C++ Committee Trip Report

Kona 2022

Dr. Matthias Kretz

GSI Helmholtz Centre for Heavy Ion Research

2022-11-23

The main task of the Kona meeting

NB comment processing

- Comments from *national bodies* after review of the *committee draft* must be answered.
- Basically, this is the bugfix phase of the standard.
- Comments must be processed & answered by the next meeting, where we finalize C++23.

Highlights

- P2644R1 Lifetime of temporaries in range-based `for` extended
- P2589R1 `static operator[]` (for consistency with `static operator()`)
- P2647R1 Permit `static constexpr` variables in `constexpr` functions
- P2564R3 `consteval` needs to propagate up
- P2505R5 “Monadic Functions for std::expected”

P2644R1 Lifetime of temporaries in range-based for extended

```cpp
std::vector<int> f();
for (auto i : range) { /*...*/ } // OK, no temporary
for (auto i : f()) { /*...*/ } // OK, temporary lives long enough
for (auto i : f() | std::views::drop(1)) { /*...*/ } // undefined behavior
```

```
for (for-range-declaration : for-range-initializer ) statement
```

is equivalent to

```
{
  auto &&range = for-range-initializer;
  auto begin = begin-expr;
  auto end = end-expr;
  for ( ; begin != end; ++begin ) {
    for-range-declaration = *begin;
    statement
  }
}
```

P2644R1 Lifetime of temporaries in range-based `for` extended

```cpp
std::vector<int> f();
for (auto i : range) { /*...*/ } // OK, no temporary
for (auto i : f()) { /*...*/ } // OK, temporary lives long enough
for (auto i : f() | std::views::drop(1)) { /*...*/ } // undefined behavior
```

```
  for (for-range-declaration : for-range-initializer ) statement
```

is equivalent to

```
  {
    auto &&range = for-range-initializer;
    auto begin = begin-expr;
    auto end = end-expr;
    for ( ; begin != end; ++begin ) {
      for-range-declaration = *begin;
      statement
    }
  }
```

scientific computing related (C++26)

1. linear algebra (free functions on `mdspan`)
   - hard to specify without specifying too much
2. `submdspan`
   - introduces vocabulary for a strided index range
3. `std::matrix<...>`
   - vectors are now simply $N \times 1$ or $1 \times N$ matrices

scientific computing related (C++26)

1. linear algebra (free functions on `mdspan`)
   - hard to specify without specifying too much
2. `submdspan`
   - introduces vocabulary for a strided index range
3. `std::matrix<...>`
   - vectors are now simply $N \times 1$ or $1 \times N$ matrices

scientific computing related (C++26)

1. linear algebra (free functions on `mdspan`)
   - hard to specify without specifying too much
2. `submdspan`
   - introduces vocabulary for a strided index range
3. `std::matrix<...>`
   - vectors are now simply $N \times 1$ or $1 \times N$ matrices

further progress

1. ongoing work on improving template metaprogramming, esp. the ergonomics of *packs*
2. `#embed`
3. allowing `static_assert(false)`
4. debugging support
   - `std::breakpoint()`
   - `std::breakpoint_if_debugging()`
   - `std::is_debugger_present()`
5. pattern matching exploration continued
6. plan for new standard: C++ ecosystem
7. aggregates are named tuples

further progress

1. ongoing work on improving template metaprogramming, esp. the ergonomics of *packs*
2. `#embed`
3. allowing `static_assert(false)`
4. debugging support
   - `std::breakpoint()`
   - `std::breakpoint_if_debugging()`
   - `std::is_debugger_present()`
5. pattern matching exploration continued
6. plan for new standard: C++ ecosystem
7. aggregates are named tuples

Aggregates are named tuples (C++26)

```
struct foo {
  int id;
  float x, y, z;
};

static_assert(std::tuple_size_v<foo> == 4);
static_assert(std::same_as<std::tuple_element_t<0, foo>, int>);
static_assert(std::same_as<std::tuple_element_t<1, foo>, float>);
static_assert(std::same_as<std::tuple_element_t<2, foo>, float>);

void f(foo x) {
  assert(std::get<0>(foo) == foo.id);
  assert(std::get<3>(foo) == foo.z);
  assert(&std::get<3>(foo) == &foo.z);
}
```

Aggregates are named tuples (C++26)

```cpp
struct foo {
  int id;
  float x, y, z;
};

static_assert(std::tuple_size_v<foo> == 4);
static_assert(std::same_as<std::tuple_element_t<0, foo>, int>);
static_assert(std::same_as<std::tuple_element_t<1, foo>, float>);
static_assert(std::same_as<std::tuple_element_t<2, foo>, float>);

void f(foo x) {
  assert(std::get<0>(foo) == foo.id);
  assert(std::get<3>(foo) == foo.z);
  assert(&std::get<3>(foo) == &foo.z);
}
```

You will be able to use a `struct` where a `std::tuple` is required now

Pattern matching (C++26?)

