【C++】Day42

Class	C++
Date	@January 20, 2022
Material	
Series Number	
Summary	

【Ch10】Generic Algorithms

10.4.3 Reverse Iterators

A reverse iterator is an iterator that traverses a container backward, from the last element toward the first.

A reverse iterator inverts the meaning of increment (and decrement). Incrementing (`++it`) a reverse moves the iterator to the previous element; decrementing (`--it`) moves the iterator to the next element.

The containers, aside from `forward_list`, all have reverse iterators. We obtain a reverse iterator by calling the `rbegin`, `rend`, `crbegin`, and `crend` members. These members return reverse iterators to the last element in the container and one "past"(i.e. one before) the beginning of the container.

The following figure shows the relationship between normal and reverse iterators:

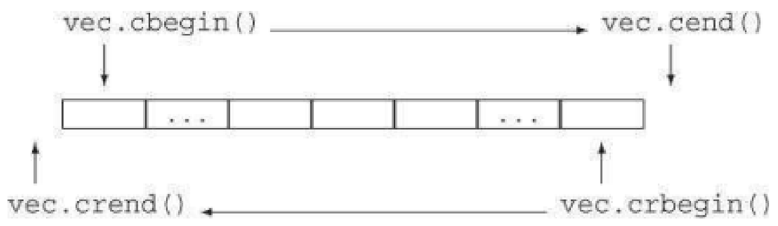


Figure 10.1. Comparing begin/cend and rbegin/crend Iterators

The following code prints the elements in vector in reverse order:

```
vector<int> vec {1, 2, 3, 4, 5, 6, 7};
for(auto r_itr = vec.crbegin(); r_itr != vec.crend(); ++r_itr) {
    std::cout << *r_itr << std::endl;
}
```

Although it may seem confusing to have the meaning of the increment and decrement operators reversed, doing so lets us use the algorithms transparently to process a container forward or backward.

For example, we can sort our vector in descending order by passing sort a pair of reverse iterators:

```
sort(vec.begin(), vec.end()); //sorts vec in "normal" order
sort(vec.rbegin(), vec.rend()); //sort in reverse: puts the smaller element at the end of vec
```

Reverse Iterators Require Decrement Operators

The stream iterators do not support decrement, because it is not possible to move backward through a stream. Therefore, it is not possible to create a reverse iterator from a `forward_list` or a stream iterator.

Relationship between Reverse Iterators and Other iterators

Suppose we have a string named line that contains a comma-separated list of words, and we want to print the first word in line.

Using find, this task is simple:

```
auto comma = find(str.cbegin(), str.cend(), ',');
std::cout << string(str.cbegin(), comma) << std::endl;
```

If we wanted the last words, we can use reverse iterators instead:

```
auto comma = find(str.crbegin(), str.crend(), ',');
std::cout << string(comma.base(), str.cend()) << std::endl;
```

Notice that here we used the `base` member to convert the reverse iterator to an ordinary iterator. It yields an iterator to the next element in the container.

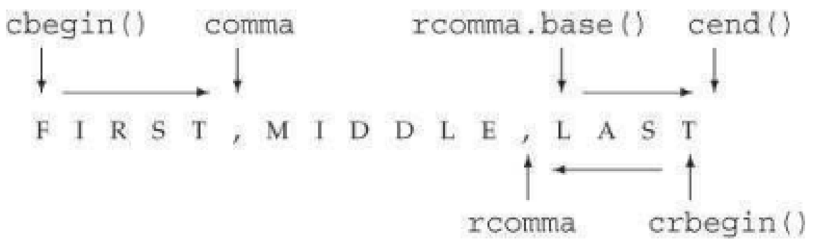
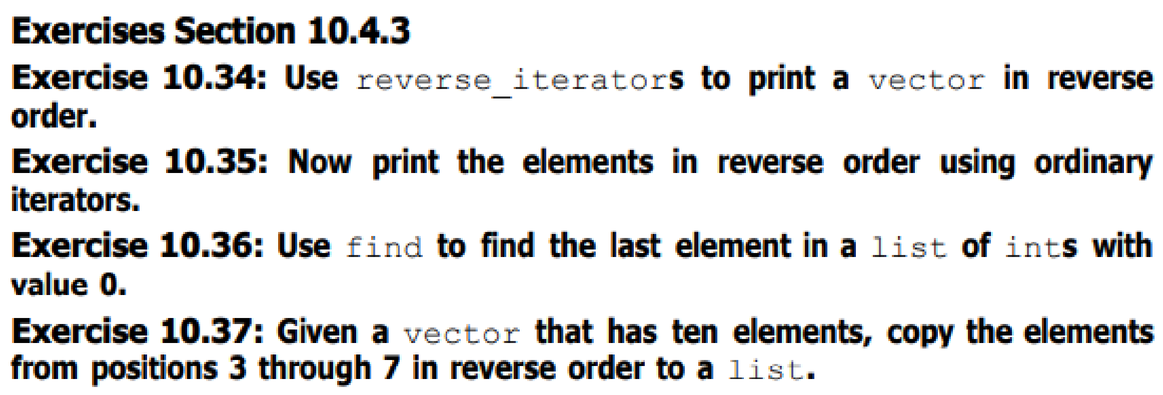


Figure 10.2. Relationship between Reverse and Ordinary Iterators

Exercise



Exercises Section 10.4.3

Exercise 10.34: **Use** `reverse_iterator`**s to print a** `vector` **in reverse order.**

Exercise 10.35: **Now print the elements in reverse order using ordinary iterators.**

Exercise 10.36: **Use** `find` **to find the last element in a** `list` **of** `int`**s with value 0.**

Exercise 10.37: **Given a** `vector` **that has ten elements, copy the elements from positions 3 through 7 in reverse order to a** `list`**.**

See code in the code folder