Appentra Feedback on Tooling with F18

Unión Europea
Fondo Europeo
de Desarrollo Regional
"Una manera de hacer Europa"

appentra
make code parallel

Appentra

- Small company centered in **static code analysis**
  - Work-in-progress: Data Race Detection for GPUs (and multicore CPUs) using OpenMP and OpenACC
- Proprietary custom engine based on **clang/LLVM technology**
  - Need **LLVM IR** for the analysis
  - Need **detailed source code information** to track the LLVM IR and report back to the user. We use clang for C, and want to use F18 for Fortran
- Years of experience with LLVM and clang (since version 2.9)
- Team of intermediate/advanced C++ developers
- Short experience using Fortran

To keep in mind

- We are talking about the use of F18 for **tooling**
- We are aware that F18 is still **under active development**
- The idea with this talk is to generate debate about the pros and cons
  - We are aware that the current F18 design has its **advantages**
  - But we are going to talk mostly about the **disadvantages**

Why is F18 hard? (I)

Steep learning curve

- No type-based inheritance
  - **Implicit** relation between types: Hard to enumerate, partition or classify them
  - There are no **common** members on related nodes (e.g. “`getParent()`”)

Why is F18 hard? (II)

Steep learning curve

- No type-based inheritance
  - **Implicit** relation between types: Hard to enumerate, partition or classify them
  - There are no **common** members on related nodes (e.g. "`getParent()`")
- No individual names for members of "`std::variant`" or "`std::tuple`"
  - Members are called "u" or "`t`" respectively.
  - There is no clear distinction between node "attributes" or "sub-nodes".

Tuples/variants: Access (F18 approach)

```
OBSCURE_MACRO_DECLARING_NEW_CLASS(Name, std::string)
OBSCURE_MACRO_DECLARING_NEW_CLASS(Bound, int)
struct ArrayAccess {
   std::tuple<Name, Bound, Bound> t;
};
```

Accessing elements by position:

```
std::get<0>(node.t)
```

- We lost semantics again
- Magic numbers
- What happens if the tuple gets re-ordered?
  - Potentially hard to find bugs

Tuples/variants: Access (F18 approach) (II)

```
OBSCURE_MACRO_DECLARING_NEW_CLASS(Name, std::string)
OBSCURE_MACRO_DECLARING_NEW_CLASS(Bound, int)
struct ArrayAccess {
   std::tuple<Name, Bound, Bound> t;
};
```

Accessing elements by type:

```
std::get<Bound>(node.t)
```

- Compilation error (users will probably use positions instead)
- This already happens in F18:
  - `std::tuple<ComplexPart, ComplexPart> t;  // real, imaginary`
  - `std::tuple<BoundExpr, BoundExpr> t;`

Tuples/variants: Access (F18 approach) (III)

```
OBSCURE_MACRO_DECLARING_NEW_CLASS(Name, std::string)
OBSCURE_MACRO_DECLARING_NEW_CLASS(LowerBound, int)
OBSCURE_MACRO_DECLARING_NEW_CLASS(UpperBound, int)
struct ArrayAccess {
   std::tuple<Name, LowerBound, UpperBound> t;
};
```

- Now you can correctly access by type!
- But, how do you prevent this from happening?

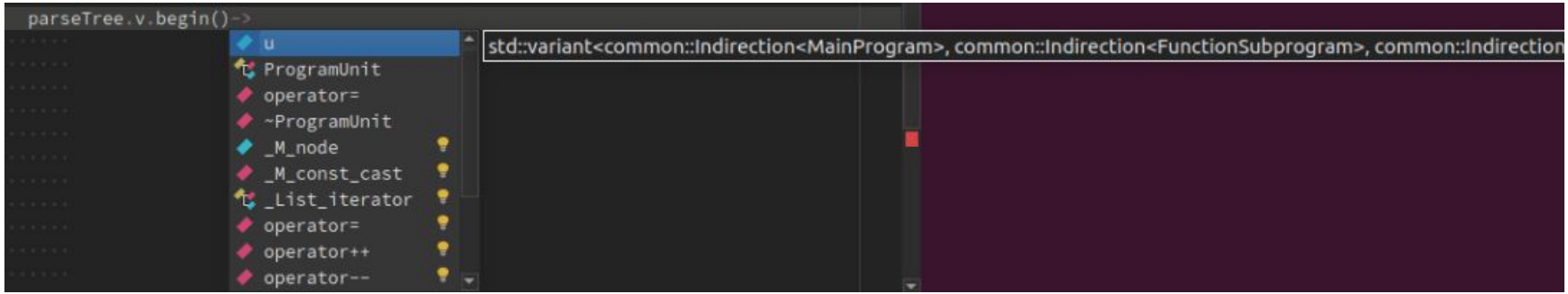
Tuples/variants: Access (Classical approach)

```
struct ArrayAccess : public Expr {
   std::string name;
   int lowerBound;
   int upperBound;
};
```

