C++ REFLECTION TS

C++Now 2019
David Sankel
Bloomberg

EXCELLENT COAUTHORS

REFERENCES

- N4766: Latest Reflection TS Draft
- P0578: Static Reflection in a Nutshell
- P0385: Static Reflection Design
- P0194: Base wording
- P0670: Function Reflection
- Reference implementation. https://github.com/matus-chochlik/clang

WHAT IS REFLECTION?

```cpp
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = T;
};
```

```cpp
GetInnerType<int>::type // int
GetInnerType<std::vector<int>>::type // int
GetInnerType<std::vector<std::vector<int>>>::type // vector<in
```

```
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = T;
};
```

```
GetInnerType<int>::type // int
GetInnerType<std::vector<int>>::type // int
GetInnerType<std::vector<std::vector<int>>>::type // vector<in
```

```cpp
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = T;
};
```

```cpp
GetInnerType<int>::type // int
GetInnerType<std::vector<int>>::type // int
GetInnerType<std::vector<std::vector<int>>>::type // vector<in
```

```cpp
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = T;
};
```

```cpp
GetInnerType<int>::type // int
GetInnerType<std::vector<int>>::type // int
GetInnerType<std::vector<std::vector<int>>>::type // vector<in
```

```
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = T;
};
```

```
GetInnerType<int>::type // int
GetInnerType<std::vector<int>>::type // int
GetInnerType<std::vector<std::vector<int>>>::type // vector<in
```

```cpp
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = T;
};
```

```cpp
GetInnerType<int>::type // int
GetInnerType<std::vector<int>>::type // int
GetInnerType<std::vector<std::vector<int>>>::type // vector<in
```

```cpp
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = typename GetInnerType<T>::type;
};
```

```cpp
GetInnerType<std::vector<std::vector<int>>>::type // int
```

```cpp
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = typename GetInnerType<T>::type;
};
```

```cpp
GetInnerType<std::vector<std::vector<int>>>::type // int
```

```cpp
template<typename T>
struct GetInnerType {
    using type = T;
};

template<typename T>
struct GetInnerType<std::vector<T>> {
    using type = typename GetInnerType<T>::type;
};
```

```cpp
GetInnerType<std::vector<std::vector<int>>>::type // int
```

TYPE TRAITS

```
is_void
is_integral
is_class
is_union
is_function
is_assignable
is_destructable
```

```cpp
template <class Fn, class... ArgTypes>
struct is_invocable;
```

BOOST.FUNCTIONTYPES

```
template<typename F, class ClassTransform = add_reference<_> >
struct parameter_types;
```

- Extracts parameter types and produces an MPL sequence.

Key observation: *C++ already has compile-time reflection*

APPLICATIONS OF REFLECTION SO FAR

- Boost.Bind (`std::bind`)
- Boost.HOF (higher order functions, e.g. fold)
- Boost.Parameter
- Boost.Python
- Boost.TTI

SOMETHING'S MISSING

BOOST.SERIALIZATION

```
class bus_stop {
    friend class boost::serialization::access;
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version)
    {
        ar & latitude;
        ar & longitude;
    }
    gps_position latitude;
    gps_position longitude;
    //...
};
```

BOOST.SERIALIZATION

```
class bus_stop {
    friend class boost::serialization::access;
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version)
    {
        ar & latitude;
        ar & longitude;
    }
    gps_position latitude;
    gps_position longitude;
    //...
};
```

BOOST.SERIALIZATION WITH XML

```
class bus_stop {
    friend class boost::serialization::access;
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version)
    {
        ar & make_nvp("latitude", latitude);
        ar & make_nvp("longitude", longitude);
    }
    gps_position latitude;
    gps_position longitude;
    //...
};
```

BOOST.SERIALIZATION WITH XML

```
class bus_stop {
    friend class boost::serialization::access;
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version)
    {
        ar & make_nvp("latitude", latitude);
        ar & make_nvp("longitude", longitude);
    }
    gps_position latitude;
    gps_position longitude;
    //...
};
```

BOOST.SERIALIZATION WITH XML

```
class bus_stop {
    friend class boost::serialization::access;
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version)
    {
        ar & make_nvp("latitude", latitude);
        ar & make_nvp("longitude", longitude);
    }
    gps_position latitude;
    gps_position longitude;
    //...
};
```

