C++ Reflection: Back on Track

David Olsen, Compiler Engineer, NVIDIA

CppNorth, 22 July 2024

```cpp
struct app_options {
  int iterations = 1'000;
  int size = 25'000;
  bool verbose = false;
  std::string inputFile = "input.dat";
};

int main(int argc, char** argv) {
  app_options opts = {};
  std::vector<std::string_view> args(argv + 1, argv + argc);
  for (auto it = args.begin(); it != args.end(); ++it) {
    if (*it == "--iterations"sv) {
      std::ispanstream(*++it) >> opts.iterations;
    } else if (*it == "--size"sv) {
      std::ispanstream(*++it) >> opts.size;
    } else if (*it == "--verbose"sv) {
      opts.verbose = true;
    } else if (*it == "--inputFile"sv) {
      opts.inputFile = *++it;
    }
  }
```

https://godbolt.org/z/Wxh13cvzj

```cpp
struct app_options {
  int iterations = 1'000;
  int size = 25'000;
  bool verbose = false;
  std::string inputFile = "input.dat";
};

int main(int argc, char** argv) {
  app_options opts = {};
  std::vector<std::string_view> args(argv + 1, argv + argc);
  for (auto it = args.begin(); it != args.end(); ++it) {
    if (*it == "--iterations"sv) {
      std::ispanstream(*++it) >> opts.iterations;
    } else if (*it == "--size"sv) {
      std::ispanstream(*++it) >> opts.size;
    } else if (*it == "--verbose"sv) {
      opts.verbose = true;
    } else if (*it == "--inputFile"sv) {
      opts.inputFile = *++it;
    }
  }
```

https://godbolt.org/z/Wxh13cvzj

```cpp
struct app_options {
  int iterations = 1'000;
  int size = 25'000;
  bool verbose = false;
  std::string inputFile = "input.dat";
};

int main(int argc, char** argv) {
  app_options opts = {};
  std::vector<std::string_view> args(argv + 1, argv + argc);
  for (auto it = args.begin(); it != args.end(); ++it) {
    if (*it == "--iterations"sv) {
      std::ispanstream(*++it) >> opts.iterations;
    } else if (*it == "--size"sv) {
      std::ispanstream(*++it) >> opts.size;
    } else if (*it == "--verbose"sv) {
      opts.verbose = true;
    } else if (*it == "--inputFile"sv) {
      opts.inputFile = *++it;
    }
  }
```

https://godbolt.org/z/Wxh13cvzj

```cpp
struct app_options {
  int iterations = 1'000;
  int size = 25'000;
  bool verbose = false;
  std::string inputFile = "input.dat";
};

int main(int argc, char** argv) {
  app_options opts = {};
  std::vector<std::string_view> args(argv + 1, argv + argc);
  for (auto it = args.begin(); it != args.end(); ++it) {
    if (*it == "--iterations"sv) {
      std::ispanstream(*++it) >> opts.iterations;
    } else if (*it == "--size"sv) {
      std::ispanstream(*++it) >> opts.size;
    } else if (*it == "--verbose"sv) {
      opts.verbose = true;
    } else if (*it == "--inputFile"sv) {
      opts.inputFile = *++it;
    }
  }
```

https://godbolt.org/z/Wxh13cvzj

```cpp
struct app_options {
  int iterations = 1'000;
  int size = 25'000;
  bool verbose = false;
  std::string inputFile = "input.dat";
};

int main(int argc, char** argv) {
  app_options opts = {};
  std::vector<std::string_view> args(argv + 1, argv + argc);
  for (auto it = args.begin(); it != args.end(); ++it) {
    if (*it == "--iterations"sv) {
      std::ispanstream(*++it) >> opts.iterations;
    } else if (*it == "--size"sv) {
      std::ispanstream(*++it) >> opts.size;
    } else if (*it == "--verbose"sv) {
      opts.verbose = true;
    } else if (*it == "--inputFile"sv) {
      opts.inputFile = *++it;
    }
  }
```

https://godbolt.org/z/Wxh13cvzj

```cpp
struct app_options {
  int iterations = 1'000;
  int size = 25'000;
  bool verbose = false;
  std::string inputFile = "input.dat";
};

int main(int argc, char** argv) {
  app_options opts = {};
  std::vector<std::string_view> args(
                            argv + 1, argv + argc);
  for (auto it = args.begin();
       it != args.end(); ++it) {
    if (*it == "--iterations"sv) {
      if (it + 1 == args.end()) {
        std::cerr <<
        "Missing value for option --iterations\n";
      } else {
        std::ispanstream iss(*++it);
        iss >> opts.iterations;
        if (!iss) std::cerr << "Invalid value \""
         << *it << "\" for option --iterations\n";
      }
    } else if (*it == "--size"sv) {
      if (it + 1 == args.end()) {
        std::cerr <<
              "Missing value for option --size\n";
      } else {
        std::ispanstream iss(*++it);
        iss >> opts.size;
        if (!iss) std::cerr << "Invalid value \""
              << *it << "\" for option --size\n";
      }
    } else if (*it == "--verbose"sv) {
      opts.verbose = true;
    } else if (*it == "--inputFile"sv) {
      if (it + 1 == args.end()) {
        std::cerr <<
         "Missing value for option --inputFile\n";
      } else {
        opts.inputFile = *++it;
      }
    }
  }
```

https://godbolt.org/z/qTo8cavYe

```
struct app_options {
  int iterations = 1'000;
  int size = 25'000;
  bool verbose = false;
  std::string inputFile = "input.dat";
};

int main(int argc, char *argv[]) {
  app_options opts =
        parse_options<app_options>(argc, argv);
```

https://godbolt.org/z/4s5685qYv

```cpp
struct app_options {
  int iterations = 1'000;
  int size = 25'000;
  bool verbose = false;
  std::string inputFile = "input.dat";
};

int main(int argc, char *argv[]) {
  app_options opts =
        parse_options<app_options>(argc, argv);
```

https://godbolt.org/z/4s5685qYv

C++ Reflection: Back on Track

David Olsen, Compiler Engineer, NVIDIA

CppNorth, 22 July 2024

What is Reflection ?

Reflection

Introspection

Program observes the structure of itself

Reflection

Introspection

Program observes the structure of itself

Program asks questions about itself

Reflection

Introspection

Program observes the structure of its

Program asks questions about itself

Reflection

Introspection

Program observes the structure of itself

Program asks questions about itself

Reflection

Introspection

Reflection queries work on “reflection values”

Reflection

Introspection

Reflection queries work on “reflections”

Reflection

Introspection

Reflection queries work on “reflections”

Operation to transition code → reflection

Reflection

Code injection

Reflective metaprogramming

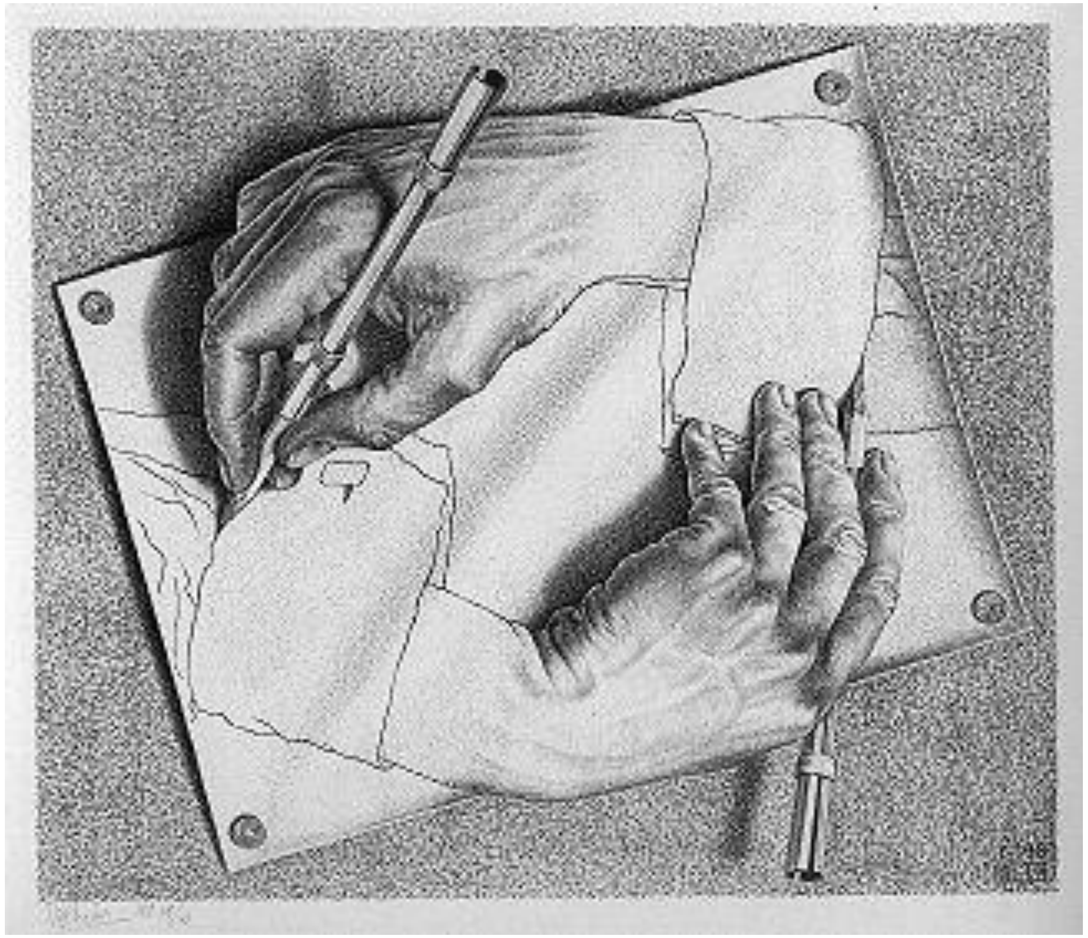
Change the state based on reflected information

