;
8  }
9
10 //after:
11 if (auto [to, ec] = std::to_chars(p, last, 42))
12 {
13     auto s = std::string(p, to);
14     ...
15 }
```

P0963 Structured binding declaration as a condition

- use as conditions in if, while, for, and switch statements
- success evaluation **prior** to destructuring
- blog: https://mariusbancila.ro/blog/2024/09/06/whats-new-in-c26-part-1/

```
1  //before (needs P2497 update to to_chars)
2  if (auto result = std::to_chars(p, last, 42)) {
3      // okay, use char pointer
4      auto [ptr, _] = result;
5  } else {
6      // handle errors
7      auto [_, ec] = result;
8  }
9
10 //after:
11 if (auto [to, ec] = std::to_chars(p, last, 42))
12 {
13     auto s = std::string(p, to);
14     ...
15 }
```

P2169 A nice placeholder with no name

```
1 //                before                              //after
2 std::lock_guard namingIsHard(mutex);                 std::lock_guard _(mutex);
3 // Structured binding
4 [[maybe_unused]] auto [x, y, iDontCare] = f();       auto [x, y, _] = f();
```

P2169 A nice placeholder with no name

```
1 //                before                          //after
2 std::lock_guard namingIsHard(mutex);             std::lock_guard _(mutex);
3 // Structured binding
4 [[maybe_unused]] auto [x, y, iDontCare] = f();   auto [x, y, _] = f();
```

P0609 Attributes for Structured Bindings

```
auto g() {
  [[maybe_unused]] auto h = f();
  auto [a, b [[maybe_unused]], c] = f();
  return a + c;
}
```

- https://godbolt.org/z/68WGq46dj

P2819 Add tuple protocol to complex

- in draft

```
1 //before
2 complex<double> c{…};
3 auto & [r, i]{reinterpret_cast<double(&)[2]>(c)};
4
5 //after
6 complex<double> c{…};
7 auto & [r, i]{c};
```

P2819 Add tuple protocol to complex

- in draft

```
1 //before
2 complex<double> c{…};
3 auto & [r, i]{reinterpret_cast<double(&)[2]>(c)};
4
5 //after
6 complex<double> c{…};
7 auto & [r, i]{c};
```

Language

Language - Misc

P2558 Add to the basic character set `@`, `$`, and `` ` ``

- C23 is adding these to the character set
- C++ should too
- several compilers already support $ as extension
- in draft

P2752 Static storage for braced initializers

```cpp
std::vector<int> v = {1,2,3}; //what does this do?

std::vector<char> vimg = {
    #embed "2mb-image.png"
};
```

- gcc 14 impl example https://godbolt.org/z/havqf5cPT

P2752 Static storage for braced initializers

```
1 std::vector<int> v = {1,2,3}; //what does this do?
2
3 std::vector<char> vimg = {
4     #embed "2mb-image.png"
5 };
```

- gcc 14 impl example https://godbolt.org/z/havqf5cPT

Language - Templates

P2662 Pack indexing

- previously meta-programming
- complex code and slow compiles

```
1  // syntax is name-of-a-pack ... [constant-expression]
2  template <typename... T>
3  constexpr auto first_plus_last(T... values) -> T...[0] {
4    return T...[0](values...[0] + values...[sizeof...(values)-1]);
5  }
6
7  int main() {
8     //first_plus_last(); // ill formed
9     static_assert(first_plus_last(1, 2, 10) == 11);
10 }
```

P2662 Pack indexing

- previously meta-programming
- complex code and slow compiles

```
1  // syntax is name-of-a-pack ... [constant-expression]
2  template <typename... T>
3  constexpr auto first_plus_last(T... values) -> T...[0] {
4    return T...[0](values...[0] + values...[sizeof...(values)-1]);
5  }
6
7  int main() {
8     //first_plus_last(); // ill formed
9     static_assert(first_plus_last(1, 2, 10) == 11);
10 }
```

P2841 Concept and variable-template template-parameters

