Olivia Quinet

olivia.quinet@cluepoints.com

📍 MeetingCPP, Nov 2025

AGENDA

- Motivation
- Definitions
- Naive Approach
- Type-Erasure
- Efficient Evaluation
- Lessons Learned
- Q&A

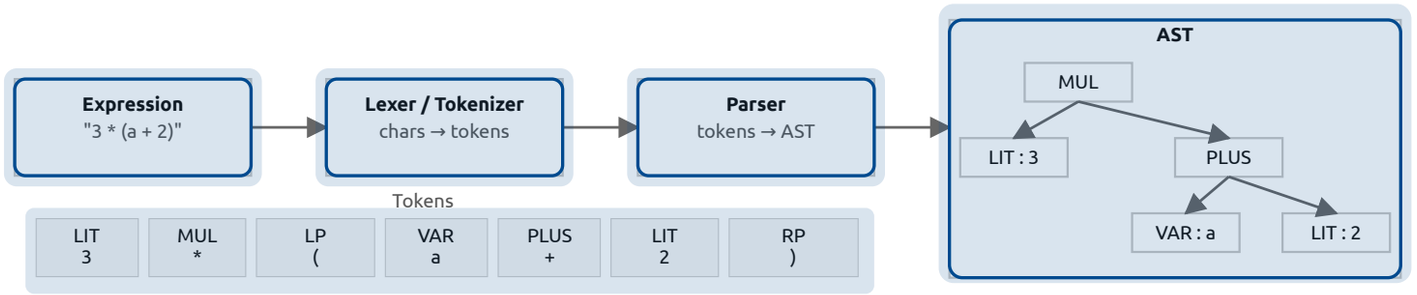
MOTIVATIONS

- Processing clinical data stored in tabular format
- Data Transformation : create new columns
- Data Extraction
- Split datasets (filtering)

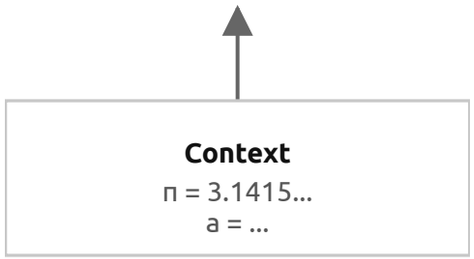
MOTIVATIONS : SUMMARY

- Evaluate user-defined expressions on clinical datasets
- Efficiency : potentially huge datasets (> millions of rows)
- Flexibility : support different data types and functions
- Error proof
- Available expression evaluators are (mostly) focused on mathematical expressions : ExprTk, ExprEval, ...

DEFINITIONS

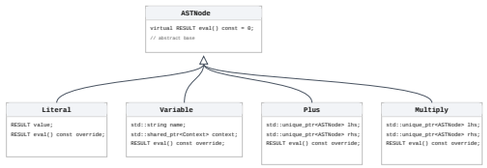


- Expression
- Lexer / Tokenizer
- Parser
- Abstract Syntax Tree
- Context containing the variables and default constants



A NAIVE APPROACH

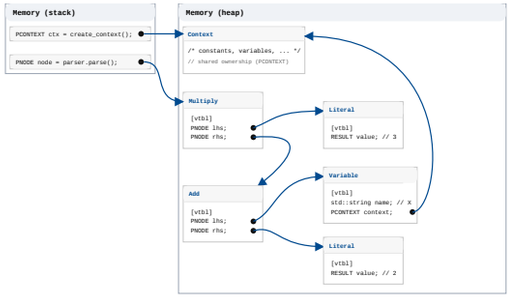
- **OO** : Each node of the AST tree is an object from a class hierarchy



```
using PNODE = std::unique_ptr<ASTNode>;
using PCONTEXT = std::shared_ptr<Context>;
using RESULT = std::variant<NullValue, Error, bool, long, double, ... >;

PCONTEXT pcontext = create_context(...);
auto parser = Parser("3*(a+2)", pcontext);
PNODE node = parser.parse();

pcontext->set_variable_value("a", 12);
RESULT result = node.eval();

std::cout << "The answer to everything is " << std::get<long>(result)
          << std::endl;
```

```
1  struct Plus : ASTNode {
2    PNODE lhs;
3    PNODE rhs;
4
5    explicit Plus(PNODE &&_lhs, PNODE &&_rhs)
6        : lhs(std::move(_lhs)), rhs(std::move(_rhs)) {}
7
8    RESULT eval() const override {
9      auto l = lhs->eval();
10     auto r = rhs->eval();
11     return std::visit(COMPUTE{}, l, r);
12   }
13
14   struct COMPUTE {
15     RESULT operator()(const long l, const long r) const noexcept {
```