Reflection

Code injection

Reflective metaprogramming

Change the state based
on reflected information



Reflection

Code injection

Reflection → code

Compile-time vs. Runtime

Need unknown type to ask questions about

Compile-time vs. Runtime

Need unknown type to ask questions about

- Program doesn’t know what type it is …

Compile-time vs. Runtime

Need unknown type to ask questions about

- Program doesn’t know what type it is …
- … but compiler does

Compile-time vs. Runtime

Need unknown type to ask questions about

- Program doesn't know what type it is …
- … but compiler does

Compile-time unknown

- Template parameter
- `template <class T> T func(T x) { /*` compiler knows T `*/ }`

Compile-time vs. Runtime

Need unknown type to ask questions about

- Program doesn't know what type it is …
- … but compiler does

Compile-time unknown

- Template parameter
- `template <class T> T func(T x) { /* compiler knows T */ }`

Runtime unknown

- Pointer to Base class
- Reference to Object in other languages

Compile-time vs. Runtime

Compile-time

- Inspect static types, e.g. template argument
- Generate different code
- Good match for statically-typed language with generics (a.k.a. templates)

Runtime

- Inspect dynamic types
- Run different code based on dynamic types
- Good match for OO language with universal base class

Compile-time vs. Runtime

Compile-time

- Inspect static types, e.g. template argument
- Generate different code
- Good match for statically-typed language with generics (a.k.a. templates)

Runtime

- Inspect dynamic types
- Run different code based on dynamic types
- Good match for OO language with universal base class

Compile-time vs. Runtime

Compile-time

- Inspect static types, e.g. template argument
- Generate different code
- Good match for statically-typed language with generics (a.k.a. templates)

Runtime

- Inspect dynamic types
- Run different code based on dynamic types
- Good match for OO language with universal base class

Reflection Survey

Reflection Survey: C#

C# Reflection

Introspection

Runtime only

Different types for different kinds

- `Type`, `MethodInfo`, `ConstructorInfo`, `FieldInfo`, `PropertyInfo`

Start with reflection of a class

- `obj.GetType()`: dynamic type of any object
- `typeof(SomeType)`: static reflection of SomeType
- `Module.GetType("SomeType")`: lookup type by name

From there, get information about the class

C# Reflection

Code Injection

Create new object

```
static object? Activator.CreateInstance(
                    Type type, params object? []? args);
```

Invoke a method on an existing object

```
object? MethodInfo.Invoke(
                    object? obj, object?[]? args);
```

Set the value of a field

```
void FieldInfo.SetValue(object? obj, object? value);
```

Set the value of a property

```
void PropertyInfo.SetValue(object? obj, object? value);
```

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Example

```
public static int SizeOrLength(object x) {
    foreach (MethodInfo m in x.GetType().GetMethods()) {
        if (m.Name == "Size" || m.Name == "Length") {
            return Convert.ToInt32(m.Invoke(x, null));
        }
    }
    return -1;
}
```

https://godbolt.org/z/4Y1ssEa3G

C# Reflection

Code Injection

Define new classes at runtime

Define methods by emitting .NET byte code

C# Reflection

Code injection example

```
public class Example {
  private int test;
  public Example(int test) { this.test = test; }
  private delegate TReturn OneParameter <TReturn, TParameter0>
                                        (TParameter0 p0);
  public static void Main() {

    Type[] methodArgs = {typeof(int)};
    DynamicMethod squareIt = new DynamicMethod(
        "SquareIt", typeof(long),
        methodArgs, typeof(Example).Module);
    ILGenerator il = squareIt.GetILGenerator();
    il.Emit(OpCodes.Ldarg_0);
    il.Emit(OpCodes.Conv_I8);
    il.Emit(OpCodes.Dup);
    il.Emit(OpCodes.Mul);
    il.Emit(OpCodes.Ret);
    OneParameter<long, int> invokeSquareIt =
        (OneParameter<long, int>)
        squareIt.CreateDelegate(typeof(OneParameter<long, int>));
    Console.WriteLine("123456789 squared = {0}",
                      invokeSquareIt(123456789));
    Type[] methodArgs2 = { typeof(Example), typeof(int) };
    DynamicMethod multiplyHidden = new DynamicMethod(
        "", typeof(int), methodArgs2, typeof(Example));
    ILGenerator ilMH = multiplyHidden.GetILGenerator();
    ilMH.Emit(OpCodes.Ldarg_0);
    FieldInfo testInfo = typeof(Example).GetField("test",
        BindingFlags.NonPublic | BindingFlags.Instance);
    ilMH.Emit(OpCodes.Ldfld, testInfo);
    ilMH.Emit(OpCodes.Ldarg_1);
    ilMH.Emit(OpCodes.Mul);
    ilMH.Emit(OpCodes.Ret);
    OneParameter<int, int> invoke = (OneParameter<int, int>)
        multiplyHidden.CreateDelegate(
            typeof(OneParameter<int, int>),
            new Example(42)
        );
    Console.WriteLine("3 * test = {0}", invoke(3));
  }
}
```

https://godbolt.org/z/qxs8Me445

https://learn.microsoft.com/en-us/dotnet/fundamentals/reflection/how-to-define-and-execute-dynamic-methods

C# Reflection

Code injection example

```
public class Example {
  private int test;
  public Example(int test) { this.test = test; }
  private delegate TReturn OneParameter <TReturn, TParameter0>
                                        (TParameter0 p0);
  public static void Main() {

    Type[] methodArgs = {typeof(int)};
    DynamicMethod squareIt = new DynamicMethod(
        "SquareIt", typeof(long),
        methodArgs, typeof(Example).Module);
    ILGenerator il = squareIt.GetILGenerator();
    il.Emit(OpCodes.Ldarg_0);
    il.Emit(OpCodes.Conv_I8);
    il.Emit(OpCodes.Dup);
    il.Emit(OpCodes.Mul);
    il.Emit(OpCodes.Ret);
    OneParameter<long, int> invokeSquareIt =
        (OneParameter<long, int>)
        squareIt.CreateDelegate(typeof(OneParameter<long, int>));
    Console.WriteLine("123456789 squared = {0}",
                      invokeSquareIt(123456789));
    Type[] methodArgs2 = { typeof(Example), typeof(int) };
    DynamicMethod multiplyHidden = new DynamicMethod(
        "", typeof(int), methodArgs2, typeof(Example));
    ILGenerator ilMH = multiplyHidden.GetILGenerator();
    ilMH.Emit(OpCodes.Ldarg_0);
    FieldInfo testInfo = typeof(Example).GetField("test",
        BindingFlags.NonPublic | BindingFlags.Instance);
    ilMH.Emit(OpCodes.Ldfld, testInfo);
    ilMH.Emit(OpCodes.Ldarg_1);
    ilMH.Emit(OpCodes.Mul);
    ilMH.Emit(OpCodes.Ret);
    OneParameter<int, int> invoke = (OneParameter<int, int>)
        multiplyHidden.CreateDelegate(
            typeof(OneParameter<int, int>),
            new Example(42)
        );
    Console.WriteLine("3 * test = {0}", invoke(3));
  }
}
```

https://godbolt.org/z/qxs8Me445

https://learn.microsoft.com/en-us/dotnet/fundamentals/reflection/how-to-define-and-execute-dynamic-methods

C# Reflection

Code injection example

```
public class Example {
  private int test;
  public Example(int test) { this.test = test; }
  static long SquareIt(int x) { return x * x; }
  int unnamed(int x) { return this.test * x; }
  public static void Main() {
    Console.WriteLine("123456789 squared = {0}",
                      SquareIt(123456789));
    Console.WriteLine("3 * test = {0}",
                      new Example(42).unnamed(3));
  }
}
```

https://godbolt.org/z/ssx4P98Y9

C# Reflection

Code injection example

```
public class Example {
  private int test;
  public Example(int test) { this.test = test; }
  static long SquareIt(int x) { return x * x; }
  int unnamed(int x) { return this.test * x; }
  public static void Main() {
    Console.WriteLine("123456789 squared = {0}",
                      SquareIt(123456789));
    Console.WriteLine("3 * test = {0}",
                      new Example(42).unnamed(3));
  }
}
```

https://godbolt.org/z/ssx4P98Y9

Reflection Survey:
Current C++ (sort of)

`std::type_info`

Extremely limited runtime reflection

`typeid(type)` gets a `type_info` for that type

`typeid(expr)` gets a `type_info` for the expression's type

- Except if `expr` is an lvalue of polymorphic class type, then gets `type_info` for the dynamic type of the expression

`.name()` is the only useful information

Use to implement `dynamic_cast` and exceptions

Type traits

Compile-time introspection of types

Type traits

Compile-time introspection of types

```
is_void is_null_pointer is_integral is_floating_point is_array is_enum is_union
is_class is_function is_pointer is_lvalue_reference is_rvalue_reference
is_member_object_pointer is_member_function_pointer is_fundamental is_arithmetic
is_scalar is_object is_compound is_reference is_member_pointer is_const
is_volatile is_trivial is_trivially_copyable is_standard_layout
has_unique_object_representation is_empty is_polymorphic is_abstract is_final
is_aggregate is_implicit_lifetime is_signed is_unsigned is_bounded_array
is_unbounded_array is_scoped_enum is_[trivially_,nothrow_]constructible
is_[trivially_,nothrow_]default_constructible
is_[trivially_,nothrow_]copy_constructible
is_[trivially_,nothrow_]move_constructible is_[trivially_,nothrow_]assignable
is_[trivially_,nothrow_]copy_assignable is_[trivially_,nothrow_]move_assignable
is_[trivially_,nothrow_]destructible has_virtual_destructor
is_[nothrow_]swappable[_with] alignment_of rank extent is_same is_base_of
is_[nothrow_]convertible is_layout_compatible is_pointer_interconvertible_base_of
is_[nothrow_]invocable[_r]
```