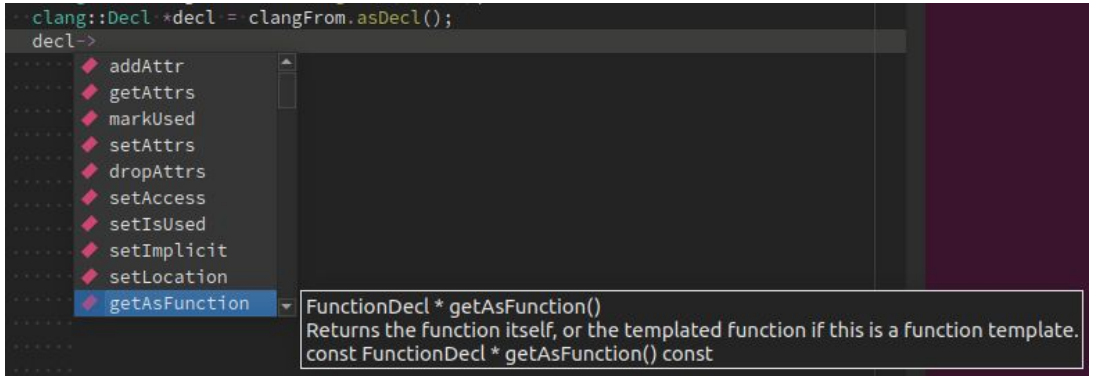
- Explicit member semantic
- Easy to read
- Reordering wouldn't be a problem
- Less performant?

Tuples/variants: Usage

F18



Clang

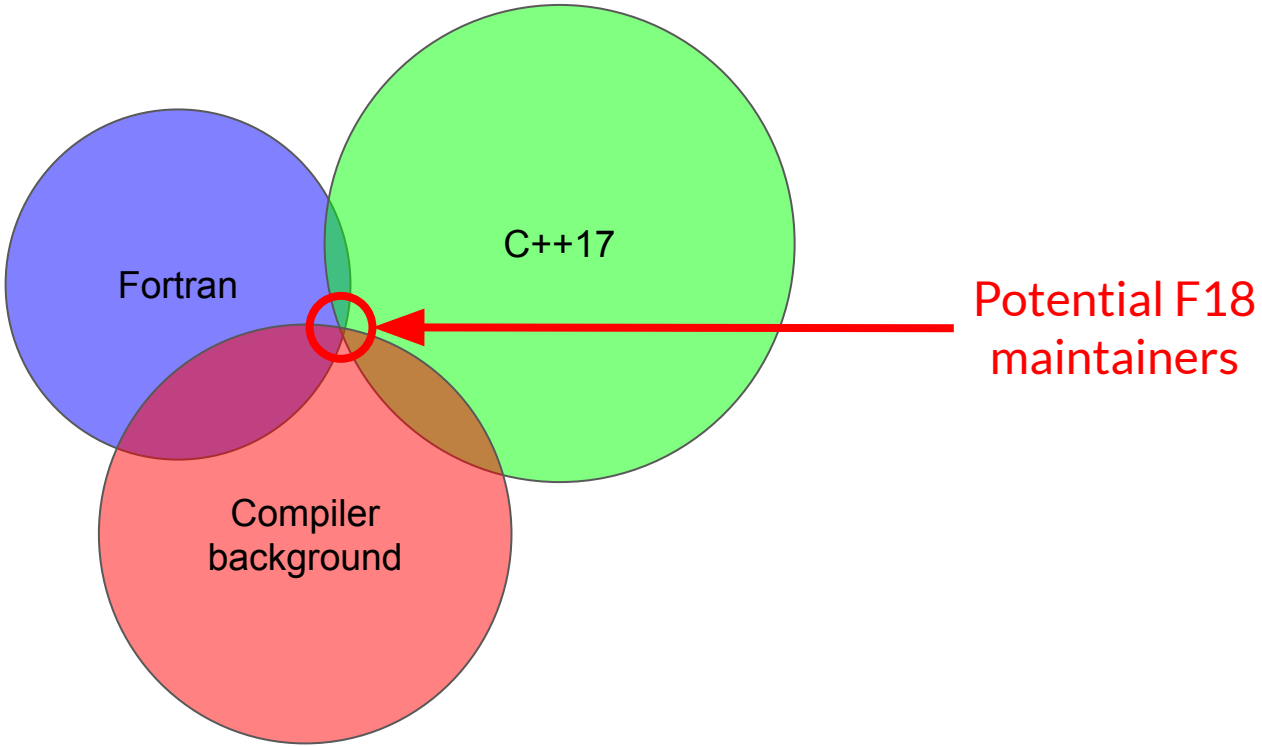


Why is F18 hard? (III)