```cpp
template<
  template <typename T> concept C,
  template <typename T> auto V
>
struct S{};

template <typename T>
concept MyConcept = true; // concept definition

template <typename T>
constexpr auto MyVar = 42; //variable template definition

S<MyConcept, MyVar> s;
```

- in Core

P2841 Concept and variable-template template-parameters

```cpp
template<
  template <typename T> concept C,
  template <typename T> auto V
>
struct S{};

template <typename T>
concept MyConcept = true; // concept definition

template <typename T>
constexpr auto MyVar = 42; //variable template definition

S<MyConcept, MyVar> s;
```

- in Core

P2841 Concept and variable-template template-parameters

```
template<
  template <typename T> concept C,
  template <typename T> auto V
>
struct S{};

template <typename T>
concept MyConcept = true; // concept definition

template <typename T>
constexpr auto MyVar = 42; //variable template definition

S<MyConcept, MyVar> s;
```

- in Core

P2841 Concept and variable-template template-parameters

```
1  template<
2    template <typename T> concept C,
3    template <typename T> auto V
4  >
5  struct S{};
6
7  template <typename T>
8  concept MyConcept = true; // concept definition
9
10 template <typename T>
11 constexpr auto MyVar = 42; //variable template definition
12
13 S<MyConcept, MyVar> s;
```

- in Core

P2893 Variadic friends

- in certain design patterns want to grant access to friends
- today must explicitly list
- prevents use of variadics

```
1  //before
2  template<class T=void,
3           class U=void>
4  class Foo {
5    friend T;
6    friend U;
7  };
8
9  //after
10 template<class... Ts>
11 class Foo {
12   friend Ts...;
13 };
```

P2893 Variadic friends

- in certain design patterns want to grant access to friends
- today must explicitly list
- prevents use of variadics

```
1  //before
2  template<class T=void,
3           class U=void>
4  class Foo {
5    friend T;
6    friend U;
7  };
8
9  //after
10 template<class... Ts>
11 class Foo {
12   friend Ts...;
13 };
```

Language - Contracts

P2900 Contracts for C++

- Minimum viable product paper
- Reviewed in Tokyo
- Provides for preconditions, postconditions, and contract assertions

```
1 int f(const int x)
2    pre (x != 1)      // a precondition assertion
3    post(r : r != 2) // a postcondition assertion; r refers to the return value of f
4 {
5    contract_assert (x != 3); // an assertion statement
6    return x;
7 }
```

evalution and contract violation

- semantics: ignore, enforce(default), or observe
- ignore –> no effect
- false evaluation of the predicate exits
    - runtime calls contract violation handler
    - buildtime (consteval) build failure

`contract_violation` details

- `::handle_contract_violation` is called
- gets `std::contracts::contract_violation`
- implementation may allow users to replace

```
//[contracts.syn]
class contract_violation {
  // No user-accessible constructor
public:
   // cannot be copied, moved, assigned
  contract_violation(const contract_violation&) = delete;
  contract_violation& operator=(const contract_violation&) = delete;
  const char* comment() const noexcept;
  detection_mode detection_mode() const noexcept;   // predicate_false, evaluation_exce
  contract_kind kind() const noexcept;  //pre,post, assert
  source_location location() const noexcept;
  contract_semantic semantic() const noexcept; //enforce, observe
};
```

what about compile time?

```
1 constexpr int l(int c) pre(c >= 2) {
2   return (c % 2 == 0) ? c / 0 : c;
3 }
4
5 const int i0 = l(0); // dynamic initialization is contract violation or undefined beh
6 const int i1 = l(1); // static initialization to 1 or contract violation at compile t
7 const int i2 = l(2); // dynamic initialization is undefined behavior
8 const int i3 = l(3); // static initialization to 3
```

Language - Reflection

P2996 Reflection for C++26

- Minimal viable product
- reflection operator (prefix ^)
- retruns opaque type `std::meta::info`
- consteval metafunctions over `meta::info`
- splicers `[: refl :]` to produce grammatical elements

enum to string

- compiler explorer
- meta functions: `enumerators_of`, `name_of`