BOOST_SERIALIZATION_NVP alternative

BOOST.FUSION

```
using namespace boost::fusion;
std::tuple<int,std::string> example_tuple(101, "hello");

std::cout << *begin(example_tuple) << '\n';       // 101
std::cout << *next(begin(example_tuple)) << '\n'; // hello
```

```
struct employee {
    std::string name;
    int age;
};

BOOST_FUSION_ADAPT_STRUCT(employee, name, age)
```

```
employee me{"David Sankel", 37};

std::cout << *begin(me) << '\n';       // David Sankel
std::cout << *next(begin(me)) << '\n'; // 37
```

```
struct employee {
    std::string name;
    int age;
};

BOOST_FUSION_ADAPT_STRUCT(employee, name, age)
```

```
employee me{"David Sankel", 37};

std::cout << *begin(me) << '\n';       // David Sankel
std::cout << *next(begin(me)) << '\n'; // 37
```

```cpp
struct employee {
    std::string name;
    int age;
};

BOOST_FUSION_ADAPT_STRUCT(employee, name, age)
```

```cpp
employee me{"David Sankel", 37};

std::cout << *begin(me) << '\n';       // David Sankel
std::cout << *next(begin(me)) << '\n'; // 37
```

```
BOOST_FUSION_DEFINE_STRUCT(
    ,
    employee,
    (std::string, name)
    (int, age))
```

```
BOOST_FUSION_DEFINE_STRUCT(
    ,
    employee,
    (std::string, name)
    (int, age))
```

```
BOOST_FUSION_DEFINE_STRUCT(
    ,
    employee,
    (std::string, name)
    (int, age))
```

```
BOOST_FUSION_DEFINE_STRUCT(
    ,
    employee,
    (std::string, name)
    (int, age))
```

REFLECTION TS FOCUS

REFLECTION TS FOCUS

- Provide (only) what's missing

REFLECTION TS FOCUS

- Provide (only) what's missing
- Focus on use cases

REFLECTION TS FOCUS

- Provide (only) what's missing
- Focus on use cases
- Focus on implementability

REFLECTION TS FOCUS

- Provide (only) what's missing
- Focus on use cases
- Focus on implementability
- Make the API low level

User Code

↑

Domain Specific Library

↑

Reflection Library

↑

Reflection Facilities

```
reflexpr(<syntax>)
reflexpr(<syntax>)
```

- Evaluates to a unnamed type

```
reflexpr(<syntax>)
```

- Evaluates to a unnamed type
- That type satisfies concepts

```
namespace std::experimental::reflect {
inline namespace v1 {

template <class T>
concept Object = /*...*/;
```

```
template <Object T1, Object T2> struct reflects_same;
template <Object T> struct get_source_line;
template <Object T> struct get_source_column;
template <Object T> struct get_source_file_name;
```

```
namespace std::experimental::reflect {
inline namespace v1 {

template <class T>
concept Object = /*...*/;
```

```
template <Object T1, Object T2> struct reflects_same;
template <Object T> struct get_source_line;
template <Object T> struct get_source_column;
template <Object T> struct get_source_file_name;
```

Every `reflexpr` result satisfies `Object`

```
namespace std::experimental::reflect {
inline namespace v1 {

template <class T>
concept Object = /*...*/;
```

```
template <Object T1, Object T2> struct reflects_same;
template <Object T> struct get_source_line;
template <Object T> struct get_source_column;
template <Object T> struct get_source_file_name;
```

Concepts have associated operations

```cpp
template <class T>
constexpr auto get_source_line_v = get_source_line<T>::value;
```

_v and _t shorthands are included

```cpp
struct S { // line 0
    int i; // line 1
};         // line 2
```

```cpp
get_source_line_v<reflexpr(S)>;
```

```
struct S { // line 0
    int i; // line 1
};         // line 2
```

```
get_source_line_v<reflexpr(S)>;
```

Evaluates to 0

TYPE SEQUENCES

```
template <ObjectSequence T>
constexpr auto get_size_v = get_size<T>::value;

template <ObjectSequence T1, size_t Index>
using get_element_t = typename get_element<T1, Index>::type;

template <ObjectSequence T1, template <class...> class Tpl>
constexpr auto unpack_sequence_v = unpack_sequence<T1, Tpl>::v
```

TYPE SEQUENCES

```
template <ObjectSequence T>
constexpr auto get_size_v = get_size<T>::value;

template <ObjectSequence T1, size_t Index>
using get_element_t = typename get_element<T1, Index>::type;

template <ObjectSequence T1, template <class...> class Tpl>
constexpr auto unpack_sequence_v = unpack_sequence<T1, Tpl>::v
```

Get sequence size

TYPE SEQUENCES

```
template <ObjectSequence T>
constexpr auto get_size_v = get_size<T>::value;

template <ObjectSequence T1, size_t Index>
using get_element_t = typename get_element<T1, Index>::type;

template <ObjectSequence T1, template <class...> class Tpl>
constexpr auto unpack_sequence_v = unpack_sequence<T1, Tpl>::v
```