3 * (X + 2)


Memory (stack)
PCONTEXT ctx = create_context();
PNODE node = parser.parse();
Memory (heap)
Context
/* constants, variables, ... */
// shared ownership (PCONTEXT)
Multiply
[vtbl]
PNODE lhs;
PNODE rhs;
Literal
[vtbl]
RESULT value; // 3
Variable
[vtbl]
std::string name; // x
PCONTEXT context;
Add
[vtbl]
PNODE lhs;
PNODE rhs;
Literal
[vtbl]
RESULT value; // 2

Cost of a call to `eval()`

- Access to the object through a smart pointer
- Access to the `eval()` via the vtbl
- Call to `std::visit`
- Create a `std::variant`

BENCHMARKS

Approach	Time (ns)	Factor
c++ expression	3.2	1.0
OO	22.6	7.1
ExprTk	3.3	1.0

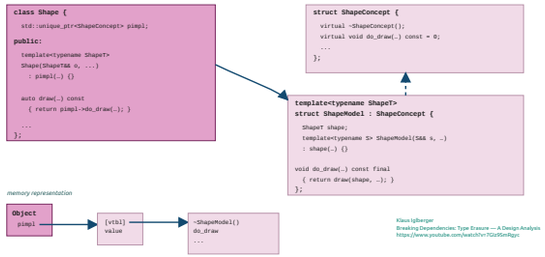
Expression `" 3 * (x + 2) "` *evaluated* 10^4 *times, 100 samples*

TYPE ERASURE DESIGN PATTERN

Provide a value-semantic interface (fixed set of methods) to manipulate unrelated types (open set of types).

```
auto shape1 = Shape(Circle(2));
auto shape2 = Shape(Square(3));
auto shape3 = Shape(Rectangle(4, 5));

shape1.draw(...);
shape2.draw(...);
shape3.draw(...);
```

TYPE-ERASURE DESIGN PATTERN : PIMPL-CONCEPT-MODEL



Klaus Iglberger
Breaking Dependencies: Type Erasure — A Design Analysis
https://www.youtube.com/watch?v=7GUr5SmRgyc

PROS:

- Polymorphism without inheritance, e.g. “duck typing”
- Architectural boundary / Decoupling & Dependency Inversion
- Customization & Extensibility
- **Break dependencies and design modular systems**

CONS:

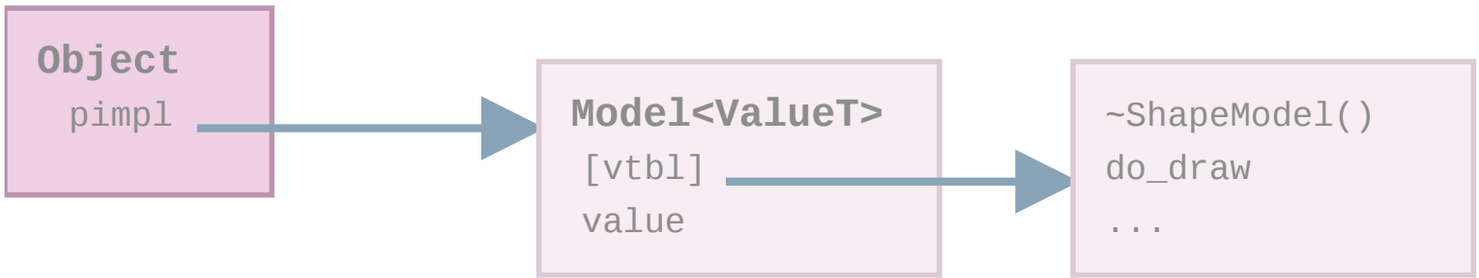
- Runtime Overhead
- Loss of Static Type Information
- Boilerplate / increased complexity
- Dynamic Allocation