```cpp
template<typename E>
  requires std::is_enum_v<E>
constexpr std::string enum_to_string(E value) {
  std::string result = "<unnamed>";
  [:expand(std::meta::enumerators_of(^E)):] >>
  [&]<auto e>{
    if (value == [:e:]) {
      result = std::meta::name_of(e);
    }
  };
  return result;
}

enum Color { red, green, blue };
static_assert(enum_to_string(Color::red) == "red");
static_assert(enum_to_string(Color(42)) == "<unnamed>");
```

meta:: traits on steroids

```
 // [meta.reflection.names], reflection names and locations
 consteval string_view name_of(info r);
 consteval string_view qualified_name_of(info r);
 consteval string_view display_name_of(info r);
 consteval source_location source_location_of(info r);

// [meta.reflection.queries], reflection queries
 consteval bool is_public(info r);
 consteval bool is_protected(info r);
 consteval bool is_private(info r);
 consteval bool is_virtual(info r);
 consteval bool is_pure_virtual(info r);
 consteval bool is_override(info r);

// [meta.reflection.member.queries], reflection member queries
template<class... Fs>
  consteval vector<info> members_of(info type, Fs... filters);
template<class... Fs>
  consteval vector<info> bases_of(info type, Fs... filters);

  many many more
```

`define_class`

- allows for generation of new types
- fairly limited for now

```
1  template<typename T> struct S;
2  constexpr auto U = define_class(^S<int>, {
3    data_member_spec(^int, {.name="i", .align=64}),
4    data_member_spec(^int, {.name="j", .align=64}),
5  });
6
7  // S<int> is now defined to the equivalent of
8  // template<> struct S<int> {
9  //   alignas(64) int i;
10 //   alignas(64) int j;
11 // };
```

Library

Library - String Processing

P2495 Interfacing stringstreams with `string_view`

- in draft

```
1 //implicitly convertable to string_view
2 const mystring str;
3
4 stringstream s1{""sv};
5 stringstream s2{str};
6 s2.str(""sv};
```

P3044 `sub-string_view` from string

```
1 string s{"Hello cruel world!"};
2 auto sub = s.subview(5);
3 //sub == "cruel world!"
4 auto subsub = sub.subview(0, 6);
5 //subsub == "cruel"
```

P2591 Concatenation of strings and string views

- add overloads of `operator+` between `string` and `string_view`
- in draft

```
string str;
string_view view;

// Appending
str + view;               // ERROR
str + std::string(view); // OK, but inefficient
str + view.data();        // Compiles, but BUG!
```

P2697 Interfacing `std::bitset` with `string_view`

- `bitset` is constructible from string
- but not `string_view`
- add a `string_view` constructor
- in draft

```
1 bitset b1{""sv};
```

P2497 Testing for success or failure of <charconv> functions

- in draft

```cpp
struct to_chars_result {
  char* ptr;
  errc ec;
  friend bool operator==(const to_chars_result&, const to_chars_result&) = default;
  constexpr explicit operator bool() const noexcept { return ec == errc{}; }
};

// before
auto [ptr, ec] = std::to_chars(p, last, 42);
if (ec == std::errc{}) // or !static_cast<bool>(ec)

// after
auto res = std::to_chars(p, last, 42);
if (res) { //use res.ptr }
```

P2497 Testing for success or failure of <charconv> functions

- in draft

```cpp
struct to_chars_result {
  char* ptr;
  errc ec;
  friend bool operator==(const to_chars_result&, const to_chars_result&) = default;
  constexpr explicit operator bool() const noexcept { return ec == errc{}; }
};

// before
auto [ptr, ec] = std::to_chars(p, last, 42);
if (ec == std::errc{}) // or !static_cast<bool>(ec)

// after
auto res = std::to_chars(p, last, 42);
if (res) { //use res.ptr }
```

P2497 Testing for success or failure of <charconv> functions

- in draft

```cpp
struct to_chars_result {
  char* ptr;
  errc ec;
  friend bool operator==(const to_chars_result&, const to_chars_result&) = default;
  constexpr explicit operator bool() const noexcept { return ec == errc{}; }
};