Type traits

Compile-time introspection of types

But only of types and type properties

- Can’t ask about class members or parent scope
- Can’t ask about namespaces or functions or templates

Type traits

Not intended as a reflection solution

Side effect of magic of function template argument deduction
applied to class template partial specialization

`requires` expression

Compile-time introspection of code validity

`requires` expression

Example

```
template <typename T>
auto size_or_length(T&& x) {
  if constexpr (requires(T y){ { y.size() } -> std::integral; }) {
    return x.size();
  } else if constexpr (requires(T y){
                         { y.length() } -> std::integral; }) {
    return x.length();
  } else {
    return -1;
  }
}
```

https://godbolt.org/z/vK1reo461

`requires` expression

Example

```
template <typename T>
auto size_or_length(T&& x) {
  if constexpr (requires(T y){ { y.size() } -> std::integral; }) {
    return x.size();
  } else if constexpr (requires(T y){
                         { y.length() } -> std::integral; }) {
    return x.length();
  } else {
    return -1;
  }
}
```

https://godbolt.org/z/vK1reo461

`requires` expression

Example

```
template <typename T>
auto size_or_length(T&& x) {
  if constexpr (requires(T y){ { y.size() } -> std::integral; }) {
    return x.size();
  } else if constexpr (requires(T y){
                         { y.length() } -> std::integral; }) {
    return x.length();
  } else {
    return -1;
  }
}
```

https://godbolt.org/z/vK1reo461

`requires` expression

Example

```
template <typename T>
auto size_or_length(T&& x) {
  if constexpr (requires(T y){ { y.size() } -> std::integral; }) {
    return x.size();
  } else if constexpr (requires(T y){
                         { y.length() } -> std::integral; }) {
    return x.length();
  } else {
    return -1;
  }
}
```

https://godbolt.org/z/vK1reo461

`requires` expression

Example

```
template <typename T>
auto size_or_length(T&& x) {
  if constexpr (requires(T y){ { y.size() } -> std::integral; }) {
    return x.size();
  } else if constexpr (requires(T y){
                         { y.length() } -> std::integral; }) {
    return x.length();
  } else {
    return -1;
  }
}
```

https://godbolt.org/z/vK1reo461

`requires` expression

Example

```
template <typename T>
auto size_or_length(T&& x) {
  if constexpr (requires(T y){ { y.size() } -> std::integral; }) {
    return x.size();
  } else if constexpr (requires(T y){
                         { y.length() } -> std::integral; }) {
    return x.length();
  } else {
    return -1;
  }
}
```

https://godbolt.org/z/vK1reo461

Reflection Survey: Past Attempts

Reflection TS

Introspection

Compile-time template metaprogramming

A reflection is a unique type

Type metafunctions to query/manipulate reflections

```
using r_str = reflexpr(std::string);
using r_basic = get_aliased_t<r_str>;
```

https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4766.pdf

Reflection TS

Code injection

Reflection to type

- `get_reflected_type_t<r>`

Reflection to function, variable, or data member

- `get_pointer_v<r>`
- Depending on what `r` is a reflection of
  - Pointer to function
  - Pointer to object
  - Pointer to member function
  - Pointer to data member

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

```
using namespace std::experimental::reflect;
template <MemberFunction... MemFuncs> struct find_size_or_length {
  template <typename T> constexpr static int value(T x) { return -1; }
};
template <MemberFunction First, MemberFunction... Rest>
struct find_size_or_length<First, Rest...> {
  template <typename T> constexpr static int value(T x) {
    if constexpr (str_eq(get_name<First>::value, "size") ||
                  str_eq(get_name<First>::value, "length")) {
      return (x.*get_pointer_v<First>)();
    }
    return find_size_or_length<Rest...>::value(x);
  }
};
template <typename T> int size_or_length(T x) {
  if constexpr (std::is_class_v<T>) {
    using memfuncs = get_public_member_functions_t<get_aliased_t<reflexpr(T)>>;
    return unpack_sequence_t<find_size_or_length, memfuncs>::value(x);
  }
  return -1;
}
```

https://godbolt.org/z/v7bee41M1

“Reflection for C++26”

Current Proposal

P2996 “Reflection for C++26,” *Daveed Vandervourde, Wyatt Childers, Peter Dimov, Dan Katz, Barry Rezvin, Andrew Sutton, Faisil Vali*

Minimal viable proposal

- More introspection
- Less code injection

Compile-time, function-based API

- All functions are `consteval`

https://wg21.link/p2996

Current Proposal

Status

Approved by Reflection and Evolution groups

Being reviewed by Library Evolution and Core

Will continue to gain consensus over the next couple meetings

`std::meta::info`

Reflection Type

Opaque scalar type

- One type for everything
- Not separate types as in Java or C#

Holds reflection information for something

Only useful at compile-time

- Slicers require a compile-time operand
- Reflection metafunctions are all consteval
- Useful as a non-type template argument

`std::meta::info`

Reflection Type

Why one type for everything?

`std::meta::info`

Reflection Type

Why one type for everything?

Language will change over time

- Easier to react to changes

`std::meta::info`

Reflection Type

Why one type for everything?

Language will change over time

- Easier to react to changes

Collections of mixed things

- `std::vector<std::meta::info>` can represent
  - all class members
  - list of template arguments

^

Reflection Operator

Converts grammatical construct into reflection value

Prefix unary `^` operator

Result is `std::meta::info` object

Operand may be:

- type
- namespace, including `::`
- name: function, variable, structured binding, template, concept

^

Examples

```
constexpr std::meta::info x =
    ^int;
```

Reflection of built-in type `int`

^

Examples

```
constexpr std::meta::info x =
    ^std::vector<int>;
```

Reflection of type `std::vector<int, std::allocator<int>>`

^

Examples

```
constexpr std::meta::info x =
    ^std::vector;
```

Reflection of class template `std::vector`

^

Examples

```
constexpr std::meta::info x =
    ^std::string;
```

Reflection of type alias `std::string`

which is an alias for

```
std::basic_string<char, std::char_traits<char>, std::allocator<char>>
```

^

Examples

```
constexpr std::meta::info x =
    ^::;
```

Reflection of global namespace

^

Examples

```
constexpr std::meta::info x =
    ^std::chrono;
```

Reflection of namespace `std::chrono`

^

Examples

```
constexpr std::meta::info x =
    ^std::fopen;
```

Reflection of function `std::fopen`

`[: :]`

Splicer

Converts reflection value into code

Operand must be a `std::meta::info`

Sometimes preceded by `typename` or `template`

Can be used wherever the thing being reflected is valid

Basic example

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

Basic example

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

Basic example

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

Basic example

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

[: :]

Splicer

‘typename’ or ‘template’ may be needed before ‘[: :]’

Similar to using ‘typename’ or ‘template’ before a name that depends on a template parameter

```
constexpr auto vt = ^std::vector;
template[: vt :]<int> vi = {};
                  // Same as std::vector<int> vi = {};
```

[: :]

Splicer

Can be used anywhere that the reflected thing is valid

[: :]

Splicer

Can be used anywhere that the reflected thing is valid

```
constexpr auto type = ^long;
std::vector<typename[:type:]> v;
typename[:type:] var = static_cast<[:type:]>(42);
```

https://godbolt.org/z/vEKbfh7zz

[: :]

Splicer

Can be used anywhere that the reflected thing is valid

```
namespace N {
    class A { };
}
constexpr auto ns = ^N;
[:ns:]::A aaa;
```

https://godbolt.org/z/d351j9j4x

[: :]

Splicer

Can be used anywhere that the reflected thing is valid

```
int f(int a, int b) { return a + b; }
constexpr auto func = ^f;
int main() {
    int (*fp)(int, int) = &[:func:];
    return [:func:](1, 2) + fp(3, 4);
}
```

https://godbolt.org/z/eGMPf5rK5

[: :]

Splicer

Can be used anywhere that the reflected thing is valid

```
struct A { int z; };
int main() {
    constexpr auto member = ^A::z;
    A a;
    a.[:member:] = 42;
    return a.[:member:];
}
```

https://godbolt.org/z/1cha1de49

[: :]

Splicer

Can be used anywhere that the reflected thing is valid

```
struct A { int z; };
struct B { int z; };
int main() {
    constexpr auto member = ^A::z;
    B b;
    b.[:member:] = 42;
    return b.[:member:];
}
```

https://godbolt.org/z/h3b157dxG

[: :]

Splicer

Can be used anywhere that the reflected thing is valid

```
struct A { int z; };
struct B { int z; };
int main() {
    constexpr auto member = ^A::z;
    B b;
    b.[:member:] = 42;
    return b.[:member:];
}
```

https://godbolt.org/z/h3b157dxG

[: :]

Splicer

Can be used anywhere that the reflected thing is valid

```
struct A { int z; };
struct B { int z; };
int main() {
    constexpr auto member = ^A::z;
    B b;
    b.[:member:] = 42;
    return b.[:member:];
}
```

```
error: qualified name is not a member
  of class "B" or its base classes
```

https://godbolt.org/z/h3b157dxG

Metafunctions

Inputs are `std::meta::info`

Outputs are information about the reflection value

- Possibly other reflection values

In namespace `std::meta`

`consteval` functions

Metafunctions

Kind

Query what kind of thing the reflection represents

```
consteval bool is_namespace(info r);
consteval bool is_function(info r);
consteval bool is_variable(info r);
consteval bool is_type(info r);
consteval bool is_alias(info r);
consteval bool is_template(info r);
consteval bool is_concept(info r);
consteval bool is_class_member(info r);
consteval bool is_base(info r);
```

