Ranges and algorithms

Victor Eijkhout, Susan Lindsey

Fall 2023
last formatted: November 16, 2023

1. **Loop algorithms**

Algorithms: for each, find, filter, …

Apply to ranges: iterable things such as vectors

Compose ranges with views

C++20 ranges

2. Ranged algorithm

With

```cpp
// rangestd/range.cpp
vector<int> generate_data() { return {2,3,4,5,6,7}; };
    /* ... */
  auto v = generate_data();
```

Code:

```cpp
1 // rangestd/range.cpp
2   count = 0;
3   std::ranges::for_each
4     ( v,
5       [&count] (int i) {
6         count += (i<5); }
7     );
8   cout << "Under five: "
9        << count << '\n';
```

Output:

Under five: 3

3. Range composition

Pipeline of ranges and views:

```cpp
// rangestd/range.cpp
vector<int> generate_data() { return {2,3,4,5,6,7}; };
    /* ... */
  auto v = generate_data();
```

Code:

```cpp
1 // rangestd/range.cpp
2   count = 0;
3   std::ranges::for_each
4     ( v |
      std::ranges::views::drop(1),
5       [&count] (int i) {
6         count += (i<5); }
7       );
8   cout << "minus first: "
9        << count << '\n';
```

Output:

minus first: 2

'pipe operator'

4. Iota and take

```
Code:

1 // rangestd/iota.cpp
2 #include <ranges>
3 namespace rng = std::ranges;
4    /* ... */
5  for ( auto n :
6          rng::views::iota(2,6) )
7    cout << n << '\n';
8  cout << "===\n";
9  for ( auto n :
10          rng::views::iota(2)
11          | rng::views::take(4) )
12    cout << n << '\n';
```

```
Output:

2
3
4
5
===
2
3
4
5
```

5. Filter

Filter is a view:

```
Code:
1 // rangestd/filter.cpp
2   vector<float> numbers
3     {1,-2.2,3.3,-5,7.7,-10};
4   auto pos_view =
5     numbers
6     | std::ranges::views::filter
7       ( [] (int i) -> bool {
8           return i>0; }
9         );
10  for ( auto n : pos_view )
11    cout << n << " ";
12  cout << '\n';
```

```
Output:
1 3.3 7.7
```

TACC

Exercise 1: Exercise

Change the filter example to let the lambda count how many
elements were > 0.

6. Range composition

Code:

```cpp
// range/filtertransform.cpp
    vector<int> v{ 1,2,3,4,5,6 };
    /* ... */
    auto times_two_over_five = v
      | rng::views::transform
        ( [] (int i) {
          return 2*i; } )
      | rng::views::filter
        ( [] (int i) {
          return i>5; } );
```

Output:

```
Original vector:
  1, 2, 3, 4, 5, 6,
Times two over five:
  6 8 10 12
```

7. Quantor-like algorithms

```
Code:

1 // rangestd/of.cpp
2   vector<int>
      integers{1,2,3,5,7,10};
3   auto any_even =
4     std::ranges::any_of
5       ( integers,
6         [=] (int i) -> bool {
7           return i%2==0; }
8       );
9   if (any_even)
10    cout << "there was an even\n";
11  else
12    cout << "none were even\n";
```

```
Output:

there was an even
```

Also `all_of`, `none_of`

Iterators

8. Iterate without iterators

```
vector data{2,3,1};
sort( begin(data),end(data) ); // open to accidents
ranges::sort(data);
```

9. Begin and end iterator

Use independent of looping:

```
Code:
1 // stl/iter.cpp
2     vector<int> v{1,3,5,7};
3     auto pointer = v.begin();
4     cout << "we start at "
5         << *pointer << '\n';
6     ++pointer;
7     cout << "after increment: "
8         << *pointer << '\n';
9
10    pointer = v.end();
11    cout << "end is not a valid
     element: "
12        << *pointer << '\n';
13    pointer--;
14    cout << "last element: "
15        << *pointer << '\n';
```

```
Output:

we start at 1
after increment: 3
end is not a valid
    element: 0
last element: 7
```

10. **Erase at/between iterators**

Erase from start to before-end:

```
Code:
1 // iter/iter.cpp
2    vector<int> counts{1,2,3,4,5,6};
3    vector<int>::iterator second =
     counts.begin()+1;
4    auto fourth = second+2;
5    counts.erase(second,fourth);
6    cout << counts[0]
7         << "," << counts[1] <<
     '\n';
```

```
Output:
1,4
```

(Also erasing a single element without end iterator.)