// before
auto [ptr, ec] = std::to_chars(p, last, 42);
if (ec == std::errc{}) // or !static_cast<bool>(ec)

// after
auto res = std::to_chars(p, last, 42);
if (res) { //use res.ptr }
```

P2497 Testing for success or failure of <charconv> functions

- in draft

```cpp
struct to_chars_result {
  char* ptr;
  errc ec;
  friend bool operator==(const to_chars_result&, const to_chars_result&) = default;
  constexpr explicit operator bool() const noexcept { return ec == errc{}; }
};

// before
auto [ptr, ec] = std::to_chars(p, last, 42);
if (ec == std::errc{}) // or !static_cast<bool>(ec)

// after
auto res = std::to_chars(p, last, 42);
if (res) { //use res.ptr }
```

P2587 `to_string` or not `to_string`

- `to_string` for floats is broken
  - precision issues
  - locale confusion
- Arithmetic overloads of `to_string` and `to_wstring` use `std::format`
- in draft

```
1  auto loc = std::locale("uk_UA.UTF-8");
2  std::locale::global(loc);
3  std::cout.imbue(loc);
4  setlocale(LC_ALL, "C");
5
6  std::cout << "iostreams:\n";
7  std::cout << 1234 << "\n";
8  std::cout << 1234.5 << "\n";
9
10 std::cout << "\nto_string:\n";
11 std::cout << std::to_string(1234) << "\n";
12 std::cout << std::to_string(1234.5) << "\n";
13 //iostreams:
14 //1 234
15 //1 234,5
16
17 //to_string:
18 //1234
19 //1234.500000
```

Library - Format and I/O Additions

P2757 Type-checking format args

expression	result
`format("{:d}", "I am not a number")`	compile err (invalid specifier for strings)
`format("{:7^*}", "hello")`	compile err (should be `*^7`)
`format("{:>10}", "hello")`	ok
`format("{0:>{1}}", "hello", 10)`	ok
`format("{0:>{2}}", "hello", 10)`	compile err (argument 2 is out of bounds)
`format("{:>{}}", "hello", "10")`	runtime err <– **wait why runtime?**

Type checking format args

```
1 format("{:>{}}", "hello", "10")  //dynamic width should be int, not char*
```

- libfmt actually doesn't have this error
- tldr; wording updates fix this problem for std::

P2918 Runtime format strings II

- compile-time format strings improved api
- runtime format api had only `vformat`
- easy to misuse api
- paper addresses this case
- in working draft

```
1 //before
2 std::string str = translate("The answer is {}.");
3 std::string msg = std::vformat(str, std::make_format_args(42));
4 //after
5 std::format(std::runtime_format(str), 42);
```

P2918 Runtime format strings II

- compile-time format strings improved api
- runtime format api had only `vformat`
- easy to misuse api
- paper addresses this case
- in working draft

```
1 //before
2 std::string str = translate("The answer is {}.");
3 std::string msg = std::vformat(str, std::make_format_args(42));
4 //after
5 std::format(std::runtime_format(str), 42);
```

P2510 Formatting pointers

- in draft

```
1 // zero fill width 18 - hex lower case
2 format("{:018}", ptr); // 0x00007ffe0325c4e4
3 // no fill -- upper case hex
4 format("{:P}", ptr);   // 0X7FFE0325C4E4
```

P2845 Formatting of std::filesystem::path

- supports unicode paths
- in draft

```
#include <filesystem>
filesystem::path p("/usr/bin");
print("{}", p);

p = filesystem::path("multi\nline");
cout << std::format("{}", p); // multi
                              // line
//debug format
cout << std::format("{:?}", p);  //"multi\nline"
```

P3142 Printing Blank Lines with `println`

- in draft

```
println(""); //c++23
println();   //new and same as above
```

Library - Optional Additions

P3168 Give std::optional Range Support

- turns optional into range
- similar to `single_view`
- https://github.com/beman-project/Optional26
- https://godbolt.org/z/9xoP5Tq5P

```cpp
using opt_int = beman::optional26::optional<int>;