CPP-TE

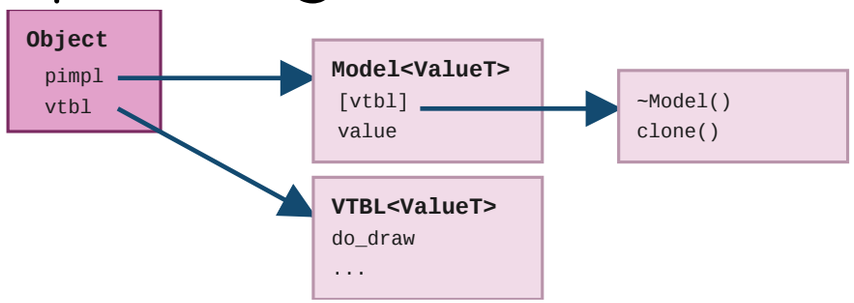
C++20 framework for Type-Erasure design pattern

https://github.com/olivia76/cpp-expr

- Default pimpl/concept/model design



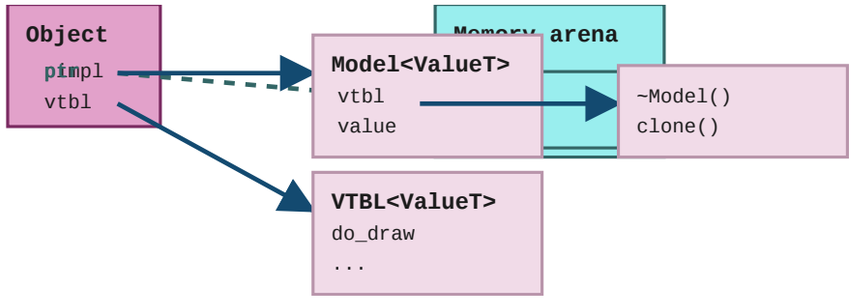
- VTBL/pimpl design



```cpp
namespace te2 {
template <typename VTBL_, typename VisitorStrategy,
          typename PimplStrategy = te2::pimpl::unique_ptr_strategy>
class base {
private:
  using PIMPL = typename PimplStrategy::PIMPL;
  using VTBL = std::decay_t<VTBL_>;
  using PVTBL = const VTBL *;
  using CAN_CLONE = typename PimplStrategy::can_clone;

  PIMPL m_pimpl;
  PVTBL m_pvtbl;
  [[no_unique_address]] VisitorStrategy m_visitor_strategy;

protected:
  template <typename ValueT, typename... Args>
  requires(!std::is_base_of<base, std::decay_t<ValueT>>::value)
  explicit base(ValueT &&tp, Args &&...args)
    : m_pimpl(PimplStrategy::make_pimpl(std::forward<ValueT>(tp),
                                        std::forward<Args>(args)...)),
      m_pvtbl(create_vtbl<ValueT>()) {}

private:
  template <typename ValueT> static PVTBL create_vtbl() {
```

```cpp
struct shape_vtbl {
  double (*do_area)(const void *);
  double (*do_perimeter)(const void *);
  ...
  template <typename CastT>
  static constexpr shape_vtbl create() {
    return {
        .do_area =
            [](const void *p) { return area(CastT::value(p)); },
        .do_perimeter =
            [](const void *p) { return perimeter(CastT::value(p)); },
        ...
    };
  }
};

using BASE = te2::base<shape_vtbl>;
struct Shape : public BASE {
  template <typename... Args>
  Shape(Args &&...args) : BASE(std::forward<Args>(args)...) {}

  auto area() const { return vtbl().do_area(pimpl()); }
  auto perimeter() const { return vtbl().do_perimeter(pimpl()); }
};
```

