Range-v3

Nowa biblioteka standardowa C++

Adam Mizerski
adam@mizerski.pl

Warsaw C++ Users Group

24 maja 2017

STL dziś

```
1  auto v = std::vector<int>{};
2  std::generate_n(std::back_inserter(v), 10, my_rand(0, 10));
3  std::sort(v.begin(), v.end());
4  v.erase(std::unique(v.begin(), v.end()), v.end());
5  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, "\n"));
```

STL dziś

```
auto v = std::vector<int>{};
std::generate_n(std::back_inserter(v), 10, my_rand(0, 10));
std::sort(v.begin(), v.end());
v.erase(std::unique(v.begin(), v.end()), v.end());
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, "\n"));
```

- 10 iteratorów


ITERATORY
ITERATORY WSZĘDZIE
memegenerator.net

Co to jest iterator?

Co to jest iterator?

Lekki interfejs do przeglądania kontenera.

- $*$it
- $++$it

- jak pointer do tablicy
- kopie, wszędzie kopie

Co to jest iterator?

Lekki interfejs do przeglądania kontenera.

- *it
- ++it

Co to jest iterator?

Kategorie

- OutputIterator (write, increment single pass)
- InputIterator (read, compare, increment single pass)
- ForwardIterator (increment multiple pass)
- BidirectionalIterator (decrement)
- RandomAccessIterator
- ContiguousIterator (C++17)

Co to jest iterator?

Co to jest iterator?
Kategorie

- OutputIterator (write, increment single pass)
- InputIterator (read, compare, increment single pass)
- ForwardIterator (increment multiple pass)
- BidirectionalIterator (decrement)
- RandomAccessIterator
- ContiguousIterator (C++17)

- To nie są typy, tylko koncepty
- Podanie InputIterator do std::sort spowoduje brzydki błąd kompilacji

Co to jest iterator?

Zakresy

```
#define ALL(v) v.begin(), v.end()
```

```
1  template<class InputIt, class UnaryFunction>
2  UnaryFunction for_each(InputIt first, InputIt last, UnaryFunction f)
3  {
4          for (; first != last; ++first) {
5                  f(*first);
6          }
7          return f;
8  }
```

```
#define ALL(v) v.begin(), v.end()
```

```
template<class InputIt, class UnaryFunction>
UnaryFunction for_each(InputIt first, InputIt last, UnaryFunction f)
{
        for (; first != last; ++first) {
                f(*first);
        }
        return f;
}
```

- iteratory brane przez wartość = kopia
- zakres półotwarty?

Co jest nie tak z iteratorami?

Problem 1

std::istream_iterator<std::string>

Problem 1

std::istream_iterator<std::string>

- Czyta ze strumienia w konstruktorze
- Trzyma kopię std::string
- end trzyma pusty std::string
- pair<It,It> jest gruba

Problem 2

end

Problem 2

end

begin i end muszą być tego samego typu

begin i end muszą być tego samego typu

- czas na dowcip

Problem 2

end

Co to jest czasoprzestrzeń?
„Kopiecie rów stąd do wieczora”

Co to jest czasoprzestrzeń?
„Kopiecie rów stąd do wieczora”

- para iterator, predykat
- np. string od początku do null
- teraz trzeba przeglądać 2 razy

Problem 3

reference_type == value_type&

Problem 3

reference_type == value_type&

- std::vector<**bool**>
- zip

Range-v3

STL dziś

przypomnienie

```
1  auto v = std::vector<int>{};
2  std::generate_n(std::back_inserter(v), 10, my_rand(0, 10));
3  std::sort(v.begin(), v.end());
4  v.erase(std::unique(v.begin(), v.end()), v.end());
5  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, "\n"));
```

Range-v3

```
1  namespace r = ranges;
2  namespace ra = r::action;
3  namespace rv = r::view;
4
5  auto v = rv::generate_n(my_rand(0, 10), 10) |
6           r::to_<std::vector<int>>();
7  v |= ra::sort | ra::unique;
8  r::copy(v, r::ostream_iterator<>(std::cout, "\n"));
```