// range-for loop over C++26 optional
opt_int empty_opt{};
for (const auto& i : empty_opt) {
    std::print("not executed: opt = {}\n", i);
}

opt_int opt{26};
for (const auto& i : opt) {
     std::print("executed: opt = {}\n", i);
}
```

P3168 Give std::optional Range Support

- turns optional into range
- similar to `single_view`
- https://github.com/beman-project/Optional26
- https://godbolt.org/z/9xoP5Tq5P

```cpp
using opt_int = beman::optional26::optional<int>;

// range-for loop over C++26 optional
opt_int empty_opt{};
for (const auto& i : empty_opt) {
    std::print("not executed: opt = {}\n", i);
}

opt_int opt{26};
for (const auto& i : opt) {
     std::print("executed: opt = {}\n", i);
}
```

P3168 Give std::optional Range Support

- turns optional into range
- similar to `single_view`
- https://github.com/beman-project/Optional26
- https://godbolt.org/z/9xoP5Tq5P

```cpp
using opt_int = beman::optional26::optional<int>;

// range-for loop over C++26 optional
opt_int empty_opt{};
for (const auto& i : empty_opt) {
    std::print("not executed: opt = {}\n", i);
}

opt_int opt{26};
for (const auto& i : opt) {
     std::print("executed: opt = {}\n", i);
}
```

P2885 optional<T&>

```
class Cat{};
//pre26
Cat* find_cat(std::string);
shared_ptr<Cat> find_cat(std::string);

Cat* thecat = find_cat("fluffy");
if (thecat != nullptr) { /* process */};

//post26
optional<Cat&> find_cat(std::string);
find_cat("fluffy").and_then([](Cat& thecat){/* process */};)
                  .or_else([]() {/* cat not found */};);
```

P2885 optional<T&>

```
1  class Cat{};
2  //pre26
3  Cat* find_cat(std::string);
4  shared_ptr<Cat> find_cat(std::string);
5
6  Cat* thecat = find_cat("fluffy");
7  if (thecat != nullptr) { /* process */};
8
9  //post26
10 optional<Cat&> find_cat(std::string);
11 find_cat("fluffy").and_then([](Cat& thecat){/* process */};)
12                   .or_else([]() {/* cat not found */};);
```

Library - Containers

P0843 `inplace_vector`

- 'drop in' replacement for vector
- like `boost::static_vector`
- no allocators, size fixed at compile time
- `push_back` that exceeds capacity causes `bad_alloc`
- https://github.com/beman-project/InplaceVector
- in draft

```
template <class T, size_t N>
class inplace_vector;

inplace_vector<int, 1> vec;
vec.push_back(1);
vec.push_back(2); //bad alloc thrown
```

P0843 `inplace_vector`

- 'drop in' replacement for vector
- like `boost::static_vector`
- no allocators, size fixed at compile time
- `push_back` that exceeds capacity causes `bad_alloc`
- https://github.com/beman-project/InplaceVector
- in draft

```
1 template <class T, size_t N>
2 class inplace_vector;
3
4 inplace_vector<int, 1> vec;
5 vec.push_back(1);
6 vec.push_back(2); //bad alloc thrown
```

inplace_vector II

```cpp
// exceptionless api for handling overruns
template <class... Args>
constexpr pointer try_emplace_back(Args&&... args);
constexpr pointer try_push_back(T&& x);
template <container-compatible-range<T> R>
constexpr ranges::borrowed_iterator_t<R> try_append_range(R&& rg);

inplace_vector<int, 1> vec;
vec.push_back(1);
auto ptr vec.try_push_back(2); //no exception
```

inplace_vector II

```cpp
// exceptionless api for handling overruns
template <class... Args>
constexpr pointer try_emplace_back(Args&&... args);
constexpr pointer try_push_back(T&& x);
template <container-compatible-range<T> R>
constexpr ranges::borrowed_iterator_t<R> try_append_range(R&& rg);

inplace_vector<int, 1> vec;
vec.push_back(1);
auto ptr vec.try_push_back(2); //no exception
```

P2447 std::span over an initializer list

- `span<const int>` replaces `const vector<int>&`
- except for initializer list

