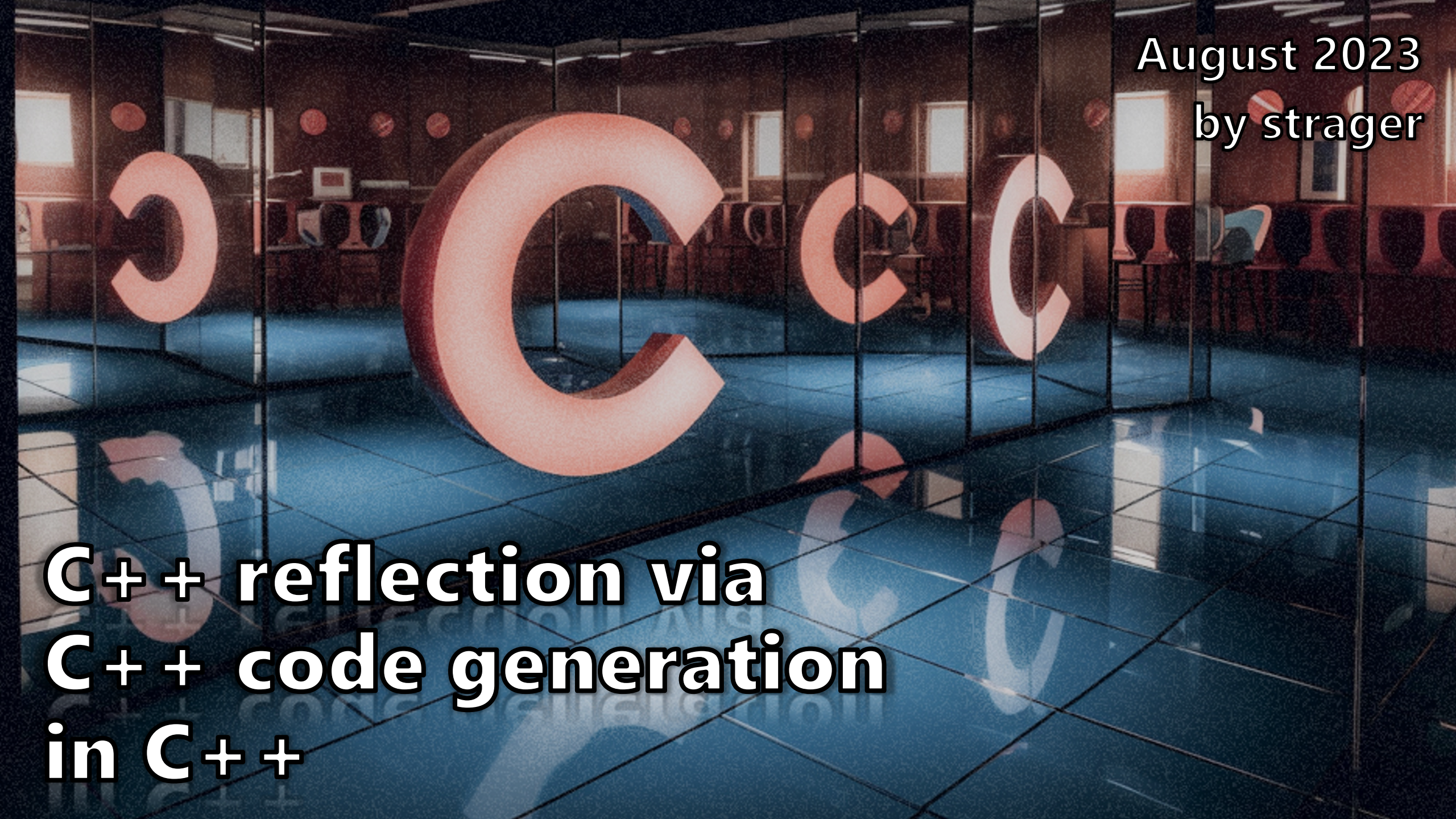

August 2023
by strager
C++ reflection via
C++ code generation
in C++

```
struct Diag_Unused_Var {
  Source_Location variable;
};
struct Diagnostic_Reporter {
  virtual void report(Diag_Unused_Var diag) = 0;
};
```

```
struct Diag_Unused_Var {
  Source_Location variable;
};
struct Diagnostic_Reporter {
         eport          l_Var

            er : D gnostic_Report     {
  void rep     Diag Unuse  _Var diag) ove ride {
    std::prin   n("{}: warning: unused variable",
                diag.variable);
  }
};
```

location
(file and line #)

severity

message

```
struct Diag_Unused_Var {
  Source_Location variable;
};
struct Diagnostic_Reporter {
  virtual void report(Diag                diag) = 0;
};
struct JSON_Reporter : Diagnostic_Report
  void report(Diag_Unused_Var diag) over
    fprintf(stderr, "{{{ {:json},
                        "severity": "warning",
                        "message": "unused variable"
                    }}}", diag.variable);
  }
```