TYPE ERASURE FOR NODE WITH SMALL OBJECT OPTIMIZATION



LET'S GO BACK TO THE FRAMEWORK...

```
1 namespace te2 {
2 template <typename VTBL_, typename VisitorStrategy,
3           typename PimplStrategy = te2::pimpl::unique_ptr_strategy>
4 class base {
5 private:
6   using PIMPL = typename PimplStrategy::PIMPL;
7   ...
8 };
9 }
```

… AND IN THE CREATION OF THE VTBL

```
1  struct shape_vtbl {
2    double (*do_area)(const void *);
3    double (*do_perimeter)(const void *);
4    ...
5    template <typename CastT>
6    static constexpr shape_vtbl create() {
7      return {
8          .do_area =
9              [](const void *p) { return area(CastT::value(p)); },
10         .do_perimeter =
11             [](const void *p) { return perimeter(CastT::value(p)); },
12         ...
13     };
14   }
15 };
```

LET'S WRITE OUR PIMPL SOO STRATEGY...

```
1  struct pimpl_soo_strategy_no_copy {
2    using PIMPL = pimpl_soo;
3
4    using can_clone = std::false_type;
5
6    template <typename ValueT>
7    requires(SmallObjectOptimizationTrivial<ValueT>)
8    constexpr static auto make_pimpl(ValueT &&x,
9                                     memory_arena &) {
10     return PIMPL::from_soo_trivial(std::forward<ValueT>(x));
11   }
12
13   template <typename ValueT, typename... Args>
14   requires(NonSmallObjectOptimization<ValueT>)
15   static auto make_pimpl(ValueT &&x, Args &&...args,
16                          memory_arena &_arena) {
17     return PIMPL::from_arena(_arena.create<ValueT>(
18         std::forward<ValueT>(x), std::forward<Args>(args)...));
19   }
20
21   template <typename ValueT>
```

INTRODUCE THE 2 CONCEPTS FOR SOO/NON-SOO

```
 1 template <typename ValueT>
 2 concept SmallObjectOptimizationTrivial =
 3     std::is_trivially_copyable_v<std::remove_cvref_t<ValueT>> &&
 4     std::is_trivially_move_constructible_v<std::remove_cvref_t<ValueT>> &&
 5     (std::alignment_of_v<std::remove_cvref_t<ValueT>> <= ALIGN_OF_PIMPL_BUFFER) &&
 6     sizeof(std::remove_cvref_t<ValueT>) <= SIZE_OF_PIMPL_BUFFER;
 7
 8 template <typename ValueT>
 9 concept NonSmallObjectOptimization =
10     !SmallObjectOptimizationTrivial<ValueT>;
```

THE PIMPL_SOO STRUCTURE

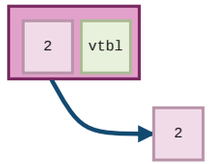
```
1  struct pimpl_soo {
2    using BUFFER = PIMPL_BUFFER<SIZE_OF_PIMPL_BUFFER>;
3
4    union alignas(ALIGN_OF_PIMPL_BUFFER) {
5      void *p;
6      BUFFER buf;
7    };
8
9    constexpr pimpl_soo() noexcept = default;
10   constexpr pimpl_soo(const pimpl_soo &) noexcept = delete;
11   constexpr pimpl_soo(pimpl_soo &&_rhs) noexcept = default;
12   constexpr pimpl_soo &operator=(const pimpl_soo &) noexcept = delete;
13   constexpr pimpl_soo &operator=(pimpl_soo &&_rhs) noexcept = default;
14
15   // Tag dispatching types
16   struct _arena {};
17   struct _soo_trivial {};
18
19   template <typename ValueT>
20   constexpr explicit pimpl_soo(ValueT *_p, _arena)
21     : p(_p) {}
```

