std::accumulate

Nikolai Kutiavin

April 29, 2025

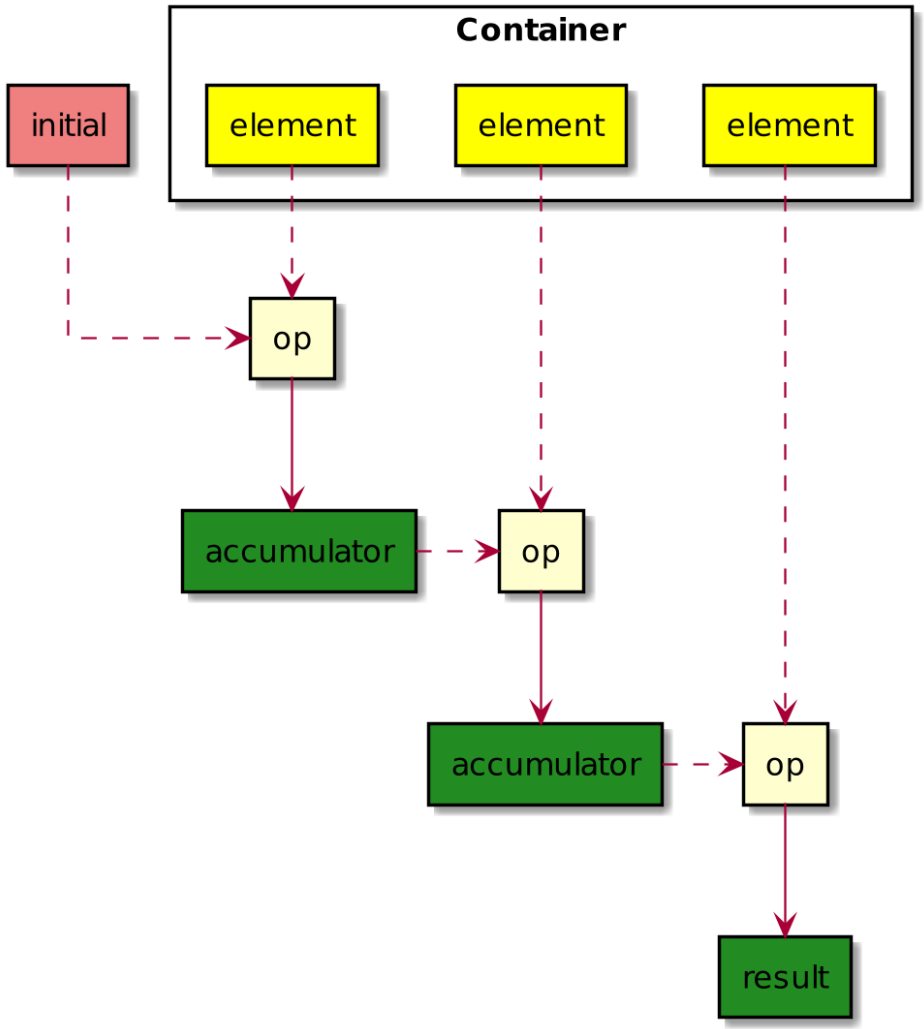
About me

- My name Nikolai
- I'm a C++ developer in BMW

Prototypes

```
1 template< class InputIt, class T, class BinaryOperation >
2 T accumulate( InputIt first, InputIt last, T init,
3                BinaryOperation op );
```

How does it work



Example

```
1 const auto numbers = std::vector<int>{1, 1, 2, 3, 4};
2
3 const auto sum =
4   std::accumulate(numbers.cbegin(), numbers.cend(), 0);
5
6 const auto mul = std::accumulate(numbers.cbegin(),
7                                   numbers.cend(), 1, std::multiplies{});
```

Max number in the strings

```
const auto strings =
       std::vector<std::string>{"one_23123", "two_43",
                                "three_551222"};
const auto max_number =
  std::accumulate(strings.cbegin(), strings.cend(),
   std::size_t{0},
   [re=std::regex{"\\w+_(\\d+)"}](auto acc, const auto &s) {
     if(std::cmatch m; std::regex_match(s.c_str(), m, re)) {

       std::size_t v = 0;
       auto num=std::from_chars(m[1].first, m[1].second, v);

       if (num.ec == std::errc{}) {
         return std::max(acc, v);
       }
     }
     return acc;
  });
```

C++20 features

- constexpr
- accumulator is moved: acc = op(std::move(acc), *i)

Employ move semantic

```
class Analyzer {
  public:
    void accept(const std::string &s);
    bool isValid() const ;
};

std::istringstream str("one two three even");

auto analyzer =
  std::accumulate(
        std::istream_iterator<std::string>(str),
        std::istream_iterator<std::string>{},
        Analyzer{},
        [](auto &&analyzer, const std::string &s){
          analyzer.accept(s);
          return analyzer;
  });
if(analyzer.isValid()) {
  // do something
}
```

std::reduce

```
1 template< class ExecutionPolicy, class ForwardIt, class T,
      class BinaryOp >
2 T reduce( ExecutionPolicy&& policy,
3           ForwardIt first, ForwardIt last, T init, BinaryOp
      binary_op );
```

Result of

- binary_op(init, *first)
- binary_op(*first, init)
- binary_op(init, init)
- binary_op(*first, *first)

must be **convertible** to T.

Max number in the strings with std::reduce

```
1  class MaxNumber {
2    public:
3      MaxNumber(const string &re) : m_re{re} {}
4    public:
5      size_t operator()(size_t acc, const string &s);
6      size_t operator()(const string &s, size_t acc);
7      size_t operator()(size_t acc1, size_t acc2);
8      size_t operator()(const string &s1, const string &s2);
9
10   private:
11     regex m_re;
12
13 };
14 // ...
15 const auto max_number =
16       std::reduce(std::execution::par,
17       strings.cbegin(), strings.cend(),
18       std::size_t{0}, MaxNumber{"\\w+_(\\d+)"});
```

Ranges C++23

```
constexpr auto std::ranges::fold_left( R&& r, T init, F f);

f(f(f(f(init, x1), x2), ...), xn)
```

```
constexpr auto std::ranges::fold_right( R&& r, T init, F f);

f(x1, f(x2, ...f(xn, init)))
```

Max number in the strings with ranges

```cpp
const auto strings =
       std::vector<std::string>{"one_23123", "tow_43",
                                "three_551222"};

const auto max_number = std::ranges::fold_left(strings,
 std::size_t{0},
 [re=std::regex{"\\w+_(\\d+)"}](auto acc, const auto &s) {
   if(std::cmatch m; std::regex_match(s.c_str(), m, re)) {
     std::size_t v = 0;
     auto num=std::from_chars(m[1].first, m[1].second, v);

           if (num.ec == std::errc{}) {
               return std::max(acc, v);
           }
   }
   return acc;
 });
```

Questions?