severity

message

location
(file and line #)

```
struct Diag_Unused_Var {
  Source_Location variable;
};
struct Diagnostic_Reporter {
  virtual void report(Diag_Unused_Var diag) = 0;
};
```

```
struct Diag_Unused_Var {
  Source_Location variable;
};
struct Diagnostic_Reporter {



  virtual void report(Diag_Unused_Var diag) = 0;

};
```

```
struct Diag_Unused_Var {
  Source_Location variable;
};
struct Diagnostic_Reporter {

          disabled_categories.contains(
            Diagnostic_Category::dead_code)


  virtual void report(Diag_Unused_Var diag) = 0;
  std::set<Diagnostic_Category> disabled_categories;
};
```

```
struct Diag_Unused_Var {
  Source_Location variable;
};
struct Diagnostic_Reporter {
  void report(Diag_Unused_Var diag) {
    if (!disabled_categories.contains(
            Diagnostic_Category::dead_code))
      report_impl(diag);
  }
  virtual void report_impl(Diag_Unused_Var diag) = 0;
  std::set<Diagnostic_Category> disabled_categories;
};
```

```
struct Diag_Unused_Var {
  Source_Location variable;
};

struct Diagnostic_Reporter {
  void report(Diag_Unused_Var diag) {
    if (!disabled_categories.contains(
          Diagnostic_Category::dead_code))
      report_impl(diag);
  }
  virtual void report_impl(Diag_Unused_Var diag)=0;
  std::set<Diagnostic_Category> disabled_categories;
};
```

```
struct CLI_Diagnostic_Reporter : Diagnostic_Reporter {
  void report_impl(Diag_Unused_Variable diag) override {
    std::println("{}: warning: unused variable",
                 diag.variable);
  }
};

struct JSON_Reporter : Diagnostic_Reporter {
  void report_impl(Diag_Unused_Var diag) override {
    std::println(R"({{ {:json},
                       "severity": "warning",
                       "message": "unused variable"
                     }})", diag.variable);
  }
};
```

```
struct Diag_Unused_Var {
  Source_Location variable;
};

struct Diag_Unexpected_Token {
  Source_Location token;
};

struct Diagnostic_Reporter {
  void report(Diag_Unused_Var diag) {
    if (!disabled_categories.contains(
          Diagnostic_Category::dead_code))
      report_impl(diag);
  }

  void report(Diag_Unexpected_Token diag) {
    if (disabled_categories.contains(
          Diagnostic_Category::parse_error))
      report_impl(diag);
  }

  virtual void report_impl(Diag_Unused_Var diag)=0;
  virtual void report_impl(Diag_Unexpected_Token)=0;

  std::set<Diagnostic_Category> disabled_ca  es;
};
```


```
struct CLI_Diagnostic_Reporter : Diagnostic_Reporter {
  void report_impl(Diag_Unused_Variable diag) override {
    std::println("{}: warning: unused variable",
                 diag.variable);
  }

  void report_impl(Diag_Unexpected_Token diag) override {
    std::println("{}: error: unexpected token",
                 diag.token);
  }
};

struct JSON_Reporter : Diagnostic_Reporter {
  void report_impl(Diag_Unused_Var diag) override {
    std::println(R"({{ {:json},
                       "severity": "warning",
                       "message": "unused variable"
                    }})", diag.variable);
  }

  void report_impl(Diag_Unused_Var diag) override {
    std::println(R"({{ {:json},
                       "severity": "error",
                       "message": "unexpected token"
                    }})", diag.token);
  }
};
```

```
struct Diag_Unused_Var {
  Source_Location variable;
};

struct Diag_Unexpected_Token {
  Source_Location token;
};

struct Diagnostic_Reporter {
  void report(Diag_Unused_Var diag) {
    if (!disabled_categories.contains(
           Diagnostic_Category::dead_code))
      report_impl(diag);
  }

  void report(Diag_Unexpected_Token diag) {
    if (disabled_categories.contains(
          Diagnostic_Category::parse_error))
      report_impl(diag);
  }

  virtual void report_impl(Diag_Unused_Var diag)=0;
  virtual void report_impl(Diag_Unexpected_Token)=0;

  std::set<Diagnostic_Category> disabled_categories;
};
```

```
struct CLI_Diagnostic_Reporter : Diagnostic_Reporter {
  void report_impl(Diag_Unused_Variable diag) override {
    std::println("{}: warning: unused variable",
                 diag.variable);
  }

  void report_impl(Diag_Unexpected_Token diag) override {
    std::println("{}: error: unexpected token",
                 diag.token);
  }
};

struct JSON_Reporter : Diagnostic_Reporter {
  void report_impl(Diag_Unused_Var diag) override {
    std::println(R"({{ {:json},
                       "severity": "warning",
                       "message": "unused variable"
                    }})", diag.variable);
  }

  void report_impl(Diag_Unused_Var diag) override {
    std::println(R"({{ {:json},
                       "severity": "error",
                       "message": "unexpected token"
                    }})", diag.token);
  }
};
```

```
[[demo::severity(warning)]]
[[demo::category(dead_code)]]
[[demo::message(variable, "unused variable")]]
struct Diag_Unused_Var {
    Source_Location variable;
};

[[demo::severity(error)]]
[[demo::category(parse_error)]]
[[demo::message(token, "unexpected token")]]
struct Diag_Unexpected_Token {
    Source_Location token;
};
```

```
[[demo::severity(warning)]]
[[demo::category(dead_code)]]
[[demo::message(variable, "unused variable")]]
struct Diag_Unused_Var {
    Source_Location variable;
};

[[demo::severity(error)]]
[[demo::category(parse_error)]]
[[demo::message(token, "unexpected token")]]
struct Diag_Unexpected_Token {
    Source_Location token;
};
```

SFINAE & concepts (same thing)

templates & macros (same thing)

AST & regex (same thing)

```
struct Diag_Unused_Var {
  Source_Location variable;
};

struct Diag_Unexpected_Token {
  Source_Location token;
};

void report_impl(Diag_Unused_Var diag) override {
  std::println("{}: warning: unused variable",
               diag.variable);
}
void report_impl(Diag_Unexpected_Token diag) override
  std::println("{}: error: unexpected token",
               diag.token);
}
```

```
struct Diag_Unused_Var {
  Source_Location variable;
  string message() { return "unused variable"; }
};
struct Diag_Unexpected_Token {
  Source_Location token;
  string message() { return "unexpected token"; }
};
void report_impl(Diag_Unused_Var diag) override {
  std::println("{}: warning: {}",
               diag.variable, diag.message());
}
void report_impl(Diag_Unexpected_Token diag) override
  std::println("{}: error: {}",
               diag.token, diag.message());
}
```

```
struct Diag_Unused_Var {
  Source_Location variable;
  string message() { return "unused variable"; }
};



void report_impl(Diag_Unused_Var diag) override {
  std::println("{}: warning: {}",
               diag.variable, diag.message());
}
```

```
struct Diag_Unused_Var {
  Source_Location variable;
  string message() { return "unused variable"; }
  string severity() { return "warning"; }
};



void report_impl(Diag_Unused_Var diag) override {
  std::println("{}: {}: {}",
               diag.variable,
               diag.severity(), diag.message());
}
```

```
struct Diag_Unused_Var {
  Source_Location variable;
  string message() { return "unused variable"; }
  string severity() { return "warning"; }
};



void report_impl(Diag_Unused_Var diag) override {
  std::println("{}: {}: {}",
               diag.variable,
               diag.severity(), diag.message());
}
```

```
struct Diag_Unused_Var {
  Source_Location variable;
  string message() { return "unused variable"; }
  string severity() { return "warning"; }
  Source_Location location() { return variable; }
};


void report_impl(Diag_Unused_Var diag) override {
  std::println("{}: {}: {}",
               diag.location(),
               diag.severity(), diag.message());
}
```

```
void report_impl(Diag_Unused_Var diag) override {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
}
```

```
template <class Diag>
void report_impl(Diag_Unused_Var diag) override {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
}
void report_impl(Diag_Unexpected_Token diag) override
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message();
}
```

virtual functions can't be templates!

```
template <class Diag>
void report_impl(Diag diag) override {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
}
```