THE PIMPL_BUFFER STRUCTURE

```
1  template <SOO_FLAG_TYPE N> struct alignas(ALIGN_OF_PIMPL_BUFFER) PIMPL_BUFFER {
2    char buf[static_cast<size_t>(N)];
3
4    PIMPL_BUFFER() noexcept = default;
5    PIMPL_BUFFER(const PIMPL_BUFFER &) noexcept = default;
6    PIMPL_BUFFER(PIMPL_BUFFER &&) noexcept = default;
7    PIMPL_BUFFER &operator=(const PIMPL_BUFFER &) noexcept = default;
8    PIMPL_BUFFER &operator=(PIMPL_BUFFER &&) noexcept = default;
9
10   template <SOO_FLAG_TYPE M>
11   requires(N < M) PIMPL_BUFFER(const PIMPL_BUFFER<M> &_rhs) {
12     std::memcpy(buf, _rhs.buf, N);
13   }
14
15   constexpr void *data() noexcept { return buf; }
16   constexpr const void *data() const noexcept { return buf; }
17 };
```

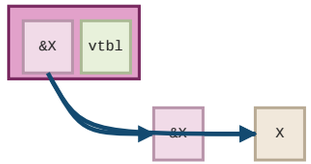
SOO IN ACTION

A *Literal* AST node, e.g. **2**



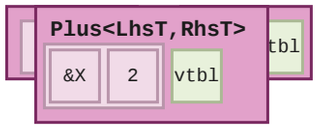
SOO IN ACTION

A *Variable* AST node, e.g. **X**



ONE STEP FURTHER

x + 2



EXAMPLE : PLUS OPERATOR

```
1  // binary_operator.hpp
2  namespace cppexpr::detail {
3  template <typename OperatorT>
4  Node binary_operator(node_builder _builder, OperatorT, Node &&_lhs,
5                       Node &&_rhs);
6
7  struct plus {
8    static INTEGER compute(const INTEGER _lhs, const INTEGER _rhs) {
9      return _lhs + _rhs;
10   }
11   static NUMBER compute(const INTEGER _lhs, const NUMBER _rhs) {
12     return compute(static_cast<NUMBER>(_lhs), _rhs);
13   }
14   static NUMBER compute(const NUMBER _lhs, const INTEGER _rhs) {
15     return compute(_lhs, static_cast<NUMBER>(_rhs));
16   }
17   static NUMBER compute(const NUMBER _lhs, const NUMBER _rhs) {
18     return _lhs + _rhs;
19   }
20   static STRING compute(const STRING &_lhs, const STRING &_rhs) {
21     // Allocate a new string
```

BENCHMARKS — I

Approach	Time (ns)	Factor
c++ expression	3.2	1.0
OO	22.6	7.1
TE + SOO	3.8	1.2
ExprTk	3.3	1.0

Expression `" 3 * (x + 2) "` *evaluated* 10^4 *times, 100 samples*

BENCHMARKS — II

Approach	Time (ns)	Factor
c++ expression	3.5	1.0
TE + SOO	10.2	2.9
ExprTk	6.6	1.9

Expression `" (1.0 + x * (3.0/4.0) ) * ( 8.0 + x / 2.0 ) - 10.0 "`

evaluated 10^4 times, 100 samples

ACHIEVEMENTS

- Math: +, -, ×, ÷, mod, exp, ...
- Logical: !, &&, ||, ..
- String: +, find, substr, contains, split, ...
- List: +, in, nin, [::], join, ...
- Date & Time
- Constant simplification
- SOO
- Efficient
- Can be extended

CONCLUSIONS

- This the nth version of an expression evaluator
- Lots of iterations
- Benchmarks are important
- Learned about SOO
- Still more ideas to test

QUESTIONS ?