Get sequence element

TYPE SEQUENCES

```
template <ObjectSequence T>
constexpr auto get_size_v = get_size<T>::value;

template <ObjectSequence T1, size_t Index>
using get_element_t = typename get_element<T1, Index>::type;

template <ObjectSequence T1, template <class...> class Tpl>
constexpr auto unpack_sequence_v = unpack_sequence<T1, Tpl>::v
```

Unpack the sequence into a template

```
template <ObjectSequence T1, template <class...> class Tpl>
constexpr auto unpack_sequence_v = unpack_sequence<T1, Tpl>::v
```

```
unpack_sequence_v<SomeSequence, boost::fusion::vector>
```

```
template <ObjectSequence T1, template <class...> class Tpl>
constexpr auto unpack_sequence_v = unpack_sequence<T1, Tpl>::v
```

```
unpack_sequence_v<SomeSequence, boost::fusion::vector>
```

Create a boost::fusion::vector

WHAT'S IN? WHAT'S OUT?

In:

- Data members
- Member types
- Enumerators
- Templates instantiations
- Aliases
- Function declarations
- Lambdas

Out:

- Reflection facilities already in C++
- Namespace members
- Templates
- Building new datatypes (`reflid`)
- Attributes

```
struct S {
    int i;
};
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v< get_type_t<FirstDataMember>
```

```
struct S {
    int i;
};
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v< get_type_t<FirstDataMember>
```

```
struct S {
    int i;
};
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v< get_type_t<FirstDataMember>
```

Get the data members

```
struct S {
    int i;
};
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v< get_type_t<FirstDataMember>
```

Extract the first one

```
struct S {
    int i;
};
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v< get_type_t<FirstDataMember>
```

Get the name, "i"

```
struct S {
    int i;
};
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v< get_type_t<FirstDataMember>
```

Get the type name, "int"

CONCEPTS IN USE

```
struct S { int i; };
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v<
    get_type_t<FirstDataMember> >;
```

CONCEPTS IN USE

```
struct S { int i; };
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v<
    get_type_t<FirstDataMember> >;
```

Class → Record

CONCEPTS IN USE

```
struct S { int i; };
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v<
    get_type_t<FirstDataMember> >;
```

Sequence

CONCEPTS IN USE

```
struct S { int i; };
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v<
    get_type_t<FirstDataMember> >;
```

RecordMember → ScopeMember → Named

CONCEPTS IN USE

```
struct S { int i; };
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v<
    get_type_t<FirstDataMember> >;
```

Variable → Typed

CONCEPTS IN USE

```
struct S { int i; };
```

```
using FirstDataMember =
  get_element_t<
    get_public_data_members_t<
      reflexpr(S) >,
    0 >;
```

```
std::string memberName = get_name_v<FirstDataMember>;
std::string typeName = get_name_v<
    get_type_t<FirstDataMember> >;
```

Type → Named

CORE CONCEPTS

Object
ObjectSequence
TemplateParameterScope
Named
Alias
RecordMember
Enumerator
Variable
ScopeMember
Typed
Namespace
GlobalScope
Class
Enum
Record
Scope
Type
Constant
Base

RECORD

```
template <Record T> struct get_data_members;
template <Record T> struct get_public_data_members;
template <Record T> struct get_accessible_data_members;

template <Record T> struct get_member_types;
template <Record T> struct get_public_member_types;
template <Record T> struct get_accessible_member_types;
```

RECORD

```
template <Record T> struct get_data_members;
template <Record T> struct get_public_data_members;
template <Record T> struct get_accessible_data_members;

template <Record T> struct get_member_types;
template <Record T> struct get_public_member_types;
template <Record T> struct get_accessible_member_types;
```

Get all members. "unsafe"

RECORD

```
template <Record T> struct get_data_members;
template <Record T> struct get_public_data_members;
template <Record T> struct get_accessible_data_members;

template <Record T> struct get_member_types;
template <Record T> struct get_public_member_types;
template <Record T> struct get_accessible_member_types;
```

Get public data members

RECORD

```
template <Record T> struct get_data_members;
template <Record T> struct get_public_data_members;
template <Record T> struct get_accessible_data_members;

template <Record T> struct get_member_types;
template <Record T> struct get_public_member_types;
template <Record T> struct get_accessible_member_types;
```