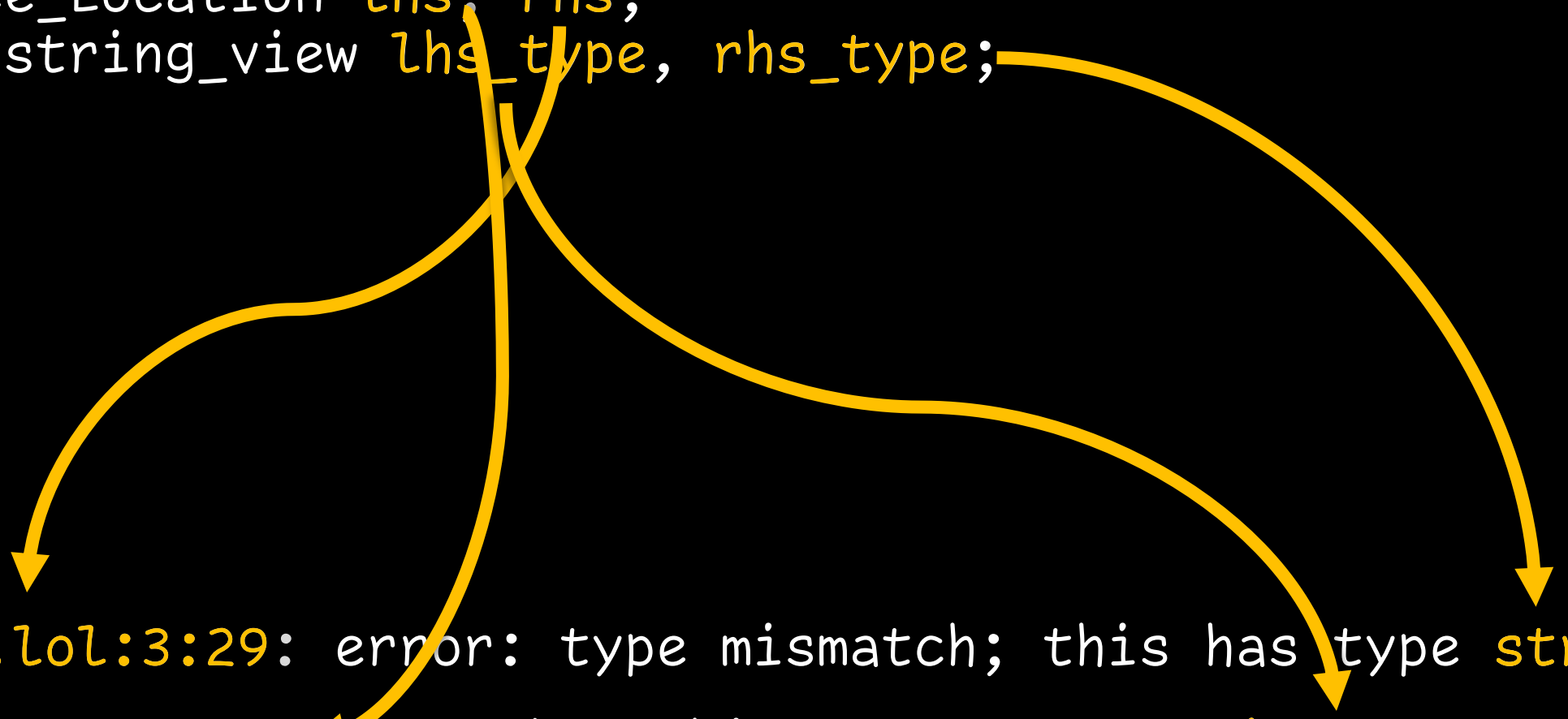
```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
}
void report_impl(Diag_Unused_Var diag) override {
  report_impl_impl(diag);
}
void report_impl(Diag_Unexpected_Token diag) override
  report_impl_impl(diag);
}
```

SFINAE & concepts (same thing)

templates & macros (same thing)

AST & regex (same thing)

```
struct Diag_Type_Mismatch {
  Source_Location lhs, rhs;
  std::string_view lhs_type, rhs_type;
};
```



```
example.lol:3:29: error: type mismatch; this has type string
example.lol:3:12: note: but this expects type integer
```

```
struct Diag_Type_Mismatch {
  Source_Location lhs, rhs;
  std::string_view lhs_type, rhs_type;
};

void report_impl(Diag_Type_Mismatch diag) override {
  std::println("{}: error: type mismatch; this has type {}",
               diag.rhs, diag.rhs_type);
  std::println("{}: note: but this expects type {}",
               diag.lhs, diag.lhs_type);
}
```

```
example.lol:3:29: error: type mismatch; this has type string
example.lol:3:12: note: but this expects type integer
```

```
struct Diag_Type_Mismatch {
  Source_Location lhs, rhs;
  std::string_view lhs_type, rhs_type;
};

void report_impl(Diag_Type_Mismatch diag) override {
  std::println("{}: error: type mismatch; this has type {}",
               diag.rhs, diag.rhs_type);
  std::println("{}: note: but this expects type {}",
               diag.lhs, diag.lhs_type);
}
```

```
struct Diag_Type_Mismatch {
  Source_Location lhs, rhs;
  std::string_view lhs_type, rhs_type;
};
```

```
void report_impl(Diag_Type_Mismatch diag) override {
  std::println("{}: error: type mismatch; this has type {}",
               diag.rhs, diag.rhs_type);
  std::println("{}: note: but this expects type {}",
               diag.lhs, diag.lhs_type);
}
```

```
struct Diag_Type_Mismatch {
  Source_Location lhs, rhs;
  std::string_view lhs_type, rhs_type;
};
```

```
void report_impl(Diag_Type_Mismatch diag) override {
  std::println("{}: error: type mismatch; this has type {}",
               diag.rhs, diag.rhs_type);
  std::println("{}: note: but this expects type {}",
               diag.lhs, diag.lhs_type);
}
```

```
struct Diag_Type_Mismatch {
  Source_Location lhs, rhs;
  std::string_view lhs_type, rhs_type;
  string message() {
    return format("type mismatch; this has type {}", rhs_type);
  }
  string severity() { return "error"; }
  Source_Location location() { return rhs; }
};
```

```
void report_impl(Diag_Type_Mismatch diag) override {
  std::println("{}: {}: {}",
               diag.location(), diag.severity(), diag.message())
  std::println("{}: note: but this expects type {}",
               diag.lhs, diag.lhs_type);
}
```

```
struct Diag_Type_Mismatch {
  Source_Location lhs, rhs;
  std::string_view lhs_type, rhs_type;
  string message() {
    return format("type mismatch; this has type {}", rhs_type);
  }
  string severity() { return "error"; }
  Source_Location location() { return rhs; }
};



void report_impl(Diag_Type_Mismatch diag) override {
  std::println("{}: {}: {}",
               diag.location(), diag.severity(), diag.message());
  std::println("{}: note: but this expects type {}",
               diag.lhs, diag.lhs_type);
}
```

```
struct Diag_Type_Mismatch {
  Source_Location lhs, rhs;
  std::string_view lhs_type, rhs_type;
  string message() {
    return format("type mismatch; this has type {}", rhs_type);
  }
  string severity() { return "error"; }
  Source_Location location() { return rhs; }
  string message_2() {
    return format("but this expects type {}", lhs_type);
  }
  Source_Location location_2() { return lhs; }
};
void report_impl(Diag_Type_Mismatch diag) override {
  std::println("{}: {}: {}",
               diag.location(), diag.severity(), diag.message());
  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());
}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());
}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());
}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());

}
```

```
  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  report_impl_impl_2(diag);
}


template <class Diag>
void report_impl_impl_2(Diag diag) {
  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());
}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  report_impl_impl_2(diag);
}

template <class Diag,
  class = decltype(declval<Diag>().location_2())>
void report_impl_impl_2(Diag diag) {
  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());
}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  report_impl_impl_2(diag);
}

template <class Diag,
  class = decltype(declval<Diag>().location_2())>
void report_impl_impl_2(Diag diag) {
  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());
}
template <class Diag>
void report_impl_impl_2(Diag) {}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  report_impl_impl_2(diag);
}

template <class Diag,
  class = decltype(declval<Diag>().location_2())>
void report_impl_impl_2(Diag diag) {
  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());
}
template <class Diag>
void report_impl_impl_2(Diag) const {}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());

  std::println("{}: note: {}",
               diag.location_2(), diag.message_2());

}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  if (__magic__) {
    std::println("{}: note: {}",
                 diag.location_2(), diag.message_2());
  }
}
```