Metafunctions

Names

```
consteval string_view name_of(info r);
consteval string_view display_name_of(info r);
consteval source_location source_location_of(info r);
```

Metafunctions

Type properties

Many queries that can also be done via `<type_traits>`

Type category:

`type_is_integral`, `type_is_array`, `type_is_class`, `type_is_enum`, `type_is_function`

Type properties:

`type_is_const`, `type_is_trivial`, `type_is_aggregate`, `type_is_signed`, `type_is_move_constructible`, `type_is_nothrow_default_constructible`

Type relations:

`type_is_same`, `type_is_base_of`, `type_is_convertible`

Metafunctions

Properties

Access:

`is_public`, `is_protected`, `is_private`

Linkage:

`has_linkage`, `has_internal_linkage`, `has_external_linkage`

Virtual modifiers:

`is_virtual`, `is_pure_virtual`, `is_override`

Other:

`is_defaulted`, `is_deleted`, `is_explicit`

Metafunctions

Members

Return type is `std::vector<std::meta::info>`

`members_of` – class or namespace

`bases_of`

`static_data_members_of`

`nonstatic_data_members_of`

`subobjects_of` – bases and non-static data members

`enumerators_of`

Metafunctions

Members

Return type is `std::vector<std::meta::info>`

Second parameter is the “from” context for accessibility

`accessible_members_of`

`accessible_bases_of`

`accessible_static_data_members_of`

`accessible_nonstatic_data_members_of`

`accessible_subobjects_of`

Metafunctions

Templates

Is a reflection a template specialization?

```
consteval bool has_template_arguments(info r);
```

If so, get the base template and the template arguments

```
consteval info template_of(info r);
consteval vector<info> template_arguments_of(info r);
```

Metafunctions

Reflecting values

Can't use ^ on an expression

Use `reflect_value` metafunction instead

```
template<typename T>
consteval auto reflect_value(T expr) -> info;
```

Takes the reflection of the given compile-time value

```
template<typename T>
consteval auto extract(info) -> T;
```

Converts a reflection of a value into a compile-time value

Metafunctions

Error reporting

What if the query is nonsensical?

Metafunctions

Error reporting

What if the query is nonsensical?

For boolean queries, returns false

```
static_assert(not is_pure_virtual(^int));
```

https://godbolt.org/z/o88KGqGnW

Metafunctions

Error reporting

What if the query is nonsensical?

For other queries, compilation fails

```
constexpr auto off = offset_of(^int);
```

```
error: call to consteval function
  'std::meta::offset_of' is not a constant expression
note: expected a reflection of a non-static data
  member, but got a type
```

https://godbolt.org/z/f56nY3Edz

Metafunctions

Error reporting

What if the query is nonsensical?

For other queries, compilation fails

```
constexpr auto off = offset_of(^int);
```

```
error: call to consteval function
  'std::meta::offset_of' is not a constant expression
note: expected a reflection of a non-static data
  member, but got a type
```

https://godbolt.org/z/f56nY3Edz

`template for`

Expansion statements

https://wg21.link/p1306

template for

Expansion statements

```
template for (constexpr auto e : enumerators_of(^E))
```

https://wg21.link/p1306

template for

Expansion statements

```
template for (constexpr auto e : enumerators_of(^E))

[:expand(enumerators_of(^E)):] >> [&]<auto e>{
```

https://wg21.link/p1306

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                      -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                      -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                      -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                      -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                      -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                      -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                    -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                      -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Size or Length

```
template <typename T> int size_or_length(T&& x) {
  if constexpr (type_is_class(^T)) {
    template for (constexpr auto member : members_of(^T)) {
      if constexpr (is_function(member) &&
                    (name_of(member) == "size" ||
                     name_of(member) == "length")  &&
                    requires(T y) { { y.[:member:]() }
                                      -> std::integral; }) {
        return x.[:member:]();
      }
    }
  }
  return -1;
}
```

https://godbolt.org/z/G699caEPW

Parse Options

```
template<typename Opts> Opts parse_options(int argc, char** argv) {
  std::vector<std::string_view> args(argv + 1, argv + argc);
  Opts opts;
  template for (constexpr auto dm : nonstatic_data_members_of(^Opts)) {
    auto it = std::find_if(args.begin(), args.end(), [](std::string_view arg){
        return arg.starts_with("--") && arg.substr(2) == name_of(dm);
      });
    if (it == args.end()) return;
    using T = typename[:type_of(dm):];
    if constexpr (std::is_same_v<T, bool>) {
      opts.[:dm:] = true;
    } else if (it + 1 == args.end()) {
      std::cerr << "Missing value for option " << *it << "\n";
    } else {
      auto iss = std::ispanstream(it[1]);
      if (iss >> opts.[:dm:]; !iss)
        std::cerr << "Invalid value \"" << it[1] << "\" for option " << *it << "\n";
    }
  };
  return opts;
}
```

https://godbolt.org/z/4s5685qYv

Parse Options

```
template<typename Opts> Opts parse_options(int argc, char** argv) {
  std::vector<std::string_view> args(argv + 1, argv + argc);
  Opts opts;
  template for (constexpr auto dm : nonstatic_data_members_of(^Opts)) {
    auto it = std::find_if(args.begin(), args.end(), [](std::string_view arg){
        return arg.starts_with("--") && arg.substr(2) == name_of(dm);
      });
    if (it == args.end()) return;
    using T = typename[:type_of(dm):];
    if constexpr (std::is_same_v<T, bool>) {
      opts.[:dm:] = true;
    } else if (it + 1 == args.end()) {
      std::cerr << "Missing value for option " << *it << "\n";
    } else {
      auto iss = std::ispanstream(it[1]);
      if (iss >> opts.[:dm:]; !iss)
        std::cerr << "Invalid value \"" << it[1] << "\" for option " << *it << "\n";
    }
  };
  return opts;
}
```

https://godbolt.org/z/4s5685qYv

Parse Options

```
template<typename Opts> Opts parse_options(int argc, char** argv) {
  std::vector<std::string_view> args(argv + 1, argv + argc);
  Opts opts;
  template for (constexpr auto dm : nonstatic_data_members_of(^Opts)) {
    auto it = std::find_if(args.begin(), args.end(), [](std::string_view arg){
        return arg.starts_with("--") && arg.substr(2) == name_of(dm);
      });
    if (it == args.end()) return;
    using T = typename[:type_of(dm):];
    if constexpr (std::is_same_v<T, bool>) {
      opts.[:dm:] = true;
    } else if (it + 1 == args.end()) {
      std::cerr << "Missing value for option " << *it << "\n";
    } else {
      auto iss = std::ispanstream(it[1]);
      if (iss >> opts.[:dm:]; !iss)
        std::cerr << "Invalid value \"" << it[1] << "\" for option " << *it << "\n";
    }
  };
  return opts;
}
```

https://godbolt.org/z/4s5685qYv

Parse Options

```
template<typename Opts> Opts parse_options(int argc, char** argv) {
  std::vector<std::string_view> args(argv + 1, argv + argc);
  Opts opts;
  template for (constexpr auto dm : nonstatic_data_members_of(^Opts)) {
    auto it = std::find_if(args.begin(), args.end(), [](std::string_view arg){
        return arg.starts_with("--") && arg.substr(2) == name_of(dm);
      });
    if (it == args.end()) return;
    using T = typename[:type_of(dm):];
    if constexpr (std::is_same_v<T, bool>) {
      opts.[:dm:] = true;
    } else if (it + 1 == args.end()) {
      std::cerr << "Missing value for option " << *it << "\n";
    } else {
      auto iss = std::ispanstream(it[1]);
      if (iss >> opts.[:dm:]; !iss)
        std::cerr << "Invalid value \"" << it[1] << "\" for option " << *it << "\n";
    }
  };
  return opts;
}
```

https://godbolt.org/z/4s5685qYv

Parse Options

```
template<typename Opts> Opts parse_options(int argc, char** argv) {
  std::vector<std::string_view> args(argv + 1, argv + argc);
  Opts opts;
  template for (constexpr auto dm : nonstatic_data_members_of(^Opts)) {
    auto it = std::find_if(args.begin(), args.end(), [](std::string_view arg){
        return arg.starts_with("--") && arg.substr(2) == name_of(dm);
      });
    if (it == args.end()) return;
    using T = typename[:type_of(dm):];
    if constexpr (std::is_same_v<T, bool>) {
      opts.[:dm:] = true;
    } else if (it + 1 == args.end()) {
      std::cerr << "Missing value for option " << *it << "\n";
    } else {
      auto iss = std::ispanstream(it[1]);
      if (iss >> opts.[:dm:]; !iss)
        std::cerr << "Invalid value \"" << it[1] << "\" for option " << *it << "\n";
    }
  };
  return opts;
}
```

https://godbolt.org/z/4s5685qYv

Parse Options

```
template<typename Opts> Opts parse_options(int argc, char** argv) {
  std::vector<std::string_view> args(argv + 1, argv + argc);
  Opts opts;
  template for (constexpr auto dm : nonstatic_data_members_of(^Opts)) {
    auto it = std::find_if(args.begin(), args.end(), [](std::string_view arg){
        return arg.starts_with("--") && arg.substr(2) == name_of(dm);
      });
    if (it == args.end()) return;
    using T = typename[:type_of(dm):];
    if constexpr (std::is_same_v<T, bool>) {
      opts.[:dm:] = true;
    } else if (it + 1 == args.end()) {
      std::cerr << "Missing value for option " << *it << "\n";
    } else {
      auto iss = std::ispanstream(it[1]);
      if (iss >> opts.[:dm:]; !iss)
        std::cerr << "Invalid value \"" << it[1] << "\" for option " << *it << "\n";
    }
  };
  return opts;
}
```

