Introduction to C++ Reflection

Jolly Chen

July 25, 2024
CAES, EEMCS, University of Twente (Enschede, The Netherlands)
EP-SFT, CERN (Geneva, Switzerland)

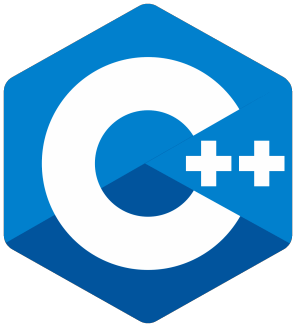


Table of Contents

1 Main Proposal Paper (P2996)

Current Status

1 Main Proposal Paper (P2996)

- ***Reflection*** = Introspection + Metaprogramming + Code Generation
- Main proposal paper: **P2996** (Revision 4: 26-06-2024)
  - Authors: Wyatt Childers, Peter Dimov, Dan Katz, Barry Revzin, Andrew Sutton, Faisal Vali, Daveed Vandevoordde
  - Planned for C++26
  - Design-approved, wording in-review (last phase)[1]
- **2.5** experimental implementations available:
  - Fork of Clang: `https://github.com/bloomberg/clang-p2996` (also on GodBolt)
  - EDG: `https://godbolt.org/z/13anqE1Pa`
    - NVIDIA HPC SDK `nvc++` 24.3+ with `-std=c++23 --reflection` (GodBolt)

[1] https://www.reddit.com/r/cpp/comments/1dwc7f2/202406_st_louis_iso_c_committee_trip_report/

Main Features

1 Main Proposal Paper (P2996)

`^x`	"*Lift*" operand `x` to a **reflection** value
`[: refl :]`	"*Splice*" a reflection to **produce grammatical elements**

Reflections return an opaque type `std::meta::info` for forward compatibility

`std::meta` Methods

P2996 provides the `std::meta` namespace with **traits-equivalent** functions, **query** functions, and functions to **construct basic structures** using reflected code [2]:

Traits-equivalent methods:

```
type_is_void, type_is_same, type_is_member_function_pointer,
type_is_copy_constructible, type_is_trivially_default_constructible,
...
```

Reflection query methods:

- **Reflection**: `name_of, type_of, is_public, is_private, is_function, ...`
- **Members**: `members_of, nonstatic_data_members_of, ...`
- **Layout**: `offset_of, size_of, alignment_of, bit_offset_of, ...`

[2] Some of the naming might change

`std::meta` Methods

Construct basic structures:

```
struct S;
constexpr auto s_int_refl = define_class(^S, {
  data_member_spec(^int, {.name="i", .alignment=64}),
  data_member_spec(^float, {.name="j", .alignment=64}),
  data_member_spec(^double, {.name="k"})
});
```

For now, only data member reflections are supported (via `data_member_spec`) but the API takes in a range of `std::meta::info` anticipating expanding this in the near future.

Back and Forth

1 Main Proposal Paper (P2996): Examples

The proposal includes several examples:

```
1 constexpr auto r = ^int;
2 typename[:r:] x = 42;        // Same as: int x = 42;
3 typename[:^char:] c = '*'; // Same as: char c = '*';
```

On Compiler Explorer: EDG, Clang

Enum to String

1 Main Proposal Paper (P2996): Examples

```
1  template <typename E> requires std::is_enum_v<E>
2    constexpr std::string enum_to_string(E value) {
3      template for (constexpr auto e : std::meta::enumerators_of(^E)) {
4        if (value == [:e:]) {
5          return std::string(std::meta::name_of(e));
6        }
7    }
8    return "<unnamed>";
9  }
10
11 enum Color { red, green, blue };
12 static_assert(enum_to_string(Color::red) == "red");
13 static_assert(enum_to_string(Color(42)) == "<unnamed>");
```

template for is part of (P1306)

On Compiler Explorer: EDG, Clang

What It Is and What It Is Not

2 Summary of P2996

P2996 includes:

- Generating class members
- Querying properties of a reflected type

Does not include:

- Generating (member) functions
- Token injection
- “Typeful reflection”
- Reflection of expressions i.e., `^(expr)`
- Splicing constructors/destructors

What It Is and What It Is Not

2 Summary of P2996

P2996 includes:

- Generating class members
- Querying properties of a reflected type

Does not include:

- Generating (member) functions (P3157)
- Token injection (P3294)
- "Typeful reflection"
- Reflection of expressions i.e., `^(expr)`
- Splicing constructors/destructors

For some of these, a separate proposal is already in the works

Table of Contents

3 Token Injection (P3294)

Token Injection (P3294) [Andrei Alexandrescu, Barry Revzin, Daveed Vandevoorde]

3 Token Injection (P3294): Main Features

P3294 adds code injection via token sequences, which can be created with:

```
^{ balanced-brace-tokens }
```

The type of a token sequence is `std::meta::info`.

They can be queued up to be injected at the end of the current constant evaluation using:

```
consteval {
    std::meta::queue_injection(^{ ... } )
}
```

Implementation available in experimental EDG compiler

Example: SoA with AoS Interface

3 Token Injection (P3294)

Together with **token injection** from P3294, we can use a *Structure of Arrays (SoA)* as a *Array of Structures (AoS)* with native C++ syntax!

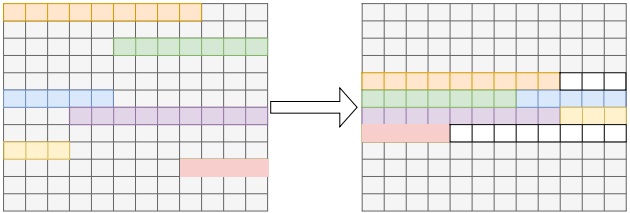
AoS:	x_1	y_1	z_1	x_2	y_2	z_2	x_3	y_3	z_3
SoA:	x_1	x_2	x_3	y_1	y_2	y_3	z_1	z_2	z_3

See: `https://godbolt.org/z/eGMGWjnYx`

Next Challenge 1: Contiguous Allocation

3 Token Injection (P3294): Example: SoA with AoS Interface

For locality and transfers to the GPU,
we ideally want to have everything allocated in one chunk



Use `std::span`? Ensure alignment[a] with padding? ...

[a] `https://developer.ibm.com/articles/pa-dalign`

Next Challenge 2: More Complicated Structures

3 Token Injection (P3294): Example: SoA with AoS Interface

How do we convert to/from more complicated structures?

- AoSoA

```
struct S { std::vector<float> A; };
std::vector<S> AoSoA;
```

- SoAoS

```
struct S { float x; };
    struct SoAoS { std::vector<S> A; };
```

- Jagged arrays
- ...?

Next Challenge 3: (Automatic) Optimal Layout

3 Token Injection (P3294): Example: SoA with AoS Interface

- Which information do we need to choose the optimal memory layout?
  - Maybe access pattern, conversion cost, system specifications, …
- Given this information, can we *automatically* decide the optimal layout?
  - Via performance modelling?

This will be the main topic for my PhD

Table of Contents

4 Conclusion & Outlook

Conclusion & Outlook

- C++ reflection will *simplify metaprogramming* and add *code generation*
- P2996 aims for minimum viable specification
  - More powerful features (code generation) not until C++29 or later
- Creating portable, performant, and user-friendly data structures would become a lot easier with C++ reflection