11. **Insert at iterator**

Insert at iterator: value, single iterator, or range:

```cpp
// iter/iter.cpp
    vector<int> counts{1,2,3,4,5,6},
      zeros{0,0};
    auto after_one =
     zeros.begin()+1;
    zeros.insert
      ( after_one,
        counts.begin()+1,
        counts.begin()+3 );
    cout << zeros[0] << ","
         << zeros[1] << ","
         << zeros[2] << ","
         << zeros[3]
         << '\n';
```

```
Output:
0,2,3,0
```

12. Iterator arithmetic

```
auto first = myarray.begin();
first += 2;
auto last  = myarray.end();
last  -= 2;
myarray.erase(first,last);
```

Algorithms with iterators

13. Reduction operation

Default is sum reduction:

```
Code:
1 // stl/reduce.cpp
2 #include <numeric>
3 using std::accumulate;
4     /* ... */
5     vector<int> v{1,3,5,7};
6     auto first = v.begin();
7     auto last  = v.end();
8     auto sum =
      accumulate(first,last,0);
9     cout << "sum: " << sum << '\n';
```

```
Output:
sum: 16
```

14. Reduction with supplied operator

Supply multiply operator:

```cpp
// stl/reduce.cpp
using std::multiplies;
    /* ... */
    vector<int> v{1,3,5,7};
    auto first = v.begin();
    auto last  = v.end();
    ++first; last--;
    auto product =
      accumulate(first,last,2,
                 multiplies<>());
    cout << "product: " << product
      << '\n';
```

Output:

product: 30

15. **Custom reduction function**

```
// stl/reduce.cpp
class x {
public:
  int i,j;
  x() {};
  x(int i,int j) : i(i),j(j)
    {};
};
```

```
// stl/reduce.cpp
    std::vector< x > xs(5);
    auto xxx =
      std::accumulate
      ( xs.begin(),xs.end(),0,
        [] ( int init,x x1 )
      -> int { return x1.i+init;
      }
        );
```

Write your own iterator

16. Vector iterator

Range-based iteration

```
for ( auto element : vec ) {
  cout << element;
}
```

is syntactic sugar around iterator use:

```
for (std::vector<int>::iterator elt_itr=vec.begin();
     elt_itr!=vec.end(); ++elt_itr) {
  element = *elt_itr;
  cout << element;
}
```

17. Custom iterators, 0

Recall that

Short hand:

```
vector<float> v;
for ( auto e : v )
  ... e ...
```

for:

```
for ( vector<float>::iterator
    e=v.begin();
      e!=v.end(); e++ )
  ... *e ...
```

If we want

```
for ( auto e : my_object )
    ... e ...
```

we need an iterator class with methods such as `begin`, `end`, `*` and `++`.

18. **Custom iterators, 1**

Ranging over a class with iterator subclass

Class:

```cpp
// loop/iterclass.cpp
class NewVector {
protected:
  // vector data
  int *storage;
  int s;
    /* ... */
public:
  // iterator stuff
  class iter;
  iter begin();
  iter end();

};
```

Main:

```cpp
// loop/iterclass.cpp
  NewVector v(s);
    /* ... */
  for ( auto e : v )
    cout << e << " ";
```

19. **Custom iterators, 2**

Random-access iterator:

```cpp
// loop/iterclass.cpp
  NewVector::iter& operator++();
  int& operator*();
  bool operator==( const NewVector::iter &other ) const;
  bool operator!=( const NewVector::iter &other ) const;
  // needed to OpenMP
  int operator-( const NewVector::iter& other ) const;
  NewVector::iter& operator+=( int add );
```

Exercise 2

Write the missing iterator methods. Here's something to get you
started.

```cpp
// loop/iterclass.cpp
class NewVector::iter {
private: int *searcher;
    /* ... */
NewVector::iter::iter( int *searcher )
  : searcher(searcher) {};
NewVector::iter NewVector::begin() {
  return NewVector::iter(storage); };
NewVector::iter NewVector::end()    {
  return NewVector::iter(storage+NewVector::s); };
```