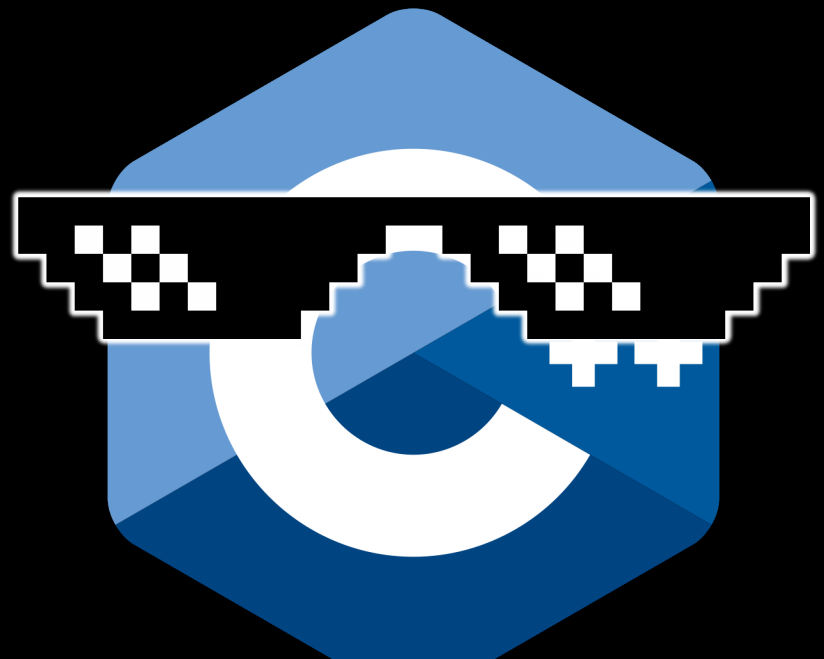
```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  if constexpr (__magic__) {
    std::println("{}: note: {}",
                 diag.location_2(), diag.message_2());
  }
}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  if constexpr (__magic__) {
    std::println("{}: note: {}",
                 diag.location_2(), diag.message_2());
  }
}

template <class Diag,
  class = decltype(declval<Diag>().location_2())>
constexpr bool has_location_2() { return true; }
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  if constexpr (__magic__) {
    std::println("{}: note: {}",
                 diag.location_2(), diag.message_2());
  }
}

template <class Diag,
  class = decltype(declval<Diag>().location_2())>
constexpr bool has_location_2() { return true; }
template <class Diag>
constexpr bool has_location_2() const {return false;}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  if constexpr (has_location_2<Diag>()) {
    std::println("{}: note: {}",
                 diag.location_2(), diag.message_2());
  }
}

template <class Diag,
  class = decltype(declval<Diag>().location_2())>
constexpr bool has_location_2() { return true; }
template <class Diag>
constexpr bool has_location_2() const {return false;}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity(), diag.message());
  if constexpr (requires { diag.location_2(); }) {
    std::println("{}: note: {}",
                 diag.location_2(), diag.message_2());
  }
}
```



~~SFINAE~~ & ~~concepts~~ (same thing)

~~templates~~ & macros (same thing)

AST & regex (same thing)

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", /* ... */);
  if constexpr (requires { diag.location_2(); }) {
    std::println("{}: note: {}", /* ... */));
  }
}
void report_impl(Diag_Unused_Var diag) override {
  report_impl_impl(diag);
}
void report_impl(Diag_Unexpected_Token diag) override
  report_impl_impl(diag);
}
void report_impl(Diag_Type_Mismatch diag) override {
  report_impl_impl(diag);
}
```

```
void report_impl(Diag_Unused_Var diag) override {
  report_impl_impl(diag);
}
void report_impl(Diag_Unexpected_Token diag) override
  report_impl_impl(diag);
}
void report_impl(Diag_Type_Mismatch diag) override {
  report_impl_impl(diag);
}
```

```
void report_impl(Diag_Unused_Var diag) override {
  report_impl_impl(diag);
}
void report_impl(Diag_Unexpected_Token diag) override
  report_impl_impl(diag);
}
void report_impl(Diag_Type_Mismatch diag) override {
  report_impl_impl(diag);
}
```

```
#define MAKE_REPORT_IMPL(Diag_Type)                    \
void report_impl(Diag_Type diag) override {            \
  report_impl_impl(diag);                              \
}

MAKE_REPORT_IMPL(Diag_Unused_Var)
MAKE_REPORT_IMPL(Diag_Unexpected_Token)
MAKE_REPORT_IMPL(Diag_Type_Mismatch)
```

```
struct Diagnostic_Reporter {
  void report(Diag_Unused_Var diag) {
    if (!disabled_categories.contains(
          Diagnostic_Category::dead_code))
      report_impl(diag);
  }
  void report(Diag_Unexpected_Token diag) {
    if (!disabled_categories.contains(
          Diagnostic_Category::parse_error))
      report_impl(diag);
  }
  virtual void report_impl(Diag_Unused_Var diag) = 0;
  virtual void report_impl(Diag_Unexpected_Token) = 0;
  std::set<Diagnostic_Category> disabled_categories;
};
```

```
struct Diagnostic_Reporter {
  void report(Diag_Unused_Var diag) {
    if (!disabled_categories.contains(
          diag.category))
      report_impl(diag);
  }
  void report(Diag_Unexpected_Token diag) {
    if (!disabled_categories.contains(
          diag.category))
      report_impl(diag);
  }
  virtual void report_impl(Diag_Unused_Var diag) = 0;
  virtual void report_impl(Diag_Unexpected_Token) = 0;
  std::set<Diagnostic_Category> disabled_categories;
};
```

```
struct Diagnostic_Reporter {
  template <class Diag>
  void report(Diag diag) {
    if (!disabled_categories.contains(
            diag.category))
      report_impl(diag);
  }



  virtual void report_impl(Diag_Unused_Var diag) = 0;
  virtual void report_impl(Diag_Unexpected_Token) = 0;
  std::set<Diagnostic_Category> disabled_categories;
};
```

```
struct Diagnostic_Reporter {
  template <class Diag>
  void report(Diag diag) {
    if (!disabled_categories.contains(
          diag.category))
      report_impl(diag);
  }


#define MAKE_REPORT_IMPL(Diag_Type)                    \
  virtual void report_impl(Diag_Type diag) = 0;
MAKE_REPORT_IMPL(Diag_Unused_Var)
MAKE_REPORT_IMPL(Diag_Unexpected_Token)
  std::set<Diagnostic_Category> disabled_categories;
};
```

```
#define MAKE_REPORT_IMPL(Diag_Type)                    \
  virtual void report_impl(Diag_Type diag) = 0;
MAKE_REPORT_IMPL(Diag_Unused_Var)
MAKE_REPORT_IMPL(Diag_Unexpected_Token)
MAKE_REPORT_IMPL(Diag_Type_Mismatch)
```

```
// Diag_Reporter (base class)
#define MAKE_REPORT_IMPL(Diag_Type)                    \
  virtual void report_impl(Diag_Type diag) = 0;
MAKE_REPORT_IMPL(Diag_Unused_Var)
MAKE_REPORT_IMPL(Diag_Unexpected_Token)
MAKE_REPORT_IMPL(Diag_Type_Mismatch)
#undef MAKE_REPORT_IMPL
// CLI_Reporter/JSON_Reporter
#define MAKE_REPORT_IMPL(Diag_Type)                    \
void report_impl(Diag_Type diag) override {            \
  report_impl_impl(diag);                              \
}
MAKE_REPORT_IMPL(Diag_Unused_Var)
MAKE_REPORT_IMPL(Diag_Unexpected_Token)
MAKE_REPORT_IMPL(Diag_Type_Mismatch)
#undef MAKE_REPORT_IMPL
```