P2211R0 syntax:

```
struct FireBlasters {
  int intensity;
  bool operator==(const FireBlasters&) const = default;
};
enum Direction{ Left, Right };
struct Move {
  Direction direction;
  bool operator==(const FireBlasters&) const = default;
};
using Command = std::variant<FireBlasters, Move>;

std::string cmdToStringV2(Command cmd) {
  return inspect(cmd) {
    <FireBlasters> [i] => std::format("Fire Blasters with power {}", i);
    <Move> [case Left] => std::string("Move Left");
    <Move> [case Right] => std::string("Move Right");
  };
}
```

my paper P2600R0 in EWG

EWG	The (language) Evolution Working Group
P2600R0	my paper "A minimal ADL restriction to avoid ill-formed template instantiation"

- related to what I presented on ADL in this group
- prerequisite to further evolution of operator overloading in C++
- which, in turn, is a prerequisite for better integration of `simd`

- Sadly, EWG's schedule was full and I didn't get a chance to present.
- I got no feedback on the paper.

my paper P2600R0 in EWG

EWG	The (language) Evolution Working Group
P2600R0	my paper "A minimal ADL restriction to avoid ill-formed template instantiation"

- related to what I presented on ADL in this group
- prerequisite to further evolution of operator overloading in C++
- which, in turn, is a prerequisite for better integration of `simd`

- Sadly, EWG's schedule was full and I didn't get a chance to present.
- I got no feedback on the paper.

my paper P1928R1 in SG1

SG1	Subgroup on parallelism & concurrency
P1928R1	my paper “Merge data-parallel types from the Parallelism TS 2”

- wants to turn `std::experimental::simd` into `std::simd` for C++26

Three polls were taken and unanimously approved:

1. After significant experience with the TS, we recommend that the next version (the TS version with improvements) of `std::simd` target the IS (C++26)
2. We like all of the recommended changes to `std::simd` proposed in p1928r1 (Includes making all of `std::simd constexpr`, and dropping an ABI stable type)
3. Future papers and future revisions of existing papers that target `std::simd` should go directly to LEWG. (We do not believe there are SG1 issues with `std::simd` today.)

my paper P1928R1 in SG1

SG1	Subgroup on parallelism & concurrency
P1928R1	my paper “Merge data-parallel types from the Parallelism TS 2”

- wants to turn `std::experimental::simd` into `std::simd` for C++26

Three polls were taken and unanimously approved:

1. After significant experience with the TS, we recommend that the next version (the TS version with improvements) of `std::simd` target the IS (C++26)
2. We like all of the recommended changes to `std::simd` proposed in p1928r1 (Includes making all of `std::simd constexpr`, and dropping an ABI stable type)
3. Future papers and future revisions of existing papers that target `std::simd` should go directly to LEWG. (We do not believe there are SG1 issues with `std::simd` today.)

Intel’s papers P2638R0 and P2663R0

P2638R0 “Intel’s response to P1915R0 for `std::simd` parallelism in TS 2”
P2663R0 “Proposal to support interleaved complex values in `std::simd`”

- P2638R0 propose a few changes to the TS design
- more importantly, both papers propose more features, simplifying the use in more diverse fields

SG1 voted to go ahead with everything proposed, deferring further review to LEWG.

Intel's papers P2638R0 and P2663R0

P2638R0	"Intel's response to P1915R0 for `std::simd` parallelism in TS 2"
P2663R0	"Proposal to support interleaved complex values in `std::simd`"

- P2638R0 propose a few changes to the TS design
- more importantly, both papers propose more features, simplifying the use in more diverse fields

SG1 voted to go ahead with everything proposed, deferring further review to LEWG.

Intel’s papers P2638R0 and P2663R0

P2638R0	“Intel’s response to P1915R0 for `std::simd` parallelism in TS 2”
P2663R0	“Proposal to support interleaved complex values in `std::simd`”

- P2638R0 propose a few changes to the TS design
- more importantly, both papers propose more features, simplifying the use in more diverse fields

SG1 voted to go ahead with everything proposed, deferring further review to LEWG.

(memory-) safety in C++

hot 🔥 topic

P2676R0	“The Val Object Model” by Dave Abrahams, Sean Parent, Dimitri Racordon, David Sankel
P2687R0	“Design Alternatives for Type-and-Resource Safe C++” by Bjarne Stroustrup, Gabriel Dos Reis
evening session	“future of C++”

US companies are breaking away

- Google seems to place all their bets on *Carbon*
- Adobe, Microsoft, Bloomberg, …
  they all need an answer to the executive order calling out “C/C++” as memory unsafe.

(memory-) safety in C++ (2)

- goal: reduce *security* issues in C++ code (CVEs and CWEs)
- goal: make C++ easier to use (i.e. to write *correct* code)
- I believe there is consensus that we cannot only blame user errors

Two road blocks:

1. users must use newer compilers, language standards, and migrate to new facilities
2. making “unsafe” code ill-formed breaks compatibility (which is why C++ is regarded as just as unsafe as C — arguably unfairly so)

Is it still C++ then?

We cannot make C++ safe by default without breaking compatibility.