Steep learning curve

- C++17 “bleeding-edge” features raise the entry level for almost any programmer.
- Close to standard terminology makes the code harder to read.
- There is no common ground with LLVM or Clang.

Future F18 maintainers?



Disclaimer: Non-representative proportions

Why is F18 hard? (IV)

Productivity penalties

- Steep learning curve
- **Slow** build times
- As the abstraction must be made through C++ templates:
  - No code autocompletion
  - No live code diagnostics
  - Harder to understand compilation errors
- Templates and macro usage make code harder to read, write and debug

Slow build times

- In this examples, we didn't change any F18 code, just included the following headers:

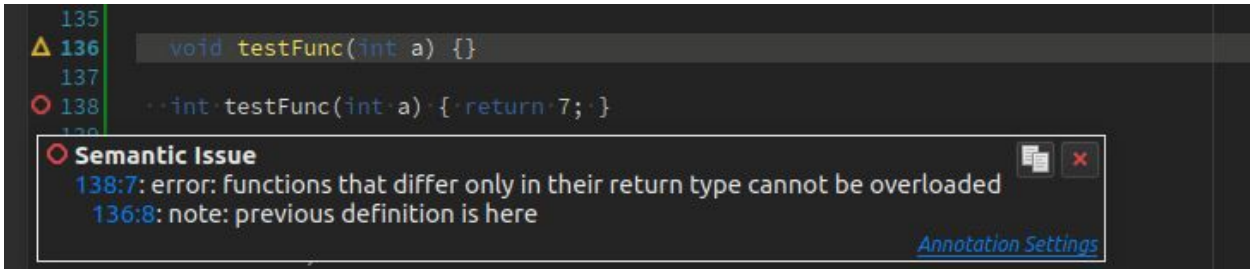
```
#include <parser/parse-tree.h>
#include <parser/provenance.h>
#include <parser/source.h>
#include <semantics/semantics.h>
#include <semantics/type.h>
```

Single file build

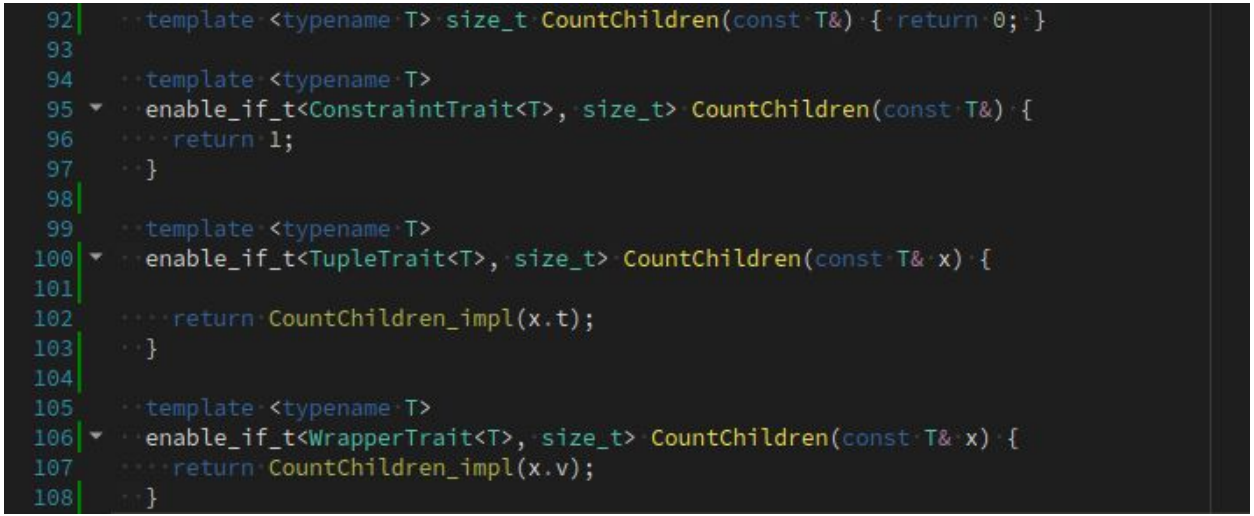
- ~40 seconds on success
- ~15 seconds on error