```
// Diag_Reporter (base class)
#define MAKE_REPORT_IMPL(Diag_Type)                    \
  virtual void report_impl(Diag_Type diag) = 0;
MAKE_REPORT_IMPL(Diag_Unused_Var)
MAKE_REPORT_IMPL(Diag_Unexpected_Token)
MAKE_REPORT_IMPL(Diag_Type_Mismatch)
#undef MAKE_REPORT_IMPL
// CLI_Reporter/JSON_Reporter
#define MAKE_REPORT_IMPL(Diag_Type)
void report_impl(Diag_Type diag) overr
  report_impl_impl(diag);
}
MAKE_REPORT_IMPL(Diag_Unused_Var)
MAKE_REPORT_IMPL(Diag_Unexpected_Token)
MAKE_REPORT_IMPL(Diag_Type_Mismatch)
#undef MAKE_REPORT_IMPL
```

```
// Diag_Reporter (base class)
#define MAKE_REPORT_IMPL(Diag_Type)                    \
  virtual void report_impl(Diag_Type diag) = 0;

DIAGNOSTIC_TYPES_X

#undef MAKE_REPORT_IMPL
// CLI_Reporter/JSON_Reporter
#define MAKE_REPORT_IMPL(Diag_Type)                    \
void report_impl(Diag_Type diag) override {            \
  report_impl_impl(diag);                              \
}

DIAGNOSTIC_TYPES_X

#undef MAKE_REPORT_IMPL
```

```
#define DIAGNOSTIC_TYPES_X                                  \
  MAKE_REPORT_IMPL(Diag_Unused_Var)                         \
  MAKE_REPORT_IMPL(Diag_Unexpected_Token)                   \
  MAKE_REPORT_IMPL(Diag_Type_Mismatch)
// Diag_Reporter (base class)
#define MAKE_REPORT_IMPL(Diag_Type)                         \
  virtual void report_impl(Diag_Type diag) = 0;
DIAGNOSTIC_TYPES_X
#undef MAKE_REPORT_IMPL
// CLI_Reporter/JSON_Reporter
#define MAKE_REPORT_IMPL(Diag_Type)                         \
void report_impl(Diag_Type diag) override {                 \
  report_impl_impl(diag);                                   \
}
DIAGNOSTIC_TYPES_X
#undef MAKE_REPORT_IMPL
```

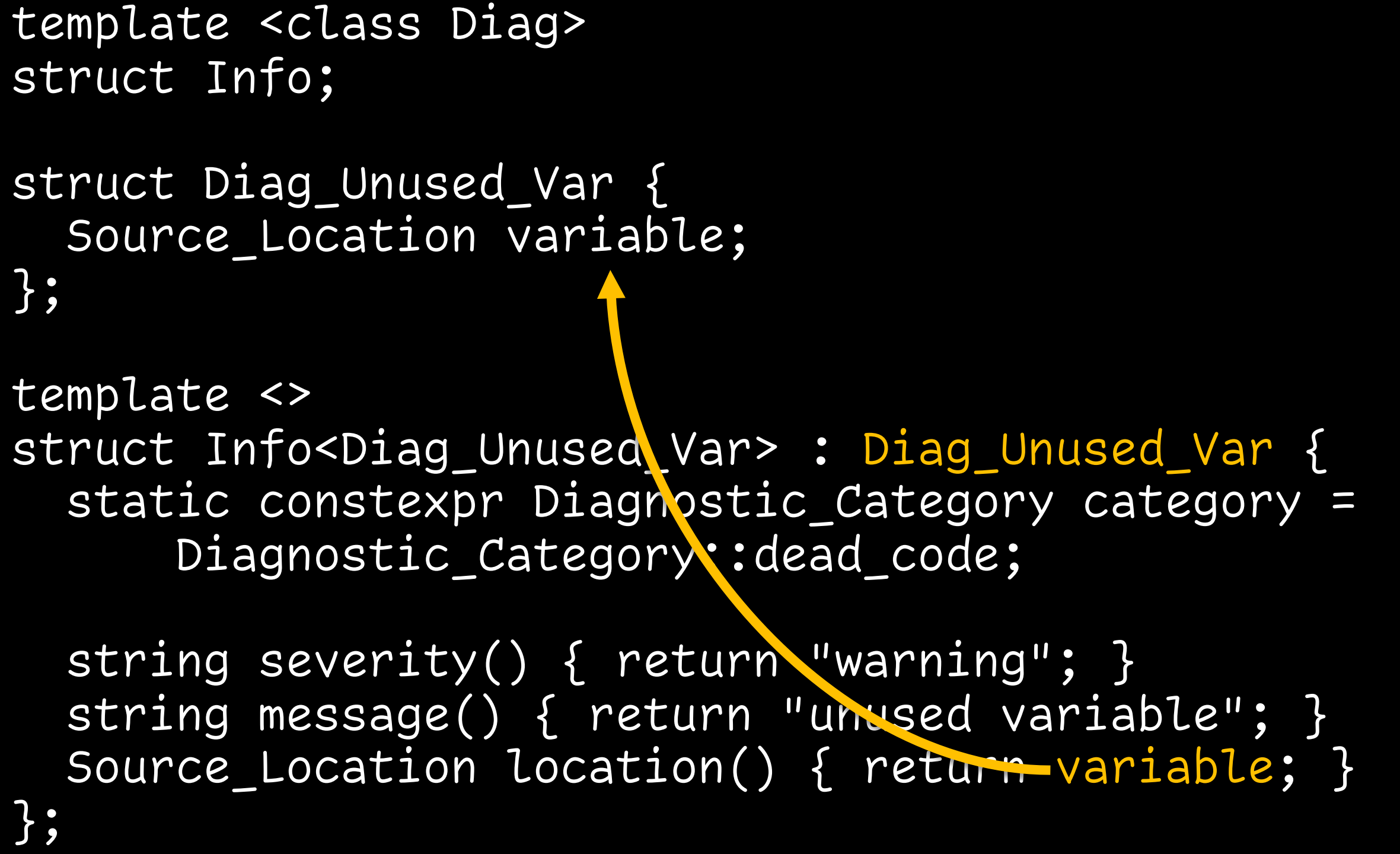
~~SFINAE~~ & ~~concepts~~ (same thing)

~~templates~~ & macros (same thing)

AST & regex (same thing)

```
struct Diag_Unused_Var {
  Source_Location variable;

  static constexpr Diagnostic_Category category =
      Diagnostic_Category::dead_code;

  string severity() { return "warning"; }
  string message() { return "unused variable"; }
  Source_Location location() { return variable; }
};
```

```
template <class Diag>
struct Info;

struct Diag_Unused_Var {
  Source_Location variable;
};

template <>
struct Info<Diag_Unused_Var> : Diag_Unused_Var {
  static constexpr Diagnostic_Category category =
      Diagnostic_Category::dead_code;

  string severity() { return "warning"; }
  string message() { return "unused variable"; }
  Source_Location location() { return variable; }
};
```


template <class Diag>
struct Info;
struct Diag_Unused_Var {
Source_Location variable;
};
template <>
struct Info<Diag_Unused_Var> : Diag_Unused_Var {
static constexpr Diagnostic_Category category =
Diagnostic_Category::dead_code;
string severity() { return "warning"; }
string message() { return "unused variable"; }
Source_Location location() { return variable; }
};

```
template <class Diag>
void report_impl_impl(Diag diag) {
  std::println("{}: {}: {}", diag.location(),
               diag.severity, diag.message());
  if constexpr (requires { diag.location_2(); }) {
    std::println("{}: note: {}", diag.location_2(),
                 diag.message_2());
  }
}
```