https://godbolt.org/z/4s5685qYv

Parse Options

```
template<typename Opts> Opts parse_options(int argc, char** argv) {
  std::vector<std::string_view> args(argv + 1, argv + argc);
  Opts opts;
  template for (constexpr auto dm : nonstatic_data_members_of(^Opts)) {
    auto it = std::find_if(args.begin(), args.end(), [](std::string_view arg){
        return arg.starts_with("--") && arg.substr(2) == name_of(dm);
      });
    if (it == args.end()) return;
    using T = typename[:type_of(dm):];
    if constexpr (std::is_same_v<T, bool>) {
      opts.[:dm:] = true;
    } else if (it + 1 == args.end()) {
      std::cerr << "Missing value for option " << *it << "\n";
    } else {
      auto iss = std::ispanstream(it[1]);
      if (iss >> opts.[:dm:]; !iss)
        std::cerr << "Invalid value \"" << it[1] << "\" for option " << *it << "\n";
    }
  };
  return opts;
}
```

https://godbolt.org/z/4s5685qYv

`substitute`

Perform a template substitution

```
consteval info substitute(info templ, span<info const> args);
```

Substitution only, not instantiation

- Instantiation will happen when the type is used in code

Sort of equivalent to

```
template[:templ:]<[:args:]...>
```

`substitute` Example

Propagate allocator type from container to its `value_type`

Given:

```
my_container<T, custom_alloc<T>>
```

If `T` has a template argument `std::allocator<U>`

change it to `custom_alloc<U>`

substitute Example

```
template <typename Value, typename Alloc> consteval std::meta::info change_allocator() {
  constexpr auto type = dealias(^Value);
  constexpr auto alloc = dealias(^Alloc);
  if constexpr (template_of(alloc) == ^std::allocator || not has_template_arguments(type)) {
    return type;
  } else {
    std::vector<std::meta::info> new_args;
    template for (constexpr auto arg : template_arguments_of(type)) {
      if constexpr (has_template_arguments(arg) && template_of(arg) == ^std::allocator) {
        using rebound = typename std::allocator_traits<typename[:alloc:]>
                              ::template rebind_alloc<typename[:template_arguments_of(arg)[0]:]>;
        new_args.push_back(^rebound);
      } else {
        new_args.push_back(arg);
      }
    }
    return substitute(template_of(type), new_args);
  }
}
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
template <typename Value, typename Alloc> consteval std::meta::info change_allocator() {
  constexpr auto type = dealias(^Value);
  constexpr auto alloc = dealias(^Alloc);
  if constexpr (template_of(alloc) == ^std::allocator || not has_template_arguments(type)) {
    return type;
  } else {
    std::vector<std::meta::info> new_args;
    template for (constexpr auto arg : template_arguments_of(type)) {
      if constexpr (has_template_arguments(arg) && template_of(arg) == ^std::allocator) {
        using rebound = typename std::allocator_traits<typename[:alloc:]>
                              ::template rebind_alloc<typename[:template_arguments_of(arg)[0]:]>;
        new_args.push_back(^rebound);
      } else {
        new_args.push_back(arg);
      }
    }
    return substitute(template_of(type), new_args);
  }
}
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
template <typename Value, typename Alloc> consteval std::meta::info change_allocator() {
  constexpr auto type = dealias(^Value);
  constexpr auto alloc = dealias(^Alloc);
  if constexpr (template_of(alloc) == ^std::allocator || not has_template_arguments(type)) {
    return type;
  } else {
    std::vector<std::meta::info> new_args;
    template for (constexpr auto arg : template_arguments_of(type)) {
      if constexpr (has_template_arguments(arg) && template_of(arg) == ^std::allocator) {
        using rebound = typename std::allocator_traits<typename[:alloc:]>
                              ::template rebind_alloc<typename[:template_arguments_of(arg)[0]:]>;
        new_args.push_back(^rebound);
      } else {
        new_args.push_back(arg);
      }
    }
    return substitute(template_of(type), new_args);
  }
}
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
template <typename Value, typename Alloc> consteval std::meta::info change_allocator() {
  constexpr auto type = dealias(^Value);
  constexpr auto alloc = dealias(^Alloc);
  if constexpr (template_of(alloc) == ^std::allocator || not has_template_arguments(type)) {
    return type;
  } else {
    std::vector<std::meta::info> new_args;
    template for (constexpr auto arg : template_arguments_of(type)) {
      if constexpr (has_template_arguments(arg) && template_of(arg) == ^std::allocator) {
        using rebound = typename std::allocator_traits<typename[:alloc:]>
                              ::template rebind_alloc<typename[:template_arguments_of(arg)[0]:]>;
        new_args.push_back(^rebound);
      } else {
        new_args.push_back(arg);
      }
    }
    return substitute(template_of(type), new_args);
  }
}
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
template <typename Value, typename Alloc> consteval std::meta::info change_allocator() {
  constexpr auto type = dealias(^Value);
  constexpr auto alloc = dealias(^Alloc);
  if constexpr (template_of(alloc) == ^std::allocator || not has_template_arguments(type)) {
    return type;
  } else {
    std::vector<std::meta::info> new_args;
    template for (constexpr auto arg : template_arguments_of(type)) {
      if constexpr (has_template_arguments(arg) && template_of(arg) == ^std::allocator) {
        using rebound = typename std::allocator_traits<typename[:alloc:]>
                                ::template rebind_alloc<typename[:template_arguments_of(arg)[0]:]>;
        new_args.push_back(^rebound);
      } else {
        new_args.push_back(arg);
      }
    }
    return substitute(template_of(type), new_args);
  }
}
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
template <typename Value, typename Alloc> consteval std::meta::info change_allocator() {
  constexpr auto type = dealias(^Value);
  constexpr auto alloc = dealias(^Alloc);
  if constexpr (template_of(alloc) == ^std::allocator || not has_template_arguments(type)) {
    return type;
  } else {
    std::vector<std::meta::info> new_args;
    template for (constexpr auto arg : template_arguments_of(type)) {
      if constexpr (has_template_arguments(arg) && template_of(arg) == ^std::allocator) {
        using rebound = typename std::allocator_traits<typename[:alloc:]>
                              ::template rebind_alloc<typename[:template_arguments_of(arg)[0]:]>;
        new_args.push_back(^rebound);
      } else {
        new_args.push_back(arg);
      }
    }
    return substitute(template_of(type), new_args);
  }
}
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
template <typename T, typename Alloc = std::allocator<T>>
struct my_container {
  // ...
  using value_type =
          typename[:change_allocator<T, Alloc>():];
  // ...
};
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
static_assert(std::same_as<
    my_container<int, custom_alloc<>>::value_type,
    int>);
template <typename T> struct B { };
static_assert(std::same_as<
    my_container<B<int>, custom_alloc<>>::value_type,
    B<int>>);
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
static_assert(std::same_as<
    my_container<std::vector<int>,
                 custom_alloc<>>::value_type,
    std::vector<int, custom_alloc<int>>>);
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
static_assert(std::same_as<
    my_container<std::vector<int>,
                 custom_alloc<>>::value_type,
    std::vector<int, custom_alloc<int>>>);
```



```
std::vector<int, std::allocator<int>>
```

https://godbolt.org/z/aTqWrb1az

substitute Example

```
static_assert(std::same_as<
    my_container<std::vector<int>,
                 custom_alloc<>>::value_type,
    std::vector<int, custom_alloc<int>>>);
```

https://godbolt.org/z/aTqWrb1az

define_class

```
consteval info define_class(info class_type,
                            span<info const> nsdm);
```

Define a class using reflection values rather than source code

Takes reflection of incomplete class and range of reflections of non-static data members

Returns reflection of complete class with the given non-static data members

Only non-static data members for now

Other kinds of members will be added later

define_class

data_member_spec

```
struct data_member_options_t {
  optional<string_view> name;
  bool is_static = false;
  optional<int> alignment;
  optional<int> width;
};
consteval info data_member_spec(
    info type, data_member_options_t options = {});
```

define_class Example

```
template<typename T> struct S;
constexpr auto U = define_class(^S<int>, {
  data_member_spec(^int, {.name="i", .align=64}),
  data_member_spec(^int, {.name="j", .align=64}),
});

// S<int> is now defined to the equivalent of
// template<> struct S<int> {
//   alignas(64) int i;
//   alignas(64) int j;
// };
```

define_class Example

AOS to SOA

```
template <typename T, std::size_t N> struct struct_of_arrays_impl;
consteval auto make_struct_of_arrays(std::meta::info type,
                                     std::meta::info N) -> std::meta::info {
  std::vector<std::meta::info> old_members = nonstatic_data_members_of(type);
  std::vector<std::meta::nsdm_description> new_members = {};
  for (std::meta::info member : old_members) {
    auto array_type = substitute(^std::array, {type_of(member), N });
    auto mem_descr = make_nsdm_description(array_type, {.name = name_of(member)});
    new_members.push_back(mem_descr);
  }
  return std::meta::define_class(substitute(^struct_of_arrays_impl, {type, N}),
                                 new_members);
}
template <typename T, std::size_t N>
    using struct_of_arrays = [: make_struct_of_arrays(^T, ^N) :];
```