IDEs support: no live code diagnostics (I)

How we usually write code:



IDEs support: no live code diagnostics (II)



```
92   template <typename T> size_t CountChildren(const T&) { return 0; }
93
94   template <typename T>
95   enable_if_t<ConstraintTrait<T>, size_t> CountChildren(const T&) {
96     return 1;
97   }
98
99   template <typename T>
100  enable_if_t<TupleTrait<T>, size_t> CountChildren(const T& x) {
101
102    return CountChildren_impl(x.t);
103  }
104
105  template <typename T>
106  enable_if_t<WrapperTrait<T>, size_t> CountChildren(const T& x) {
107    return CountChildren_impl(x.v);
108  }
```

There is a compilation error if we call the following code:

```
void foo(const Fortran::parser::MainProgram &x) {
   std::cout << CountChildren(x) << "\n";
}
```

Harder to understand compilation errors (I)

```
In file included from <...>/children-counter/main.cpp:15:
<...>/children-counter/children-counter.h:80:13: fatal error: call to member function 'CountChildren' is
ambiguous
        << CountChildren(x) << "\n";
           ^~~~~~~~~~~~~
<...>/parser/parse-tree-visitor.h:191:15: note: in instantiation of function template specialization
'ChildrenCounter::Pre<Fortran::parser::MainProgram>' requested here
 if (mutator.Pre(x)) {
             ^
<...>/parser/parse-tree-visitor.h:248:3: note: in instantiation of function template specialization
'Fortran::parser::Walk<Fortran::parser::MainProgram, ChildrenCounter>' requested here
 Walk(x.value(), mutator);
 ^
<...>/parser/parse-tree-visitor.h:148:31: note: in instantiation of function template specialization
'Fortran::parser::Walk<Fortran::parser::MainProgram, ChildrenCounter>' requested here
   std::visit([&](auto &y) { Walk(y, mutator); }, x);
                             ^
```

Harder to understand compilation errors (II)

```
<...>/parser/parse-tree-visitor.h:207:5: note: in instantiation of function template specialization
'Fortran::parser::Walk<ChildrenCounter, Fortran::common::Indirection<Fortran::parser::MainProgram, false>,
Fortran::common::Indirection<Fortran::parser::FunctionSubprogram, false>,
Fortran::common::Indirection<Fortran::parser::SubroutineSubprogram, false>,
Fortran::common::Indirection<Fortran::parser::Module, false>, Fortran::common::Indirection<Fortran::parser::Submodule,
false>, Fortran::common::Indirection<Fortran::parser::BlockData, false> >' requested here
   Walk(x.u, mutator);
   ^
<...>/parser/parse-tree-visitor.h:88:5: note: in instantiation of function template specialization
'Fortran::parser::Walk<Fortran::parser::ProgramUnit, ChildrenCounter>' requested here
   Walk(elem, mutator);
   ^
<...>/parser/parse-tree-visitor.h:222:5: note: in instantiation of function template specialization
'Fortran::parser::Walk<Fortran::parser::ProgramUnit, ChildrenCounter>' requested here
   Walk(x.v, mutator);
   ^
<...>/children-counter/main.cpp:70:14: note: in instantiation of function template specialization
'Fortran::parser::Walk<Fortran::parser::Program, ChildrenCounter>' requested here
 f18parser::Walk(parseTree, childCounter);
            ^
```

Harder to understand compilation errors (III)

```
<...>/children-counter/children-counter.h:99:32: note: candidate function [with T = Fortran::parser::MainProgram]
 template <typename T> size_t CountChildren(const T&) { return 0; }
                             ^
<...>/children-counter/children-counter.h:111:38: note: candidate function [with T = Fortran::parser::MainProgram]
 enable_if_t<TupleTrait<T>, size_t> CountChildren(const T& x) {
                                    ^
<...>/children-counter/children-counter.h:116:40: note: candidate template ignored: requirement
'WrapperTrait<Fortran::parser::MainProgram>' was not satisfied [with T = Fortran::parser::MainProgram]
 enable_if_t<WrapperTrait<T>, size_t> CountChildren(const T& x) {
                                      ^
<...>/children-counter/children-counter.h:106:32: note: candidate template ignored: could not match
'variant<type-parameter-0-0>' against 'const Fortran::parser::MainProgram'
 template <typename T> size_t CountChildren(const variant<T>& x) {
                              ^
<...>/children-counter/children-counter.h:102:43: note: candidate template ignored: requirement
'ConstraintTrait<Fortran::parser::MainProgram>' was not satisfied [with T = Fortran::parser::MainProgram]
 enable_if_t<ConstraintTrait<T>, size_t> CountChildren(const T&) {
                                         ^
1 error generated.
```