```
template <class Diag>
void report_impl_impl(Diag diag) {
  // Undefined behavior! (don't tell anyone 🤫)
  const auto& info = (const Info<Diag>&)diag;
  std::println("{}: {}: {}", info.location(),
               info.severity, info.message());
  if constexpr (requires { info.location_2(); }) {
    std::println("{}: note: {}", info.location_2(),
                 info.message_2());
  }
}
```

```
template <>
struct Info<Diag_Unused_Var> : Diag_Unused_Var {
  static constexpr Diagnostic_Category category =
      Diagnostic_Category::dead_code;
  string severity() { return "warning"; }
  string message() { return "unused variable"; }
  Source_Location location() { return variable; }
};
```

```
string message() { return "unused variable"; }
Source_Location location() { return variable; }
```

```
#define DIAGNOSTIC_MESSAGE(loc, msg)                    \
  string message() { return msg; }                      \
  Source_Location location() { loc; }


  DIAGNOSTIC_MESSAGE(variable, "unused variable")
```

```
#define DIAGNOSTIC_MESSAGE(loc, msg)                    \
  string message() { return msg; }                      \
  Source_Location location() { loc; }


  DIAGNOSTIC_MESSAGE(variable, "unused variable")


  string message() {
    return format("this has type {}", rhs_type);
  }
  Source_Location location() { return rhs; }
```

```
#define DIAGNOSTIC_MESSAGE(loc, ...)                    \
  string message() { return format(__VA_ARGS__); } \
  Source_Location location() { loc; }


  DIAGNOSTIC_MESSAGE(variable, "unused variable")


  string message() {
    return format("this has type {}", rhs_type);
  }
  Source_Location location() { return rhs; }
```

```
#define DIAGNOSTIC_MESSAGE(loc, ...)                    \
  string message() { return format(__VA_ARGS__); } \
  Source_Location location() { loc; }


  DIAGNOSTIC_MESSAGE(variable, "unused variable")


  DIAGNOSTIC_MESSAGE(
    rhs, "this has type {}", rhs_type)
```

```
#define DIAGNOSTIC_MESSAGE(loc, ...)                    \
  string message() { return format(__VA_ARGS__); } \
  Source_Location location() { loc; }

  DIAGNOSTIC_MESSAGE(
    rhs, "this has type {}", rhs_type)
```

```
#define DIAGNOSTIC_MESSAGE(loc, ...)                    \
  string message() { return format(__VA_ARGS__); } \
  Source_Location location() { loc; }

  DIAGNOSTIC_MESSAGE(
    rhs, "this has type {}", rhs_type)

  string message_2() {
    return format("but this expects type {}", lhs_type
  }
  Source_Location location_2() { return lhs; }
```

```
#define DIAGNOSTIC_MESSAGE(loc, ...)                    \
  string message() { return format(__VA_ARGS__); } \
  Source_Location location() { loc; }

  DIAGNOSTIC_MESSAGE(
    rhs, "this has type {}", rhs_type)

#define DIAGNOSTIC_MESSAGE_2(loc, ...)                  \
  string message_2() { return format(__VA_ARGS__); } \
  Source_Location location_2() { loc; }

  DIAGNOSTIC_MESSAGE_2(
    lhs, "but this expects type {}", lhs_type)
```

```
struct Diag_Unused_Var {
  Source_Location variable;
};
template <>
struct Info<Diag_Unused_Var> : Diag_Unused_Var {
  static constexpr Diagnostic_Category category =
      Diagnostic_Category::dead_code;
  string severity() { return "warning"; }
  DIAGNOSTIC_MESSAGE(variable, "unused variable")
};
```

```
#define MAKE_DIAGNOSTIC()                                    \
struct Diag_Unused_Var {                                     \
  Source_Location variable;                                  \
};                                                           \
template <>                                                  \
struct Info<Diag_Unused_Var> : Diag_Unused_Var {             \
  static constexpr Diagnostic_Category category =            \
      Diagnostic_Category::dead_code;                        \
  string severity() { return "warning"; }                    \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")            \
};

MAKE_DIAGNOSTIC()
```

```
#define MAKE_DIAGNOSTIC()                                    \
struct Diag_Unused_Var {                                     \
  Source_Location variable;                                  \
};                                                           \
template <>                                                  \
struct Info<Diag_Unused_Var> : Diag_Unused_Var {             \
  static constexpr Diagnostic_Category category =            \
      Diagnostic_Category::dead_code;                        \
  string severity() { return "warning"; }                    \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")            \
};
              MAKE_DIAGNOSTIC()
```

```
#define MAKE_DIAGNOSTIC(Type)                                  \
struct Type {                                                  \
  Source_Location variable;                                    \
};                                                             \
template <>                                                    \
struct Info<Type> : Type {                                     \
  static constexpr Diagnostic_Category category =              \
      Diagnostic_Category::dead_code;                          \
  string severity() { return "warning"; }                      \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")              \
};

              MAKE_DIAGNOSTIC(
                Diag_Unused_Var)
```

```
#define MAKE_DIAGNOSTIC(Type)                                  \
struct Type {                                                  \
  Source_Location variable;                                    \
};                                                             \
template <>                                                    \
struct Info<Type> : Type {                                     \
  static constexpr Diagnostic_Category category =              \
      Diagnostic_Category::dead_code;                          \
  string severity() { return "warning"; }                      \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")              \
};
              MAKE_DIAGNOSTIC(
                Diag_Unused_Var)
```

```
#define MAKE_DIAGNOSTIC(Type, body)                                    \
struct Type body                                                       \
                                                                       \
                                                                       \
                                                                       \
template <>                                                            \
struct Info<Type> : Type {                                             \
  static constexpr Diagnostic_Category category =                      \
      Diagnostic_Category::dead_code;                                  \
  string severity() { return "warning"; }                              \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")                      \
};
              MAKE_DIAGNOSTIC(
                Diag_Unused_Var, {
                  Source_Location variable;
                })
```

```
#define MAKE_DIAGNOSTIC(Type, body)                                    \
struct Type body                                                       \
                                                                       \
                                                                       \
                                                                       \
template <>                                                            \
struct Info<Type> : Type {                                             \
  static constexpr Diagnostic_Category category =                      \
      Diagnostic_Category::dead_code;                                  \
  string severity() { return "warning"; }                              \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")                      \
};
              MAKE_DIAGNOSTIC(
                Diag_Unused_Var, {
                  Source_Location variable;
                })
```

```
#define MAKE_DIAGNOSTIC(Type, body, sev)                                \
struct Type body                                                        \
                                                                        \
                                                                        \
                                                                        \
template <>                                                             \
struct Info<Type> : Type {                                              \
  static constexpr Diagnostic_Category category =                       \
      Diagnostic_Category::dead_code;                                   \
  string severity() { return #sev; }                                    \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")                       \
};
              MAKE_DIAGNOSTIC(
                Diag_Unused_Var, {
                  Source_Location variable;
                },
                warning)
```