https://godbolt.org/z/Whdvs3j1n

define_class Example

AOS to SOA

```
template <typename T, std::size_t N> struct struct_of_arrays_impl;
consteval auto make_struct_of_arrays(std::meta::info type,
                                     std::meta::info N) -> std::meta::info {
  std::vector<std::meta::info> old_members = nonstatic_data_members_of(type);
  std::vector<std::meta::nsdm_description> new_members = {};
  for (std::meta::info member : old_members) {
    auto array_type = substitute(^std::array, {type_of(member), N });
    auto mem_descr = make_nsdm_description(array_type, {.name = name_of(member)});
    new_members.push_back(mem_descr);
  }
  return std::meta::define_class(substitute(^struct_of_arrays_impl, {type, N}),
                                 new_members);
}
template <typename T, std::size_t N>
    using struct_of_arrays = [: make_struct_of_arrays(^T, ^N) :];
```

https://godbolt.org/z/Whdvs3j1n

define_class Example

AOS to SOA

```
template <typename T, std::size_t N> struct struct_of_arrays_impl;
consteval auto make_struct_of_arrays(std::meta::info type,
                                     std::meta::info N) -> std::meta::info {
  std::vector<std::meta::info> old_members = nonstatic_data_members_of(type);
  std::vector<std::meta::nsdm_description> new_members = {};
  for (std::meta::info member : old_members) {
    auto array_type = substitute(^std::array, {type_of(member), N });
    auto mem_descr = make_nsdm_description(array_type, {.name = name_of(member)});
    new_members.push_back(mem_descr);
  }
  return std::meta::define_class(substitute(^struct_of_arrays_impl, {type, N}),
                                 new_members);
}
template <typename T, std::size_t N>
    using struct_of_arrays = [: make_struct_of_arrays(^T, ^N) :];
```

https://godbolt.org/z/Whdvs3j1n

define_class Example

AOS to SOA

```
template <typename T, std::size_t N> struct struct_of_arrays_impl;
consteval auto make_struct_of_arrays(std::meta::info type,
                                     std::meta::info N) -> std::meta::info {
  std::vector<std::meta::info> old_members = nonstatic_data_members_of(type);
  std::vector<std::meta::nsdm_description> new_members = {};
  for (std::meta::info member : old_members) {
    auto array_type = substitute(^std::array, {type_of(member), N });
    auto mem_descr = make_nsdm_description(array_type, {.name = name_of(member)});
    new_members.push_back(mem_descr);
  }
  return std::meta::define_class(substitute(^struct_of_arrays_impl, {type, N}),
                                 new_members);
}
template <typename T, std::size_t N>
    using struct_of_arrays = [: make_struct_of_arrays(^T, ^N) :];
```

https://godbolt.org/z/Whdvs3j1n

define_class Example

AOS to SOA

```
template <typename T, std::size_t N> struct struct_of_arrays_impl;
consteval auto make_struct_of_arrays(std::meta::info type,
                                     std::meta::info N) -> std::meta::info {
  std::vector<std::meta::info> old_members = nonstatic_data_members_of(type);
  std::vector<std::meta::nsdm_description> new_members = {};
  for (std::meta::info member : old_members) {
    auto array_type = substitute(^std::array, {type_of(member), N });
    auto mem_descr = make_nsdm_description(array_type, {.name = name_of(member)});
    new_members.push_back(mem_descr);
  }
  return std::meta::define_class(substitute(^struct_of_arrays_impl, {type, N}),
                                 new_members);
}
template <typename T, std::size_t N>
    using struct_of_arrays = [: make_struct_of_arrays(^T, ^N) :];
```

https://godbolt.org/z/Whdvs3j1n

define_class Example

AOS to SOA

```
struct point {
  float x;
  float y;
  float z;
};

using points = struct_of_arrays<point, 30>;
// equivalent to:
// struct points {
//   std::array<float, 30> x;
//   std::array<float, 30> y;
//   std::array<float, 30> z;
// };
```

https://godbolt.org/z/8rT77KxjP

Reflection Implementations

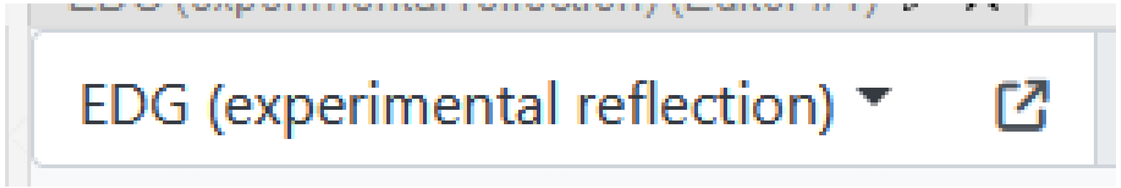
EDG

Implementation

First implementation

Hidden option in EDG front end

Available on Compiler explorer



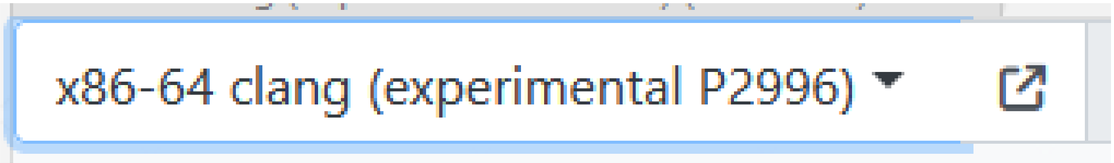
https://godbolt.org/z/13anqE1Pa

Clang

Implementation

Fork of Clang

Available on Compiler explorer



https://godbolt.org/z/oTGEhh1v4

NVIDIA

Implementation

NVIDIA HPC C++ compiler (NVC++) release 24.3 or newer

- Has EDG front end

Available on Compiler Explorer

- Options needed to turn on reflection

Available for download on Linux

https://developer.nvidia.com/hpc-sdk-downloads

https://godbolt.org/z/PqTqcP7x4

Reflection Example

Serialization

Without reflection

Serialization libraries handle most types automatically

Serialization

Without reflection

Serialization libraries handle most types automatically

But they can’t figure out classes

Require a user-supplied function to tell how to serialize a class

Serialization

Without reflection

Require a user-supplied function to tell how to serialize a class

```
class gps_position {
   friend class boost::serialization::access;
   template<class Archive> void serialize(Archive & ar, const unsigned int version) {
      ar & degrees;
      ar & minutes;
      ar & seconds;
   }
   int degrees;
   int minutes;
   float seconds;
```

https://www.boost.org/doc/libs/1_79_0/libs/serialization/doc/tutorial.html

Serialization

With reflection

Reflection is the missing piece

- Serialization code can inspect the data members
- No user-input required

```cpp
template <typename T> requires std::is_arithmetic_v<T>
void Serialization::serialize_arithmetic(
      std::ostream& output, T const& x) {
  output << x << " ";
}

template <typename T> requires std::is_pointer_v<T>
void Serialization::serialize_pointer(
      std::ostream& output, T const& x) {
  output << static_cast<const void*>(x) << " ";
}

template <typename T> requires std::is_enum_v<T>
void Serialization::serialize_enum(
      std::ostream& output, T const& x) {
  output << static_cast<std::underlying_type_t<T>>(x)
         << " ";
}

template <typename E, int N>
void Serialization::serialize_array(
      std::ostream& output, E const (& x)[N]) {
  output << "[ ";
  for (int i = 0; i < N; ++i) {
    serialize_object(output, x[i]);
  }
  output << "] ";
}
```

```cpp
template <typename T>
void Serialization::serialize_object(
      std::ostream& output, T const& x) {
  if constexpr (std::is_array_v<T>) {
    serialize_array(output, x);
  } else if constexpr (std::is_class_v<T>) {
    serialize_struct(output, x);
  } else if constexpr (std::is_enum_v<T>) {
    serialize_enum(output, x);
  } else if constexpr (std::is_pointer_v<T>) {
    serialize_pointer(output, x);
  } else if constexpr (std::is_arithmetic_v<T>) {
    serialize_arithmetic(output, x);
  } else {
    assert(false);
  }
}

template <typename T>
std::string Serialization::serialize(T const& x) {
  std::stringstream result;
  serialize_object(result, x);
  return result.str();
}
```

Serialization

Serialize struct

```
template <typename T> requires std::is_class_v<T>
void Serialization::serialize_struct(std::ostream& output, T const& x) {
  output << "{ ";
  template for (constexpr auto base : bases_of(^T)) {
    serialize_object(output,
                     static_cast<typename [: type_of(base) :] const &>(x));
  }
  template for (constexpr auto member : nonstatic_data_members_of(^T)) {
    serialize_object(output, x.[: member :]);
  }
  output << "} ";
}
```

https://godbolt.org/z/PvY8nGx6o

Serialization

Serialize struct

```
template <typename T> requires std::is_class_v<T>
void Serialization::serialize_struct(std::ostream& output, T const& x) {
  output << "{ ";
  template for (constexpr auto base : bases_of(^T)) {
    serialize_object(output,
                     static_cast<typename [: type_of(base) :] const &>(x));
  }
  template for (constexpr auto member : nonstatic_data_members_of(^T)) {
    serialize_object(output, x.[: member :]);
  }
  output << "} ";
}
```

https://godbolt.org/z/PvY8nGx6o

Serialization

Serialize struct

```
template <typename T> requires std::is_class_v<T>
void Serialization::serialize_struct(std::ostream& output, T const& x) {
  output << "{ ";
  template for (constexpr auto base : bases_of(^T)) {
    serialize_object(output,
                     static_cast<typename [: type_of(base) :] const &>(x));
  }
  template for (constexpr auto member : nonstatic_data_members_of(^T)) {
    serialize_object(output, x.[: member :]);
  }
  output << "} ";
}
```