Members available in this context

GOING FROM META TO REAL

```
template <Type T>
using get_reflected_type_t = typename get_reflected_type<T>::t

template <Variable T>
constexpr auto get_pointer_v = get_pointer<T>::value;
```

```
get_reflected_type_t<
    get_type_t<FirstDataMember> > i = 3;
```

GOING FROM META TO REAL

```
template <Type T>
using get_reflected_type_t = typename get_reflected_type<T>::t

template <Variable T>
constexpr auto get_pointer_v = get_pointer<T>::value;
```

```
get_reflected_type_t<
    get_type_t<FirstDataMember> > i = 3;
```

"meta" type to type

GOING FROM META TO REAL

```
template <Type T>
using get_reflected_type_t = typename get_reflected_type<T>::t

template <Variable T>
constexpr auto get_pointer_v = get_pointer<T>::value;
```

```
get_reflected_type_t<
    get_type_t<FirstDataMember> > i = 3;
```

To `int i = 3`

GOING FROM META TO REAL

```
template <Type T>
using get_reflected_type_t = typename get_reflected_type<T>::t

template <Variable T>
constexpr auto get_pointer_v = get_pointer<T>::value;
```

```
get_reflected_type_t<
    get_type_t<FirstDataMember> > i = 3;
```

Get pointer to data member

NAMED

```
template <Named T>
constexpr auto get_name_v = get_name<T>::value;
template <Named T>
constexpr auto get_display_name_v = get_display_name<T>::value
```

NAMED

```
template <Named T>
constexpr auto get_name_v = get_name<T>::value;
template <Named T>
constexpr auto get_display_name_v = get_display_name<T>::value
```

NAMED

```
template <Named T>
constexpr auto get_name_v = get_name<T>::value;
template <Named T>
constexpr auto get_display_name_v = get_display_name<T>::value
```

APPLICATIONS

SERIALIZATION

```
struct C { /*...*/ };
struct D {
  std::string s;
  C c;
};
```

```
D d = /*...*/
std::cout << to_json(d);
```

SERIALIZATION

```
struct C { /*...*/ };
struct D {
  std::string s;
  C c;
};
```

```
D d = /*...*/
std::cout << to_json(d);
```

```
{ s: "hello", c: {/*...*/} }
```

```
class bus_stop {
    friend class boost::serialization::access;
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version)
    {
        ar & make_nvp("latitude", latitude);
        ar & make_nvp("longitude", longitude);
    }
    gps_position latitude;
    gps_position longitude;
    //...
};
```

```
class bus_stop : public boost::serialization2<bus_stop> {
    gps_position latitude;
    gps_position longitude;
    //...
};
```

```
class bus_stop : public boost::serialization2<bus_stop> {
    gps_position latitude;
    gps_position longitude;
    //...
};
```

CRTP

BOOST.OPERATORS

```
struct person : public boost::less_than_comparable<person>
{
  std::string name;
  int id;

  bool operator<(const person &a) const {
    return std::tie(name, id) < std::tie(a.name, a.id);
  }
};
```

```cpp
struct person : public boost::operators2::comparisons<person>
{
  std::string name;
  int id;
};
```

```cpp
struct person : public boost::operators2::strong_ordering<pers
{
  std::string name;
  int id;
};
```

```
struct person : public boost::operators2::strong_ordering<pers
{
  std::string name;
  int id;
};
```

We didn't really need <=>. Oops

```cpp
struct School {
  std::string name;
  double gpa;
};

struct Job {
  std::string name;
  std::vector<std::string> references;
};

struct Candidate {
  std::string first_name;
  std::string last_name;
  std::variant<School, Job> most_recent_setting;
};
```

JACKIE KAY COMMAND-LINE PARSER

```cpp
struct ParsedCommandLine {
  std::optional<int> verbosity;
  ServiceId serviceid;
};

int main( int argc, char** argv ) {
  ParsedCommandLine cmdLine
    = boost::args::parseCommandLine<ParsedCommandLine>(argc, a
  if(cmdLine.verbose)
    //...
}
```

```
program --verbosity=3 --serviceid=293923:1.0
program --serviceid=293923:1.0
program # error serviceid not specified ...
program --help
```

JACKIE KAY COMMAND-LINE PARSER

```
struct ParsedCommandLine {
  std::optional<int> verbosity;
  ServiceId serviceid;
};

int main( int argc, char** argv ) {
  ParsedCommandLine cmdLine
    = boost::args::parseCommandLine<ParsedCommandLine>(argc, a
  if(cmdLine.verbose)
    //...
}
```

```
program --verbosity=3 --serviceid=293923:1.0
program --serviceid=293923:1.0
program # error serviceid not specified ...
program --help
```