```
1 int take(const std::vector<int>& v) { return v[0] + v.size(); } //c++17
2 int take(std::span<const int> v) { return v[0] + v.size(); }    //c++20
3
4 take( {1,2,3} ); //fail, ok after P2447
```

P2447 std::span over an initializer list

- `span<const int>` replaces `const vector<int>&`
- except for initializer list

```
1 int take(const std::vector<int>& v) { return v[0] + v.size(); } //c++17
2 int take(std::span<const int> v) { return v[0] + v.size(); }    //c++20
3
4 take( {1,2,3} ); //fail, ok after P2447
```

P2447 std::span over an initializer list

- `span<const int>` replaces `const vector<int>&`
- except for initializer list

```
1 int take(const std::vector<int>& v) { return v[0] + v.size(); } //c++17
2 int take(std::span<const int> v) { return v[0] + v.size(); }    //c++20
3
4 take( {1,2,3} ); //fail, ok after P2447
```

P2821 `span::at()`

- `vector`, `array`, etc all have `at()` in addition to `operator[]`
- `span` does not
- related change: `views_interface::at()`
- in draft

Associative Containers - misc improvements

- P2363 Extending associative containers with the remaining heterogeneous overloads
- P1901 Enabling the Use of `weak_ptr` as Keys in Unordered Associative Containers

Library - Ranges

- A Plan for C++26 Ranges
- `views::concat` (in draft)
- Take/Drop
  - `views::slice`
  - `views::take_exactly`
  - `views::drop_exactly`
- Adaptor compositions
  - `views::upto`
- …much more…

P2542 `views::concat`

- in draft
- implementation
  https://github.com/huixie90/cpp_papers/tree/main/impl/concat

```cpp
std::vector<int> v1{1,2,3}, v2{4,5}, v3{};
std::array  a{6,7,8};
auto s = std::views::single(9);
std::print("{}\n",
           std::views::concat(v1, v2, v3, a, s));
// output:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

P3060 Add `std::views::upto(n)`

- still in work

```
std::vector rng(5, 0);
// auto res1 = views::iota(0, ranges::size(rng)); // does not compile

auto res2 = iota(range_size_t<decltype(rng)>{}, ranges::size(rng));
std::print("{}", res2); // [0, 1, 2, 3, 4]

//after
std::vector rng(5, 0);
std::print("{}", views::upto(ranges::size(rng))); // [0, 1, 2, 3, 4]
```

P3060 Add `std::views::upto(n)`

- still in work

```cpp
std::vector rng(5, 0);
// auto res1 = views::iota(0, ranges::size(rng)); // does not compile

auto res2 = iota(range_size_t<decltype(rng)>{}, ranges::size(rng));
std::print("{}", res2); // [0, 1, 2, 3, 4]

//after
std::vector rng(5, 0);
std::print("{}", views::upto(ranges::size(rng))); // [0, 1, 2, 3, 4]
```

P3230 `views::(take|drop)_exactly`

- still in work
- Example implementation

```
1 auto r0 = views::iota(0) | views::take_exactly(5);
2 print("{}\n", r0); [0, 1, 2, 3, 4]
3
4 auto r1 = views::iota(0) | views::drop_exactly(5);
5 print("{}\n", r1 | std::views::take_exactly(5)); [5, 6, 7, 8, 9]
```

P3230 `views::(take|drop)_exactly`

- still in work
- Example implementation

```
1 auto r0 = views::iota(0) | views::take_exactly(5);
2 print("{}\n", r0); [0, 1, 2, 3, 4]
3
4 auto r1 = views::iota(0) | views::drop_exactly(5);
5 print("{}\n", r1 | std::views::take_exactly(5)); [5, 6, 7, 8, 9]
```

P2727 `std::iterator_interface`

- in LEWG
- Writing STL iterators is difficult & tedious
  - Iterators have numerous typedefs and operations
  - typically the operations of a given iterator
- Also good for adapting iterators
- Boost stl interfaces