https://godbolt.org/z/PvY8nGx6o

Serialization

Serialize struct

```
template <typename T> requires std::is_class_v<T>
void Serialization::serialize_struct(std::ostream& output, T const& x) {
  output << "{ ";
  template for (constexpr auto base : bases_of(^T)) {
    serialize_object(output,
                     static_cast<typename [: type_of(base) :] const &>(x));
  }
  template for (constexpr auto member : nonstatic_data_members_of(^T)) {
    serialize_object(output, x.[: member :]);
  }
  output << "} ";
}
```

https://godbolt.org/z/PvY8nGx6o

Serialization

Serialize struct

```
template <typename T> requires std::is_class_v<T>
void Serialization::serialize_struct(std::ostream& output, T const& x) {
  output << "{ ";
  template for (constexpr auto base : bases_of(^T)) {
    serialize_object(output,
                     static_cast<typename [: type_of(base) :] const &>(x));
  }
  template for (constexpr auto member : nonstatic_data_members_of(^T)) {
    serialize_object(output, x.[: member :]);
  }
  output << "} ";
}
```

https://godbolt.org/z/PvY8nGx6o

Serialization

Serialize struct

```
template <typename T> requires std::is_class_v<T>
void Serialization::serialize_struct(std::ostream& output, T const& x) {
  output << "{ ";
  template for (constexpr auto base : bases_of(^T)) {
    serialize_object(output,
                     static_cast<typename [: type_of(base) :] const &>(x));
  }
  template for (constexpr auto member : nonstatic_data_members_of(^T)) {
    serialize_object(output, x.[: member :]);
  }
  output << "} ";
}
```

https://godbolt.org/z/PvY8nGx6o

Serialization

Deserialize struct

```
template <typename T> requires std::is_class_v<T>
void Serialization::deserialize_struct(std::istream& input, T& x) {
  input.ignore(2, '{');
  template for (constexpr auto base : bases_of(^T)) {
    deserialize_object(input, static_cast<typename [: type_of(base) :]&>(x));
  }
  template for (constexpr auto member : nonstatic_data_members_of(^T)) {
    if constexpr (is_bit_field(member)) {
      typename [: type_of(member) :] value;
      deserialize_arithmetic(input, value);
      x.[: member :] = value;
    } else {
      deserialize_object(input, x.[: member :]);
    }
  }
  input.ignore(2, '}');
}
```

https://godbolt.org/z/PvY8nGx6o

Serialization

Deserialize struct

```
template <typename T> requires std::is_class_v<T>
void Serialization::deserialize_struct(std::istream& input, T& x) {
  input.ignore(2, '{');
  template for (constexpr auto base : bases_of(^T)) {
    deserialize_object(input, static_cast<typename [: type_of(base) :]&>(x));
  }
  template for (constexpr auto member : nonstatic_data_members_of(^T)) {
    if constexpr (is_bit_field(member)) {
      typename [: type_of(member) :] value;
      deserialize_arithmetic(input, value);
      x.[: member :] = value;
    } else {
      deserialize_object(input, x.[: member :]);
    }
  }
  input.ignore(2, '}');
}
```

https://godbolt.org/z/PvY8nGx6o

Serialization

Lessons

- Reflection can handle many classes automatically
  - but not all
- Problem areas
  - References
  - Unions
  - Non-assignable members
  - Pointers
    - Workaround: smarter serialization framework

Serialization

Lessons

A good serialization framework must allow for user control over how a class is serialized

Future directions

Token Sequences

Reflection of arbitrary tokens

^{ *balanced-brace-tokens* }

Has type `std::meta::info`

```
constexpr auto t1 = ^{ a + b };       // three tokens
static_assert(std::is_same_v<decltype(t1), const std::meta::info>);
constexpr auto t2 = ^{ a += ( };      // code need not be meaningful
constexpr auto t3 = ^{ abc { def };   // Error, unpaired brace
```

https://wg21.link/p3294

Token Sequences

Reflection of arbitrary tokens

^{ *balanced-brace-tokens* }

Has type `std::meta::info`

```
constexpr auto t1 = ^{ a + b };       // three tokens
static_assert(std::is_same_v<decltype(t1), const std::meta::info>);
constexpr auto t2 = ^{ a += ( };      // code need not be meaningful
constexpr auto t3 = ^{ abc { def };   // Error, unpaired brace
```

https://wg21.link/p3294

Token Sequences

Reflection of arbitrary tokens

^{ *balanced-brace-tokens* }

Has type `std::meta::info`

```
constexpr auto t1 = ^{ a + b };       // three tokens
static_assert(std::is_same_v<decltype(t1), const std::meta::info>);
constexpr auto t2 = ^{ a += ( };      // code need not be meaningful
constexpr auto t3 = ^{ abc { def };   // Error, unpaired brace
```

https://wg21.link/p3294

Token Sequences

Reflection of arbitrary tokens

^{ *balanced-brace-tokens* }

Has type `std::meta::info`

```
constexpr auto t1 = ^{ a + b };       // three tokens
static_assert(std::is_same_v<decltype(t1), const std::meta::info>);
constexpr auto t2 = ^{ a += ( };      // code need not be meaningful
constexpr auto t3 = ^{ abc { def };   // Error, unpaired brace
```

https://wg21.link/p3294

Token Sequences

Reflection of arbitrary tokens

^{ *balanced-brace-tokens* }

Has type **`std::meta::info`**

```
constexpr auto t1 = ^{ a + b };       // three tokens
static_assert(std::is_same_v<decltype(t1), const std::meta::info>);
constexpr auto t2 = ^{ a += ( };      // code need not be meaningful
constexpr auto t3 = ^{ abc { def };   // Error, unpaired brace
```

https://wg21.link/p3294

Token Sequences

Reflection of arbitrary tokens

^{ *balanced-brace-tokens* }

Has type `std::meta::info`

```
constexpr auto t1 = ^{ a + b };       // three tokens
static_assert(std::is_same_v<decltype(t1), const std::meta::info>);
constexpr auto t2 = ^{ a += ( };      // code need not be meaningful
constexpr auto t3 = ^{ abc { def };   // Error, unpaired brace
```

https://wg21.link/p3294

`consteval` blocks

Evaluate code at compile time

```
consteval {
      statement;
      statement;
      …
}
```

Can be used anywhere a `static_assert` is allowed

https://wg21.link/p3289

Token Sequences

Injection

```
std::meta::queue_injection(
                  std::meta::info tokens)
```

Injects the token sequence at the end of the `consteval` block

```
std::meta::namespace_inject(
                  std::meta::info ns,
                  std::meta::info tokens)
```

Immediately injects the token sequence into the given namespace

https://wg21.link/p3294

Token Sequences

Injection example

```
constexpr auto define = ^{ constexpr int x = 42; };
constexpr auto check = ^{ static_assert(N::x == 42); };

namespace N { }

consteval {
  queue_injection(check);
  namespace_inject(^N, define);
}
```

https://godbolt.org/z/EdbPdE5n6

Token Sequences

Interpolation

Replace tokens with information from the environment

`\(`*expression*`)`

replaced by a pseudo-literal token with the value of the *expression*

`\id(`*string*, ...`)`

replaced by an identifier token that is the concatenation of the strings

`\tokens(`*expression*`)`

replaced by the given tokens; *expression* must be the reflection of a token sequence

https://wg21.link/p3294

Token Sequences

Interpolation

Replace tokens with information from the environment

`\(`*`expression`*`)`

replaced by a pseudo-literal token with the value of the *expression*

`\id(`*string*`, ...)`

replaced by an identifier token that is the concatenation of the strings

`\tokens(`*expression*`)`

replaced by the given tokens; *expression* must be the reflection of a token sequence

https://wg21.link/p3294

Token Sequences

Interpolation

Replace tokens with information from the environment

`\(`*expression*`)`

replaced by a pseudo-literal token with the value of the *expression*

`\id(`*`string`*`, ...)`

replaced by an identifier token that is the concatenation of the strings

`\tokens(`*expression*`)`

replaced by the given tokens; *expression* must be the reflection of a token sequence

https://wg21.link/p3294

Token Sequences

Interpolation

Replace tokens with information from the environment

`\(`*expression*`)`

replaced by a pseudo-literal token with the value of the *expression*

`\id(`*string*, ...`)`

replaced by an identifier token that is the concatenation of the strings

`\tokens(`*`expression`*`)`

replaced by the given tokens; *expression* must be the reflection of a token sequence

https://wg21.link/p3294

Token Sequences

Interpolation example

```
template <typename T>
consteval auto define(std::string_view name, T val) {
  return ^{ constexpr [:\(^T):] \id(name) = \(val); };
}
template <typename T>
consteval auto check(std::meta::info ns, std::string_view name, T val) {
  return ^{ static_assert([:\(ns):]::\id(name) == \(val)); };
}
namespace N {}
consteval {
  queue_injection(check(^N, "x", 42));
  namespace_inject(^N, define("x", 42));
}
```

https://godbolt.org/z/5Me4Kn8db

Token Sequences

Interpolation example

```
template <typename T>
consteval auto define(std::string_view name, T val) {
  return ^{ constexpr [:\(^T):] \id(name) = \(val); };
}
template <typename T>
consteval auto check(std::meta::info ns, std::string_view name, T val) {
  return ^{ static_assert([:\(ns):]::\id(name) == \(val)); };
}
namespace N {}
consteval {
  queue_injection(check(^N, "x", 42));
  namespace_inject(^N, define("x", 42));
}
```