Range-v3

```
1  namespace r = ranges;
2  namespace ra = r::action;
3  namespace rv = r::view;
4
5  const auto v = rv::generate_n(my_rand(0, 10), 10) |
6            r::to_<std::vector<int>>() |
7            ra::sort | ra::unique;
8  r::copy(v, r::ostream_iterator<>(std::cout, "\n"));
```

ranges::action

- gorliwy
- operuje na kontenerach

ranges::view

- leniwy
- lekki jak iterator
- nie modyfikuje danych

Przykłady

pythags

haskell

```haskell
import Control.Monad

pythags = do
          z <- [1..]
          x <- [1..z]
          y <- [x..z]
          guard (x^2 + y^2 == z^2)
          return (x, y, z)

main = print $ take 1000 pythags
```

pythags

haskell

```haskell
import Control.Monad

pythags =
        [1..] >>= \ z ->
        [1..z] >>= \ x ->
        [x..z] >>= \ y ->
        guard (x^2 + y^2 == z^2) >>
        [(x, y, z)]

main = print $ take 1000 pythags
```

pythags

range-v3

```
1  #include <iostream>
2  #include <tuple>
3
4  #include <range/v3/all.hpp>
5  namespace r = ranges;
6  namespace rv = ranges::view;
7
8  template <typename... Args>
9  std::ostream& operator<<(std::ostream& o, const std::tuple<Args...>& t)
10 {
11         o << '(';
12         r::tuple_for_each(t, [&o](const auto& v) {
13                 o << v << ',';
14         });
15         o << ')';
16         return o;
17 }
```

pythags

range-v3

```
1  auto pythags()
2  {
3          return rv::for_each(rv::ints(1), [ ](auto z) {
4          return rv::for_each(rv::ints(1, z), [ z](auto x) {
5          return rv::for_each(rv::ints(x, z), [x,z](auto y) {
6          return r::yield_if(
7                  x*x + y*y == z*z,
8                  std::make_tuple(x, y, z)
9          );});});});});
10 }
11
12 int main()
13 {
14         std::cout << (pythags() | rv::take(1000)) << '\n';
15 }
```

pythags

czas

	kompilacja	wykonanie
haskell	0.826s	15.7s
range-v3	5.734s	0.56s

Intel® Core™ i5 M-460 2.53GHz

primes

haskell

```haskell
primes = filterPrime [2..]
         where filterPrime (p:xs) =
                   p : filterPrime [x | x <- xs, x `mod` p /= 0]

main = print $ take 1000 primes
```

primes

range-v3: próba 1

```
1  auto filterPrime(auto rng) {
2          const auto p = r::front(rng);
3          return rv::concat(
4                  rv::single(p),
5                  filterPrime(rng | rv::tail |
6                          rv::remove_if([=](auto x){ return x % p == 0; }))
7          );
8  }
9
10 auto primes() { return filterPrime(rv::ints(2)); }
11
12 int main() { std::cout << (primes() | rv::take(100)) << std::endl; }
```

primes

range-v3: próba 1

```
1  auto filterPrime(auto rng) {
2          const auto p = r::front(rng);
3          return rv::concat(
4                  rv::single(p),
5                  filterPrime(rng | rv::tail |
6                          rv::remove_if([=](auto x){ return x % p == 0; }))
7          );
8  }
9
10 auto primes() { return filterPrime(rv::ints(2)); }
11
12 int main() { std::cout << (primes() | rv::take(100)) << std::endl; }
```

Nie kompiluje się, bo typ filterPrime puchnie w nieskończoność.

primes

range-v3

```
1  template <typename T>
2  r::any_view<T> filterPrime(r::any_view<T> rng) {
3          const auto p = r::front(rng);
4          return rv::concat(
5                  rv::single(p),
6                  filterPrime<T>(rng | rv::tail |
7                          rv::remove_if([=](auto x){ return x % p == 0; }))
8          );
9  }
10
11 auto primes() { return filterPrime<int>(rv::ints(2)); }
12
13 int main() { std::cout << (primes() | rv::take(100)) << std::endl; }
```

primes
range-v3

```
1  template <typename T>
2  r::any_view<T> filterPrime(r::any_view<T> rng) {
3          const auto p = r::front(rng);
4          return rv::concat(
5                  rv::single(p),
6                  filterPrime<T>(rng | rv::tail |
7                          rv::remove_if([=](auto x){ return x % p == 0; }))
8          );
9  }
10
11 auto primes() { return filterPrime<int>(rv::ints(2)); }
12
13 int main() { std::cout << (primes() | rv::take(100)) << std::endl; }
```

Zapętla się, próbując gorliwie wyliczyć wynik.