```
#define MAKE_DIAGNOSTIC(Type, body, sev)                                \
struct Type body                                                        \
                                                                        \
                                                                        \
template <>                                                             \
struct Info<Type> : Type {                                              \
  static constexpr Diagnostic_Category category =                       \
      Diagnostic_Category::dead_code;                                   \
  string severity() { return #sev; }                                    \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")                       \
};
              MAKE_DIAGNOSTIC(
                Diag_Unused_Var, {
                  Source_Location variable;
                },
                warning)
```

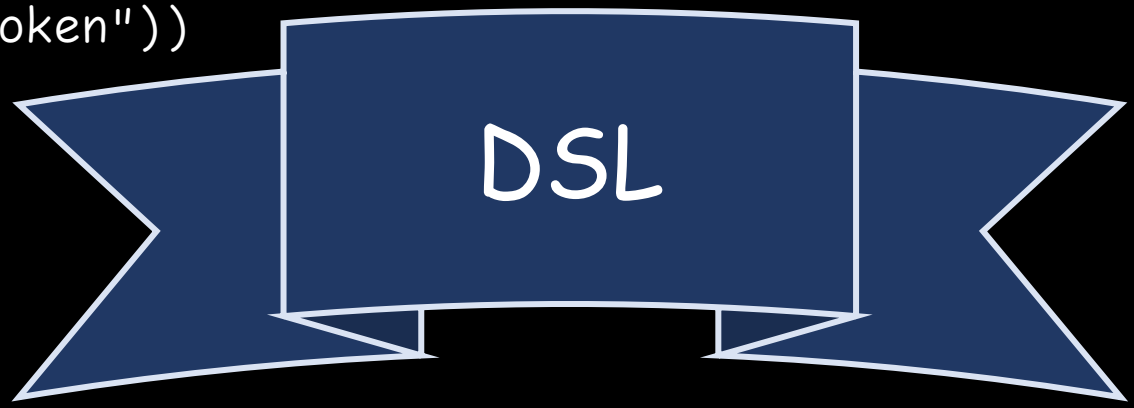
```
#define MAKE_DIAGNOSTIC(Type, body, sev, cat)                          \
struct Type body                                                       \
                                                                       \
                                                                       \
                                                                       \
template <>                                                            \
struct Info<Type> : Type {                                             \
  static constexpr Diagnostic_Category category =                      \
      Diagnostic_Category::cat;                                        \
  string severity() { return #sev; }                                   \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")                      \
};
              MAKE_DIAGNOSTIC(
                Diag_Unused_Var, {
                  Source_Location variable;
                },
                warning,
                dead_code)
```

```
#define MAKE_DIAGNOSTIC(Type, body, sev, cat)                          \
struct Type body                                                       \
                                                                       \
                                                                       \
                                                                       \
template <>                                                            \
struct Info<Type> : Type {                                             \
  static constexpr Diagnostic_Category category =                      \
      Diagnostic_Category::cat;                                        \
  string severity() { return #sev; }                                   \
  DIAGNOSTIC_MESSAGE(variable, "unused variable")                      \
};
              MAKE_DIAGNOSTIC(
                Diag_Unused_Var, {
                  Source_Location variable;
                },
                warning,
                dead_code)
```

```
#define MAKE_DIAGNOSTIC(Type, body, sev, cat, msgs)                \
struct Type body                                                   \
                                                                   \
                                                                   \
                                                                   \
template <>                                                        \
struct Info<Type> : Type {                                         \
  static constexpr Diagnostic_Category category =                  \
      Diagnostic_Category::cat;                                    \
  string severity() { return #sev; }                               \
  msgs                                                             \
};
              MAKE_DIAGNOSTIC(
                Diag_Unused_Var, {
                  Source_Location variable;
                },
                warning,
                dead_code,
                DIAGNOSTIC_MESSAGE(variable, "unused variable"))
```

```
MAKE_DIAGNOSTIC(
  Diag_Unused_Var, {
    Source_Location variable;
  },
  warning,
  dead_code,
  DIAGNOSTIC_MESSAGE(variable, "unused variable"))
```

```
MAKE_DIAGNOSTIC(
  Diag_Unused_Var, {
    Source_Location variable;
  },
  warning,
  dead_code,
  DIAGNOSTIC_MESSAGE(variable, "unused variable"))

MAKE_DIAGNOSTIC(
  Diag_Unexpected_Token, {
    Source_Location token;
  },
  error,
  parse_error,
  DIAGNOSTIC_MESSAGE(token, "unexpected token"))

MAKE_DIAGNOSTIC(
  Diag_Type_Mismatch, {
    Source_Location lhs, rhs;
    std::string_view lhs_type, rhs_type;
  },
  error,
  type_error,
  DIAGNOSTIC_MESSAGE(rhs, "type mismatch; this has type {}", rhs_type)
  DIAGNOSTIC_MESSAGE_2(lhs, "but this expects type {}", lhs_type))
```



```
MAKE_DIAGNOSTIC(
  Diag_Unused_Var, {
    Source_Location variable;
  },
  warning,
  dead_code,
  DIAGNOSTIC_MESSAGE(variable, "unused variable"))

MAKE_DIAGNOSTIC(
  Diag_Unexpected_Token, {
    Source_Location token;
  },
  error,
  parse_error,
  DIAGNOSTIC_MESSAGE(token, "unexpected token"))

MAKE_DIAGNOSTIC(
  Diag_Type_Mismatch, {
    Source_Location lhs, rhs;
    std::string_view lhs_type, rhs_type;
  },
  error,
  type_error,
  DIAGNOSTIC_MESSAGE(rhs, "type mismatch; this has type {}", rhs_type)
  DIAGNOSTIC_MESSAGE_2(lhs, "but this expects type {}", lhs_type))
```

```
#define DIAGNOSTIC_TYPES_X                    \
  MAKE_REPORT_IMPL(Diag_Unused_Var)           \
  MAKE_REPORT_IMPL(Diag_Unexpected_Token)     \
  MAKE_REPORT_IMPL(Diag_Type_Mismatch)
```

```
MAKE_DIAGNOSTIC(
  Diag_Unused_Var, {
    Source_Location variable;
  },
  warning,
  dead_code,
  DIAGNOSTIC_MESSAGE(variable, "unused variable"))

MAKE_DIAGNOSTIC(
  Diag_Unexpected_Token, {
    Source_Location token;
  },
  error,
  parse_error,
  DIAGNOSTIC_MESSAGE(token, "unexpected token"))

MAKE_DIAGNOSTIC(
  Diag_Type_Mismatch, {
    Source_Location lhs, rhs;
    std::string_view lhs_type, rhs_type;
  },
  error,
  type_error,
  DIAGNOSTIC_MESSAGE(rhs, "type mismatch; this has type {}", rhs_type)
  DIAGNOSTIC_MESSAGE_2(lhs, "but this expects type {}", lhs_type))
```

```
#define DIAGNOSTIC_TYPES_X                    \
  MAKE_REPORT_IMPL(Diag_Unused_Var)           \
  MAKE_REPORT_IMPL(Diag_Unexpected_Token)     \
  MAKE_REPORT_IMPL(Diag_Type_Mismatch)
```

```
#define DIAGNOSTIC_TYPES_X

  DO_DIAGNOSTIC(
    Diag_Unused_Var, {
      Source_Location variable;
    },
    warning,
    dead_code,
    DIAGNOSTIC_MESSAGE(variable, "unused variable"))

  DO_DIAGNOSTIC(
    Diag_Unexpected_Token, {
      Source_Location token;
    },
    error,
    parse_error,
    DIAGNOSTIC_MESSAGE(token, "unexpected token"))

  DO_DIAGNOSTIC(
    Diag_Type_Mismatch, {
      Source_Location lhs, rhs;
      std::string_view lhs_type, rhs_type;
    },
    error,
    type_error,
    DIAGNOSTIC_MESSAGE(rhs, "type mismatch; this has type {}", rhs_type)
    DIAGNOSTIC_MESSAGE_2(lhs, "but this expects type {}", lhs_type))
```