https://godbolt.org/z/5Me4Kn8db

Token Sequences

Interpolation example

```
template <typename T>
consteval auto define(std::string_view name, T val) {
  return ^{ constexpr [:\(^T):] \id(name) = \(val); };
}
template <typename T>
consteval auto check(std::meta::info ns, std::string_view name, T val) {
  return ^{ static_assert([:\(ns):]::\id(name) == \(val)); };
}
namespace N {}
consteval {
  queue_injection(check(^N, "x", 42));
  namespace_inject(^N, define("x", 42));
}
```

https://godbolt.org/z/5Me4Kn8db

Token Sequences

Interpolation example

```
template <typename T>
consteval auto define(std::string_view name, T val) {
  return ^{ constexpr [:\(^T):] \id(name) = \(val); };
}
template <typename T>
consteval auto check(std::meta::info ns, std::string_view name, T val) {
  return ^{ static_assert([:\(ns):]::\id(name) == \(val)); };
}
namespace N {}
consteval {
  queue_injection(check(^N, "x", 42));
  namespace_inject(^N, define("x", 42));
}
```

https://godbolt.org/z/5Me4Kn8db

Token Sequences

Interpolation example

```
template <typename T>
consteval auto define(std::string_view name, T val) {
  return ^{ constexpr [:\(^T):] \id(name) = \(val); };
}
template <typename T>
consteval auto check(std::meta::info ns, std::string_view name, T val) {
  return ^{ static_assert([:\(ns):]::\id(name) == \(val)); };
}
namespace N {}
consteval {
  queue_injection(check(^N, "x", 42));
  namespace_inject(^N, define("x", 42));
}
```

https://godbolt.org/z/5Me4Kn8db

Token Sequences

Interpolation example

```
template <typename T>
consteval auto define(std::string_view name, T val) {
  return ^{ constexpr [:\(^T):] \id(name) = \(val); };
}
template <typename T>
consteval auto check(std::meta::info ns, std::string_view name, T val) {
  return ^{ static_assert([:\(ns):]::\id(name) == \(val)); };
}
namespace N {}
consteval {
  queue_injection(check(^N, "x", 42));
  namespace_inject(^N, define("x", 42));
}
```

https://godbolt.org/z/5Me4Kn8db

Token Sequences

Interpolation example

```
template <typename T>
consteval auto define(std::string_view name, T val) {
  return ^{ constexpr [:\(^T):] \id(name) = \(val); };
}
template <typename T>
consteval auto check(std::meta::info ns, std::string_view name, T val) {
  return ^{ static_assert([:\(ns):]::\id(name) == \(val)); };
}
namespace N {}
consteval {
  queue_injection(check(^N, "x", 42));
  namespace_inject(^N, define("x", 42));
}
```

https://godbolt.org/z/5Me4Kn8db

Token Sequences

Example: enable_if

```
template <bool B, class T = void>
struct enable_if { };

template <class T>
struct enable_if<true, T> {
    using type = T;
};
```

Token Sequences

Example: enable_if

```
template <bool B, class T = void>
struct enable_if { };

template <class T>
struct enable_if<true, T> {
    using type = T;
};
```

Where's the "if"?

Token Sequences

Example: enable_if

```
template <bool B, class T = void>
struct enable_if {
    consteval {
        if (B) {
            queue_injection(^{ using type = T; });
        }
    }
};
```

https://godbolt.org/z/jfMoe34Ea

Token Sequences

Example: tuple

```
template <class... Ts>
struct Tuple {
  consteval {
    std::meta::info types[] = {^Ts...};
    for (size_t i = 0; i != sizeof...(Ts); ++i) {
      queue_injection(^{ [:\(types[i]):] \id("_", i); });
    }
  }
};
```

https://godbolt.org/z/861MsqzPx

Token Sequences

Example: properties

```
consteval void property(std::meta::info type, std::string_view name) {
  auto member = ^{ \id("m_"sv, name) };
  queue_injection(^{ [:\(type):] \tokens(member); });
  queue_injection(^{
    auto \id("get_"sv, name)() -> [:\(type):] const& {
      return \tokens(member);
    }
  });
  queue_injection(^{
    auto \id("set_"sv, name)(typename [:\(type):] const& x) -> void {
      \tokens(member) = x;
    }
  });
}
```

https://godbolt.org/z/sqKs6eKzG

Token Sequences

Example: properties

```
consteval void property(std::meta::info type, std::string_view name) {
  auto member = ^{ \id("m_"sv, name) };
  queue_injection(^{ [:\(type):] \tokens(member); });
  queue_injection(^{
    auto \id("get_"sv, name)() -> [:\(type):] const& {
      return \tokens(member);
    }
  });
  queue_injection(^{
    auto \id("set_"sv, name)(typename [:\(type):] const& x) -> void {
      \tokens(member) = x;
    }
  });
}
```

https://godbolt.org/z/sqKs6eKzG

Token Sequences

Example: properties

```
consteval void property(std::meta::info type, std::string_view name) {
  auto member = ^{ \id("m_"sv, name) };
  queue_injection(^{ [:\(type):] \tokens(member); });
  queue_injection(^{
    auto \id("get_"sv, name)() -> [:\(type):] const& {
      return \tokens(member);
    }
  });
  queue_injection(^{
    auto \id("set_"sv, name)(typename [:\(type):] const& x) -> void {
      \tokens(member) = x;
    }
  });
}
```

https://godbolt.org/z/sqKs6eKzG

Token Sequences

Example: properties

```
consteval void property(std::meta::info type, std::string_view name) {
  auto member = ^{ \id("m_"sv, name) };
  queue_injection(^{ [:\(type):] \tokens(member); });
  queue_injection(^{
    auto \id("get_"sv, name)() -> [:\(type):] const& {
      return \tokens(member);
    }
  });
  queue_injection(^{
    auto \id("set_"sv, name)(typename [:\(type):] const& x) -> void {
      \tokens(member) = x;
    }
  });
}
```

https://godbolt.org/z/sqKs6eKzG

Token Sequences

Example: properties

```
consteval void property(std::meta::info type, std::string_view name) {
  auto member = ^{ \id("m_"sv, name) };
  queue_injection(^{ [:\(type):] \tokens(member); });
  queue_injection(^{
    auto \id("get_"sv, name)() -> [:\(type):] const& {
      return \tokens(member);
    }
  });
  queue_injection(^{
    auto \id("set_"sv, name)(typename [:\(type):] const& x) -> void {
      \tokens(member) = x;
    }
  });
}
```

https://godbolt.org/z/sqKs6eKzG

Token Sequences

Example: properties

```
consteval void property(std::meta::info type, std::string_view name) {
  auto member = ^{ \id("m_"sv, name) };
  queue_injection(^{ [:\(type):] \tokens(member); });
  queue_injection(^{
    auto \id("get_"sv, name)() -> [:\(type):] const& {
      return \tokens(member);
    }
  });
  queue_injection(^{
    auto \id("set_"sv, name)(typename [:\(type):] const& x) -> void {
      \tokens(member) = x;
    }
  });
}
```

https://godbolt.org/z/sqKs6eKzG

Token Sequences

Example: properties

```
consteval void property(std::meta::info type, std::string_view name) {
  auto member = ^{ \id("m_"sv, name) };
  queue_injection(^{ [:\(type):] \tokens(member); });
  queue_injection(^{
    auto \id("get_"sv, name)() -> [:\(type):] const& {
      return \tokens(member);
    }
  });
  queue_injection(^{
    auto \id("set_"sv, name)(typename [:\(type):] const& x) -> void {
      \tokens(member) = x;
    }
  });
}
```

https://godbolt.org/z/sqKs6eKzG

Token Sequences

Example: properties

```
struct Book {
    consteval {
        property(^std::string, "title");
        property(^std::string, "author");
    }
};
```

https://godbolt.org/z/sqKs6eKzG

Token Sequences

Example: properties

```
struct Book {
  consteval {
    property(^std::string, "title");
    property(^std::string, "author");
  }
};
```

Token Sequences

Example: properties

```
struct Book {
  consteval {
    property(^std::string, "title");
    property(^std::string, "author");
  }
};
```

```
struct Book {
  std::string m_title;
  auto get_title() -> std::string const& {
    return m_title;
  }
  auto set_title(std::string const& x) -> void {
    m_title = x;
  }
  std::string m_author;
  auto get_author() -> std::string const& {
    return m_author;
  }
  auto set_author(std::string const& x) -> void {
    m_author = x;
  }
};
```

Meta Classes

“Meta: Thoughts on generative C++,” Herb Sutter, CppCon 2017 keynote

P0707 “Metaclass functions: Generative C++,” Herb Sutter

Takes code injection to the extreme

Built on top of reflection

https://www.youtube.com/watch?v=4AfRAVcThyA
https://wg21.link/p0707

Conclusion

Other Resources

Presentations

“Welcome to the meta::[[verse]]!” Inbal Levi, ACCU 2024 keynote link

“Reflection is Good for (Code) Health,” Saksham Sharma, C++Now 2024 link

Other Resources

Presentations

“Welcome to the meta::[[verse]]!” Inbal Levi, ACCU 2024 keynote link

“Reflection is Good for (Code) Health,” Saksham Sharma, C++Now 2024 link

“C++ Reflection: Back on Track,” David Olsen, C++Now 2024 link

Other Resources

Committee papers

P2996 “Reflection for C++26”

P1306 “Expansion statements”

P3068 “Allowing exception throwing in constant-evaluation”

P3096 “Function Parameter Reflection”

P3157 “Generative Extensions for Reflection”

P3289 “`consteval` blocks”

P3293 “Splicing a base class subobject”

P3294 “Code Injection with Token Sequences”

Conclusion

C++ reflection is coming

Compile-time

Reduce boilerplate code