P3216 `views::slice`

- in LEWG
- `slice(M, N)` is equivalent to `views::drop(M) | views::take(N - M)`

```
1 auto ints = views::iota(0);
2 auto fifties = ints | views::slice(50, 60);
3 println("{} ", fifties); // prints [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
```

P3216 `views::slice`

- in LEWG
- `slice(M, N)` is equivalent to `views::drop(M) | views::take(N - M)`

```
1 auto ints = views::iota(0);
2 auto fifties = ints | views::slice(50, 60);
3 println("{} ", fifties); // prints [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
```

Library - Utilities

P1759 Native handles and file streams

```
// No need to use platform-specific APIs to open the file
{
    std::ofstream of("~/foo.txt");
    auto lm = last_modified(of.native_handle());
    of << std::chrono::format("%c", lm) << '\n';
    // RAII does ownership handling for us
}
```

P2637 Member visit and apply

```
std::variant<int, string> value;
//before
std::visit(overload{
  [](int i){ std::print("i={}\n", i); },
  [](std::string s){ std::print("s={:?}\n", s); }}, value);
//after
value.visit(overload{
  [](int i){ std::print("i={}\n", i); },
  [](std::string s){ std::print("s={:?}\n", s); }});
```

P2637 Member visit and apply

```
std::variant<int, string> value;
//before
std::visit(overload{
  [](int i){ std::print("i={}\n", i); },
  [](std::string s){ std::print("s={:?}\n", s); }}, value);
//after
value.visit(overload{
  [](int i){ std::print("i={}\n", i); },
  [](std::string s){ std::print("s={:?}\n", s); }});
```

Library - General Math Support

P0543 Saturation arithmetic

- addition, subtraction, multiplication, and division provided
- in draft

```
#include <numeric>
template<class T>
   constexpr T add_sat(T x, T y) noexcept;
 template<class T>
   constexpr T sub_sat(T x, T y) noexcept;
 template<class T>
   constexpr T mul_sat(T x, T y) noexcept;
 template<class T>
   constexpr T div_sat(T x, T y) noexcept;
 template<class T, class U>
   constexpr T saturate_cast(U x) noexcept;
```

Saturation arithmetic usage & theory

- Saturation arithmetic

```
//range of values is from −100 to 100
    60 + 30 → 90.
    60 + 43 → 100. (not the expected 103.)
    (60 + 43) − (75 + 25) → 0. (not the expected 3.) (100 − 100 → 0.)
    10 × 11 → 100. (not the expected 110.)
    99 × 99 → 100. (not the expected 9801.)
    30 × (5 − 1) → 100. (not the expected 120.) (30 × 4 → 100.)
    (30 × 5) − (30 × 1) → 70. (not the expected 120. not the previous 100.) (100 − 30 → 70.)
```

P3103 More Bitset Operations

<bit> function	Proposed bitset member
`std::has_single_bit(T)`	`one()`
`std::countl_zero(T)`	`countl_zero()`
	`countl_zero(size_t)`
`std::countl_one(T)`	`countl_one()`
	`countl_one(size_t)`

More Bitset Operations

<bit> function	Proposed bitset member
`std::countr_zero(T)`	`countr_zero()`
	`countr_zero(size_t)`
`std::countr_one(T)`	`countr_one()`
	`countr_one(size_t)`
`std::rotl(T, int)`	`rotl(size_t)`
`std::rotr(T, int)`	`rotr(size_t)`
	`reverse()`

More bitset Operations

- in draft

```
bitset<128> bits;
for (size_t i = 0; i != 128; ++i) {
    i += bits.countr_zero(i);
    if (i == 128) break;
    // ...
}
```

Library - Linear Algebra and mdspan

- P2630 submdspan
- Adding Blas based Linear Algebra
- P3029 New CTAD for std::span and std::mdspan with integral constants

P1673 A free function linear algebra interface based on the BLAS

- elementwise vector sums
- multiplying all elements of a vector or matrix by a scalar
- 2-norms and 1-norms of vectors
- vector-vector, matrix-vector, and matrix-matrix products (contractions)
- low-rank updates of a matrix
- triangular solves with one or more “right-hand side” vectors
- generating and applying plane (Givens) rotations
- Blas1 - Blas3 algorithms