```
#define MAKE_REPORT_IMPL(Diag_Type)                    \
void report_impl(Diag_Type diag) override {            \
  report_impl_impl(diag);                              \
}
DIAGNOSTIC_TYPES_X
#undef MAKE_REPORT_IMPL

#define MAKE_REPORT_IMPL(Diag_Type)                    \
  virtual void report_impl(Diag_Type diag) = 0;
DIAGNOSTIC_TYPES_X
#undef MAKE_REPORT_IMPL
```

```
#define DO_DIAGNOSTIC(Diag_Type, ...)                    \
void report_impl(Diag_Type diag) override {              \
  report_impl_impl(diag);                                \
}
DIAGNOSTIC_TYPES_X
#undef DO_DIAGNOSTIC

#define DO_DIAGNOSTIC(Diag_Type, ...)                    \
  virtual void report_impl(Diag_Type diag) = 0;
DIAGNOSTIC_TYPES_X
#undef DO_DIAGNOSTIC
```

~~SFINAE~~ & ~~concepts~~ (same thing)

~~templates~~ & ~~macros~~ (same thing)

AST & regex (same thing)

```
[[demo::severity(warning)]]
[[demo::category(dead_code_warning)]]
[[demo::message(variable, "unused variable")]]
struct Diag_Unused_Variable {
  Source_Location variable;
};

[[demo::severity(error)]]
[[demo::category(parse_error)]]
[[demo::message(token, "unexpected token")]]
struct Diag_Unexpected_Token {
  Source_Location token;
};
```

```
struct
  [[demo::severity(warning)]]
  [[demo::category(dead_code_warning)]]
  [[demo::message(variable, "unused variable")]]
    Diag_Unused_Variable {
  Source_Location variable;
};

struct
  [[demo::severity(error)]]
  [[demo::category(parse_error)]]
  [[demo::message(token, "unexpected token")]]
    Diag_Unexpected_Token {
  Source_Location token;
};
```

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]]
  [[demo::category(dead_code_warning)]]
  [[demo::message(variable, "unused variable")]]
  Source_Location variable;
};

struct Diag_Unexpected_Token {
  [[demo::severity(error)]]
  [[demo::category(parse_error)]]
  [[demo::message(token, "unexpected token")]]
  Source_Location token;
};
```

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]]
    [[demo::category(dead_code_warning)]]
      [[demo::message(variable, "unused variable")]]
};

struct Diag_Unexpected_Token {
  [[demo::severity(error)]]
    [[demo::category(parse_error)]]
      [[demo::message(token, "unexpected token")]] S
};
```

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]]
  [[demo::category(dead_code_warning)]]
  [[demo::message(variable, "unused variable")]]
  Source_Location variable;
};

struct Diag_Unexpected_Token {
  [[demo::severity(error)]]
  [[demo::category(parse_error)]]
  [[demo::message(token, "unexpected token")]]
  Source_Location token;
};
```

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]];
  [[demo::category(dead_code_warning)]];
  [[demo::message(variable, "unused variable")]];
  Source_Location variable;
};

struct Diag_Unexpected_Token {
  [[demo::severity(error)]];
  [[demo::category(parse_error)]];
  [[demo::message(token, "unexpected token")]];
  Source_Location token;
};
```

```
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wmissing-declarations"
#pragma clang diagnostic ignored "-Wunknown-attributes"

struct Diag_Unused_Variable {
  [[demo::severity(warning)]];
  [[demo::category(dead_code_warning)]];
  [[demo::message(variable, "unused variable")]];
  Source_Location variable;
};

struct Diag_Unexpected_Token {
  [[demo::severity(error)]];
  [[demo::category(parse_error)]];
  [[demo::message(token, "unexpected token")]];
  Source_Location token;
};

#pragma clang diagnostic pop
```


LibClang

LibClang does not expose attributes

LibClang's cursors API

LibClang's cursors API

inside each statement

butes

5. generate C++ source code
(Info<> template specializations)

Do not use LibClang when you…:

- want full control over the Clang AST

LibClang documentation

libclang features are implemented on-demand. Need to get more information about XXX for your project? Implement that yourself.

Jonathan “foonathan” Müller

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]];
  [[demo::category(dead_code_warning)]];
  [[demo::message(variable, "unused variable")]];
  Source_Location variable;
};

struct Diag_Unexpected_Token {
  [[demo::severity(error)]];
  [[demo::category(parse_error)]];
  [[demo::message(token, "unexpected token")]];
  Source_Location token;
};
```

```
struct Diag_Unused_Variable {
  [ [ demo :: severity ( warning ) ] ] ;
  [ [ demo :: category ( dead_code_warning ) ] ] ;
  [ [ demo :: message ( variable, "..." ) ] ] ;
  Source_Location variable ;
}
```

```
struct Diag_Unexpected_Token {
  [ [ demo :: severity ( error ) ] ] ;
  [ [ demo :: category ( parse_error ) ] ] ;
  [ [ demo :: message ( token, "..." ) ] ] ;
  Source_Location token ;
} ;
```

LibClang problems

- manually parsing C++ tokens
- installing LibClang
  - did you install the right version?
- using LibClang in your build system
- using your code generator in your build system
  - what about cross compilation?

~~SFINAE~~ & ~~concepts~~ (same thing)

~~templates~~ & ~~macros~~ (same thing)

AST & regex (same thing)

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]];
  [[demo::category(dead_code_warning)]];
  [[demo::message(variable, "unused variable")]];
  Source_Location variable;
};
```

```
struct Diag_Unused_Variable {
   [[demo::severity(warning)]];
   [[demo::category(dead_code_warning)]];
   [[demo::message(variable, "unused variable")]];
   Source_Location variable;
};
```

```
\[\[demo::severity\(\s*(\w+)\s*\)\]\]
```

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]];
  [[demo::category(dead_code_warning)]];
  [[demo::message(variable, "unused variable")]];
  Source_Location variable;
};
```

```
\[\[demo::severity\(\s*(\w+)\s*\)\]\]

\[\[demo::category\(\s*(\w+)\s*\)\]\]
```

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]];
  [[demo::category(dead_code_warning)]];
  [[demo::message(variable, "unused variable")]];
  Source_Location variable;
};
```

```
\[\[demo::severity\(\s*(\w+)\s*\)\]\]

\[\[demo::category\(\s*(\w+)\s*\)\]\]

\[\[demo::message\(\s*(\w+)\s*,\s*([^]*?)\s*\)\]\]
```

```
struct Diag_Unused_Variable {
  [[demo::severity(warning)]];
  [[demo::category(dead_code_warning)]];
  [[demo::message(variable, "unused variable")]];
  Source_Location variable;
};
```

```
\bstruct (Diag_\w+)

  \[\[demo::severity\(\s*(\w+)\s*\)\]\]

  \[\[demo::category\(\s*(\w+)\s*\)\]\]

\[\[demo::message\(\s*(\w+)\s*,\s*([^]*?)\s*\)\]\]
```

~~SFINAE~~ & ~~concepts~~ (same thing)

~~templates~~ & ~~macros~~ (same thing)

~~AST~~ & regex (same thing)

reflection & code gen

- struct metadata
  - this talk
  - Qt Meta-Object Compiler
- pretty-printing structs and enums
  - Rust #[derive(Debug)]
- [de]serialization (JSON, binary, …)
- language bindings (Python, Rust, …)
- documentation (--help)