Type driven development

JACKIE KAY COMMAND-LINE PARSER

```
struct ParsedCommandLine {
  std::optional<int> verbosity;
  ServiceId serviceid;
};

int main( int argc, char** argv ) {
  ParsedCommandLine cmdLine
    = boost::args::parseCommandLine<ParsedCommandLine>(argc, a
  if(cmdLine.verbose)
    //...
}
```

```
program --verbosity=3 --serviceid=293923:1.0
program --serviceid=293923:1.0
program # error serviceid not specified ...
program --help
```

Fictional Boost library

JACKIE KAY COMMAND-LINE PARSER

```
struct ParsedCommandLine {
  std::optional<int> verbosity;
  ServiceId serviceid;
};

int main( int argc, char** argv ) {
  ParsedCommandLine cmdLine
    = boost::args::parseCommandLine<ParsedCommandLine>(argc, a
  if(cmdLine.verbose)
    //...
}
```

```
program --verbosity=3 --serviceid=293923:1.0
program --serviceid=293923:1.0
program # error serviceid not specified ...
program --help
```

JACKIE KAY COMMAND-LINE PARSER

```
struct ParsedCommandLine {
  std::optional<int> verbosity;
  ServiceId serviceid;
};

int main( int argc, char** argv ) {
  ParsedCommandLine cmdLine
    = boost::args::parseCommandLine<ParsedCommandLine>(argc, a
  if(cmdLine.verbose)
    //...
}
```

```
program --verbosity=3 --serviceid=293923:1.0
program --serviceid=293923:1.0
program # error serviceid not specified ...
program --help
```

std::optional treated specially

JACKIE KAY COMMAND-LINE PARSER

```
struct ParsedCommandLine {
  std::optional<int> verbosity;
  ServiceId serviceid;
};

int main( int argc, char** argv ) {
  ParsedCommandLine cmdLine
    = boost::args::parseCommandLine<ParsedCommandLine>(argc, a
  if(cmdLine.verbose)
    //...
}
```

```
program --verbosity=3 --serviceid=293923:1.0
program --serviceid=293923:1.0
program # error serviceid not specified ...
program --help
```

AND THAT'S NOT ALL...

FUNCTION REFLECTION

FunctionParameter
Callable
Expression
ParenthesizedExpression
FunctionCallExpression
FunctionalTypeConversion
Function
MemberFunction

SpecialMemberFunction
Constructor
Destructor
Operator
ConversionOperator
Lambda
LambdaCapture

```
class Foo {
  //...
public:
  std::vector<string> ls();
  void copy(std::string source, std::string destination);
  void touch(std::string path);
  //...
};
```

```
void main(int argc, char** argv) {
  Foo f;
  make_me_a_rest_service(argc, argv, f);
}
```

```
class Foo {
  //...
public:
  std::vector<string> ls();
  void copy(std::string source, std::string destination);
  void touch(std::string path);
  //...
};
```

```
void main(int argc, char** argv) {
  Foo f;
  make_me_a_rest_service(argc, argv, f);
}
```

```cpp
class Foo {
  //...
public:
  std::vector<string> ls();
  void copy(std::string source, std::string destination);
  void touch(std::string path);
  //...
};
```

```cpp
void main(int argc, char** argv) {
  Foo f;
  make_me_a_rest_service(argc, argv, f);
}
```

```
class Foo {
  //...
public:
  std::vector<string> ls();
  void copy(std::string source, std::string destination);
  void touch(std::string path);
  //...
};
```

```
void main(int argc, char** argv) {
  Foo f;
  make_me_an_interactive_console(argc, argv, f);
}
```

```cpp
class Foo {
  //...
public:
  std::vector<string> ls();
  void copy(std::string source, std::string destination);
  void touch(std::string path);
  //...
};
```

```cpp
void main(int argc, char** argv) {
  Foo f;
  make_me_an_interactive_console(argc, argv, f);
}
```

```
class Foo {
  //...
public:
  std::vector<string> ls();
  void copy(std::string source, std::string destination);
  void touch(std::string path);
  //...
};
```

```
> ls
foo
bar
> copy source=foo destination=baz
> ls
foo
bar
baz
```

```
class Foo {
  //...
public:
  std::vector<string> ls();
  static const std::string helpText_ls;
};
```

YOUR IMAGINATION IS THE LIMIT

TS STATUS

- Will be published shortly
- Official clang implementation in the works

FUTURE DIRECTION

Reface with `constexpr`-based syntax

- Make reflection more accessible
- Make metaprogramming more efficient
- See P0953…

C++ REFLECTION TS

C++Now 2019
David Sankel
Bloomberg