Compiler Construction

Further with Visitors

Goals & Non-Goal

Tips and Ticks ...

...to improve visitors in C++

You must understand ideas, not necessarily how to implement them!

Const Visitor

Idea

Ensure that some visitor will not modify the AST

Similar to **iterator** and **const_iterator**

- Use C++ templates to factor **Visitor** and **ConstVisitor**
- Use C++ overloading to have only **visit** instead of **visitBin** and **visitNum**

Const and non-const Default Visitors

Problem Description

If we are only interested in variable declarations...
$\Rightarrow$ We still have to write a full visitor

Solution

Write a **DefaultVisitor**!
Use inheritance to process!

Visitor Combinators

- Work and traversal are still too heavily interrelated

→ Create visitors from basic traversal bricks: *combinators*

Combinator	Description
Identity	Do nothing.
Sequence(v_1, v_2)	Sequentially run visitor v_1 then v_2.
Fail	Raise an exception.
Choice(v_1, v_2)	Try visitor v_1; if v_1 fails, try v_2.
All(v)	Apply visitor v sequentially to every immediate subtree.
One(v)	Apply visitor v sequentially to the immediate subtrees until it succeeds.

Object function (1/2)

Use **overloading** and **operator()** instead of **visit***
⇒ Pure convenience

```
struct Evaluator : public ConstVisitor {
  void operator()(const Exp& e) override { e.accept(*this); }
  void operator()(const Num& e) override { value = e.val; }

  void operator()(const Bin& e) override {
    e.lhs()->accept(*this); int lhs = value;
    e.rhs()->accept(*this); int rhs = value;
    value = lhs + rhs;
  }

  int value;
};
```

Object Function (2/2)

```
int eval(const Exp& e) {
  auto eval = Evaluator{};
  eval(e);
  return eval.value;
}
```

Going further... (very technical)

```
struct Evaluator : public ConstVisitor{
  int eval(const Exp& e) {
    e.accept(*this); return value;
  }

  void operator()(const Exp& e) { e.accept(*this); }
  void operator()(const Bin& e) override {
    value = eval(e.lhs()) + eval(e.rhs());
  }
  void operator()(const Num& e) override {
    value = e.val;
  }

  int value;
};
```

Remark on the pretty printer

Applying the same strategy to pretty printer works!

- use overloading
- define an external print method

Using `operator<<`

... will no longer work if we want to pass additional data!
⇒ Use xalloc!

Summary

Combinator

Default visitors

Const visitors

Object Functions

xalloc