# Appendix: recursive variants

Section 8

# In this section

- Definition of recursive variants
- Visitation of recursive variants

# Recursive data structures

- Variants can be defined in a recursive manner in order to model recursive data structures. E.g.
  - JSON
  - Abstract syntax trees
  - Arithmetical expressions
- Some sort of indirection is required, as the size of the variant type must be fixed and known at compile-time

# Recursive data structures - *arithmetical expression* example

```
<number> ::= `int`
<op>     ::= `plus` | `minus`
<expr>   ::= <number> | <number> <op> <expr>
```

```
using number = int;

struct plus  { };
struct minus { };
using op = std::variant<plus, minus>;
```

# Recursive data structures - *arithmetical expression* example

```
<number> ::= `int`
<op>     ::= `plus` | `minus`
<expr>   ::= <number> | <number> <op> <expr>
```

```
struct expr;

using r_expr = std::tuple<number, op, expr>;

struct expr
{
    std::variant<number, std::unique_ptr<r_expr>> _data;
};
```

(on *godbolt.org*)

# Recursive data structures - *arithmetical expression* example

| | |
|---|---|
| `e0` | $5$ |
| `e1` | $9 + 3$ |
| `e2` | $1 - (3 + 7)$ |

```
expr e0{5};

expr e1{make_unique<r_expr>(9, plus{}, 3)};

expr e2{make_unique<r_expr>(1, minus{},
            make_unique<r_expr>(3, plus{}, 7))};
```

# Recursive data structures - visitation

- A `struct` with overloaded `operator()` will be used.
- One or more `operator()` overloads will *recursively visit* the variant by invoking `std::visit` on the parent `struct`.

# Recursive data structures - visitation

```
struct evaluator
{
    auto operator()(number x) { return x; }
    auto operator()(const std::unique_ptr<r_expr>& x)
    {
        const auto& [lhs, op, rhs] = *x;
        const auto rest = std::visit(*this, rhs._data);

        return match(
            [&](plus) { return lhs + rest; },
            [&](minus){ return lhs - rest; })(op);
    }
};
```

# Recursive data structures - visitation

```
// 5
expr e0{5};

// 9 + 3
expr e1{make_unique<r_expr>(9, plus{}, 3)};

// 1 - (3 + 7)
expr e2{make_unique<r_expr>(1, minus{},
            make_unique<r_expr>(3, plus{}, 7))};
```

```
cout << std::visit(evaluator{}, e0._data); // "5"
cout << std::visit(evaluator{}, e1._data); // "12"
cout << std::visit(evaluator{}, e2._data); // "-9"
```

(on *wandbox.org*)