Some of our typical use cases

- **Find a node's parent**
  - We still don't know how to recover a "parent" for a given node. Our only implementation was with an hypothetical visitor.
- **Count the node's children**
  - Meaningless implementation with basic template meta-programming.
- **Retrieve the location of a node**
  - Only certain nodes like "`Expr`" and "`UnlabeledStatement`" have "`source`" members.

Find a node's parent

```
Typename AllNodes = std::variant< /* All F18 nodes */ >;

struct ParentFinder {
public:
 template <typename T> bool Pre(const T& x) {
   if (&x == TargetNode) return false;
   NodePath.push(&x);
   return true;
 }
 template <typename T> void Post(const T&) { NodePath.pop(); }

private:
 void* TargetNode;
 std::stack<AllNodes> NodePath;
};
```

Count the node's children

```
// Entry point
template <typename T, typename... Dummy> size_t CountChildren(const T&, Dummy...) { return 0; }
template <typename T> enable_if_t<ConstraintTrait<T>, size_t> CountChildren(const T&) { return 1; }
template <typename T> enable_if_t<TupleTrait<T>, size_t> CountChildren(const T& x) { return CountChildren_impl(x.t); }
template <typename T> enable_if_t<WrapperTrait<T>, size_t> CountChildren(const T& x) { return CountChildren_impl(x.v); }
template <typename T> size_t CountChildren(const variant<T>& x) { return CountChildren_impl(UnwrapperHelper::Unwrap(x.u)); }

// Overloads by type
template <typename T, typename... Dummy> size_t CountChildren_impl(const T&, Dummy...) { return 1; }
template <typename T> size_t CountChildren_impl(const list<T>& x) { return x.size(); }
template <typename T> size_t CountChildren_impl(const optional<T>& x) { if (x) { return CountChildren_impl(*x); } return 0; }
template <typename T> size_t CountChildren_impl(const variant<T>&) { return 1; }
template <typename... T> size_t CountChildren_impl(const tuple<T...>& x)
  { return std::apply([this](const auto&... elems) { return TupleChildrenSum(elems...); }, x); }

// Tuple-specific counter
template <typename T> size_t TupleChildrenSum(const T&) { return 1; }
template <typename T> size_t TupleChildrenSum(const optional<T>& x) { return x ? 1 : 0; }
template <typename T> size_t TupleChildrenSum(const list<T>& x) { return x.size(); }
 template <typename T, typename... Elems>
 size_t TupleChildrenSum(const T& x, const Elems&... elems) {
   return TupleChildrenSum(elems...) + TupleChildrenSum(x);
 }
```

Count the node's children (in clang)

```
struct ClangSimpleChildrenCounter {
  std::size_t CountChildren(clang::Decl *D) {
    if (auto *DC = dyn_cast<clang::DeclContext>(D)) {
      return std::distance(DC->decls_begin(), DC->decls_end());
    }
    return 0;
  }

  std::size_t CountChildren(clang::FunctionDecl *D) { return D->param_size() + (D->hasBody() ? 1 : 0); }

  std::size_t CountChildren(clang::VarDecl *D)      { return D->hasInit(); }

  std::size_t CountChildren(clang::Stmt *S)         { return std::distance(S->child_begin(), S->child_end()); }

  std::size_t CountChildren(clang::DeclStmt *S)     { return std::distance(S->decl_begin(), S->decl_end()); }
};
```

Count the node's children (in FC)

```
class FCSimpleChildrenCounter {
public:
  std::size_t CountChildren(fc::ast::ProgramUnit *node) {
    return node->getProgramUnitList().size() + 2; // + SpecPart + ExecPart
  }

  std::size_t CountChildren(fc::ast::Block *node) {
    return node->getStmtList().size();
  }

  std::size_t CountChildren(fc::ast::Stmt *node) {
    return node->getOperands().size();
  }
};
```

Open questions

- Why not use plain data objects and inheritance to define the core objects?
- How important is the developers' productivity?
- How to make the project appealing to new developers?
- **Are these programming techniques really worth?**

Appentra Feedback on Tooling with F18

Unión Europea
Fondo Europeo
de Desarrollo Regional
"Una manera de hacer Europa"

appentra
make code parallel