(memory-) safety in C++ (2)

- goal: reduce *security* issues in C++ code (CVEs and CWEs)
- goal: make C++ easier to use (i.e. to write *correct* code)
- I believe there is consensus that we cannot only blame user errors

Two road blocks:

1. users must use newer compilers, language standards, and migrate to new facilities
2. making "unsafe" code ill-formed breaks compatibility (which is why C++ is regarded as just as unsafe as C — arguably unfairly so)

Is it still C++ then?

We cannot make C++ safe by default without breaking compatibility.

(memory-) safety in C++ (2)

- goal: reduce *security* issues in C++ code (CVEs and CWEs)
- goal: make C++ easier to use (i.e. to write *correct* code)
- I believe there is consensus that we cannot only blame user errors

Two road blocks:

1. users must use newer compilers, language standards, and migrate to new facilities
2. making “unsafe” code ill-formed breaks compatibility (which is why C++ is regarded as just as unsafe as C — arguably unfairly so)

Is it still C++ then?

We cannot make C++ safe by default without breaking compatibility.

personal opinion

- “The Val Object Model” is worth a good look
- Can we integrate new parameter passing so that ...
  - users can pass values, avoiding the pitfalls of references and pointers?
  - local reasoning is maximized (better optimization, less incorrect code)?

- I’d like to focus a lot more of my effort on this topic, but I’m already overworked.
- How can the science C++ userbase take a more active role here? 🤔

personal opinion

- “The Val Object Model” is worth a good look
- Can we integrate new parameter passing so that …
  - users can pass values, avoiding the pitfalls of references and pointers?
  - local reasoning is maximized (better optimization, less incorrect code)?

- I’d like to focus a lot more of my effort on this topic, but I’m already overworked.
- How can the science C++ userbase take a more active role here? 🤔

Meeting Rooms

Projecting ephemeral parts | An end to proxy problems

```
type Angle {
  public var radians: Double

  public property degrees: Double {
    inout {
      var d = radians * 180.0 / Double.pi
      yield &d
      radians = d * Double.pi / 180.0
    }
    set(newValue) {
      radians = newValue * Double.pi / 180.0
    }
  }
}
```

Meeting Rooms

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Around the Hotel

Kona Misc.

Kona Misc.

Kona Misc.

Kona Misc.

Kona Misc.

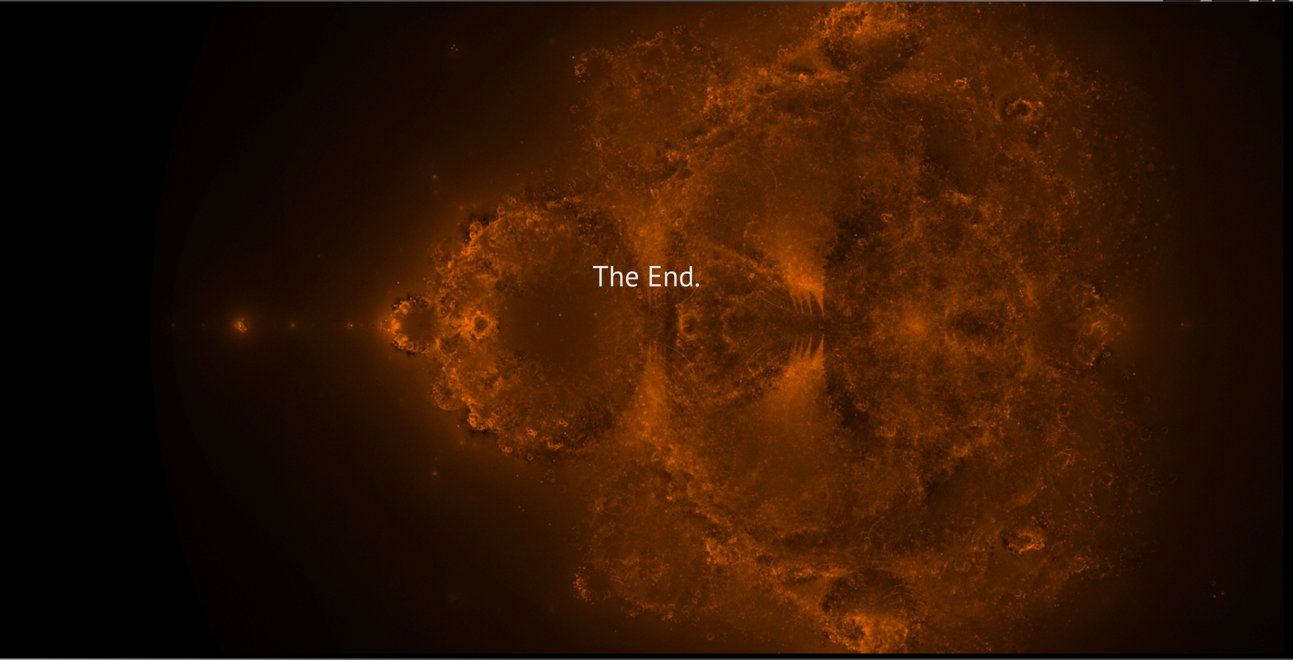
Kona Misc.

Kona Misc.

Kona Misc.

Kona Misc.

Flying back Home



The End.