primes

range-v3

```
1  template <typename Rng, typename Fun>
2  auto lazy(Rng&& rng, Fun&& fun)
3  { return rv::single(rng) | rv::transform(fun) | rv::join; }
4
5  template <typename T>
6  r::any_view<T> filterPrime(r::any_view<T> rng) {
7          const auto p = r::front(rng);
8          return rv::concat(
9                  rv::single(p),
10                 lazy(rng | rv::tail |
11                         rv::remove_if([=](auto x){ return x % p == 0; }),
12                         filterPrime<T>
13                 )
14         );
15 }
16
17 auto primes() { return filterPrime<int>(rv::ints(2)); }
18
19 int main() { std::cout << (primes() | rv::take(100)) << std::endl; }
```

primes

czas kompilacji

haskell	0.750s
range-v3	8.971s

Intel® Core™ i5 M-460 2.53GHz

primes

czas wykonania

	10
haskell	0.002s
range-v3	0.004s

Intel® Core™ i5 M-460 2.53GHz

primes

czas wykonania

	10	100
haskell	0.002s	0.003s
range-v3	0.004s	0.380s

Intel® Core™ i5 M-460 2.53GHz

primes

czas wykonania

	10	100	1000
haskell	0.002s	0.003s	0.04s
range-v3	0.004s	0.380s	

Intel® Core™ i5 M-460 2.53GHz

primes

czas wykonania

	10	100	1000
haskell	0.002s	0.003s	0.04s
range-v3	0.004s	0.380s	73m

Intel® Core™ i5 M-460 2.53GHz

shell

```
#!/bin/bash

ls test/* | xargs cat | grep "-"
```

shell

range-v3: ls

```
1  namespace fs = boost::filesystem;
2
3  // Out: range<fs::path>
4  auto ls(const fs::path& path)
5  {
6          return r::make_iterator_range(
7                  fs::directory_iterator{path},
8                  fs::directory_iterator{}
9          );
10 }
```

shell

range-v3: cat

```
1  // Out: range<std::string>
2  auto read_file(const fs::path& path)
3  {
4          return
5                  rv::single(std::make_shared<fs::ifstream>(path)) |
6                  rv::transform([](auto streamPtr) {
7                          return r::getlines(*streamPtr);
8                  }) |
9                  rv::join;
10 }
11
12 // Args: range<fs::path>
13 // Out: range<std::string>
14 const auto cat = [](auto rng) {
15         return rng | rv::transform(read_file) | rv::join;
16 };
```

shell

range-v3: xargs

```
1  template <typename Fun>
2  auto xargs(Fun fun)
3  {
4          return r::make_pipeable([fun](auto rng) {
5                  return fun(rng);
6          });
7  }
```

shell

range-v3: grep

```
1  // In: range<std::string>
2  // Out: range<std::string>
3  auto grep(const std::string& pattern)
4  {
5          return rv::remove_if([pattern](const std::string& line) {
6                  return line.find(pattern) == std::string::npos;
7          });
8  }
```

shell

range-v3: main

```
1  // In: range<std::string>
2  const auto print = r::make_pipeable([](auto rng) {
3          r::copy(rng, r::ostream_iterator<>(std::cout, "\n"));
4  });
5
6  int main()
7  {
8          ls("test") | xargs(cat) | grep("-") | print;
9  }
```

shell

czas

	kompilacja	wykonanie
bash	0s	1.430s
range-v3	7.939s	0.846s

100000 plików, w każdym 3 linijki

Intel® Core™ i5 M-460 2.53GHz

shell

strace

```
$ strace -e open,close ./shell
[...]
open("test", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 3
[...]
open("test/0", O_RDONLY)                = 4
-0
-1
open("test/1", O_RDONLY)                = 5
-2
open("test/2", O_RDONLY)                = 6
close(5)                                = 0
-3
-4
open("test/3", O_RDONLY)                = 5
close(6)                                = 0
-5
open("test/4", O_RDONLY)                = 6
close(5)                                = 0
-6
-7
close(3)                                = 0
close(4)                                = 0
close(6)                                = 0
```

Podsumowując...

Korzyści

- Haskell w C++!
- Leniwe przetwarzanie nieskończonych strumieni danych
- komponowalne elementy

Problemy

- długi czas kompilacji
- brak konceptów – okropne błędy kompilacji[1]
- śladowe ilości dokumentacji (ale za to są dobre testy)

[1] Jest 'emulacja' konceptów na szablonach, więc nie jest tragicznie

https://github.com/ericniebler/range-v3

https://github.com/etam/ranges_presentation