Linalg example

```
constexpr size_t N = 40;
constexpr size_t M = 20;

std::vector<double> A_vec(N*M);
std::vector<double> x_vec(M);
std::array<double, N> y_vec(N);

mdspan A(A_vec.data(), N, M);
mdspan x(x_vec.data(), M);
mdspan y(y_vec.data(), N);

// y = 0.5 * y + 2 * A * x
linalg::matrix_vector_product(std::execution::par,
      linalg::scaled(2.0, A), x,
      linalg::scaled(0.5, y), y);
```

`matrix_vector_product` overloads

```
1  // [linalg.algs.blas2.gemv],
2  // general matrix-vector product
3  template<in-matrix InMat,
4           in-vector InVec,
5           out-vector OutVec>
6  void matrix_vector_product(InMat A,
7                              InVec x,
8                              OutVec y);
9  template<class ExecutionPolicy,
10          in-matrix InMat,
11          in-vector InVec,
12          out-vector OutVec>
13 void matrix_vector_product(ExecutionPolicy&& exec,
14                             InMat A,
15                             InVec x,
16                             OutVec y);
```

Library - Concurrency

P2300 `std::execution` (aka Senders and Receivers)

- provides structured concurrency framework
- several follow-on papersin work
- on-going work https://github.com/cplusplus/sender-receiver

P2545 Read-Copy Update

- synchronization mechanism for linked data structures
- frequently read, but seldom updated
- not provide mutual exclusion
- user to schedule specified actions

P2530 Hazard Pointers

- single-writer multi-reader pointer
- owned one thread at any time
- only the owner can set its value
- any number of threads can read
- deferred reclaimation

Library - `std::simd`

`std::simd` constructors

```
template<class U>
constexpr simd(U&& value) noexcept;

template<class U, class UAbi>
constexpr explicit simd(const simd<U, UAbi>&) noexcept;

template<class G> constexpr explicit simd(G&& gen) noexcept;

template<contiguous_iterator It, class Flags = element_aligned_tag>
constexpr simd(const It& first, Flags = {})

template<class U, class Flags = element_aligned_tag>
void copy_from(const U* mem, Flags = {}) &&

template<class U, class Flags = element_aligned_tag>
void copy_to(U* mem, Flags = {}) const &&;
```

generator function construction

```
#include <experimental/simd>
namespace stdx = std::experimental;
using intv  = stdx::fixed_size_simd<int,8>;

std::random_device rd;  // a seed source for the random number engine
std::mt19937 gen(rd()); // mersenne_twister_engine seeded with rd()
std::uniform_int_distribution<> distrib(1, 20);

intv a([&gen, &distrib](int){ return distrib(gen);});
// use a ...
```

- https://godbolt.org/z/8WG6qq78s

algorithms

- `any_of`, `all_of`, `none_of`
- `reduce`, `reduce_min`, `hmax`, `abs`

reduce

```
namespace stdx = std::experimental;
using intv  = stdx::fixed_size_simd<int,8>;

int main()
{
    std::array<int,8> a_data = {-1, 2, 3, 4, 5, 6, 7, -8};
    intv a;
    a.copy_from( a_data.begin(), stdx::vector_aligned );

    //returns fixed_size_simd<int,8>
    auto a_abs = abs(a);
    int sum = reduce( a_abs );
    int min = reduce_min( a_abs );
    int max = reduce_max( a_abs );

    // 36 1 8
    print("{} {} {}", sum, min, max);
}
```

- https://godbolt.org/z/7W4sxaoEq

Library - constexpr all the things

- P2562 constexpr Stable Sorting
- P1383 R2 More constexpr for cmath and complex

`atexit()`

> *"Form and Function are one" – Frank Loyld Wright*

- http://crystalclearsoftware.com/2024cppcon/cpp26preview.pdf
- https://en.cppreference.com/w/cpp/compiler_support
- https://en.cppreference.com/w/cpp/utility/program/atexit