

【C++】 Day32


▼ Class	C++
📅 Date	@January 4, 2022
🔗 Material	
# Series Number	
☰ Summary	

【Ch9】 Sequential Container

9.2.5 Assignment and swap

The assignment-related operators listed below act on the entire container.

Table 9.4. Container Assignment Operations

<code>c1 = c2</code>	Replace the elements in <code>c1</code> with copies of the elements in <code>c2</code> . <code>c1</code> and <code>c2</code> must be the same type.
<code>c = {a, b, c...}</code>	Replace the elements in <code>c1</code> with copies of the elements in the initializer list. (Not valid for array.)
<code>swap(c1, c2)</code> <code>c1.swap(c2)</code>	Exchanges elements in <code>c1</code> with those in <code>c2</code> . <code>c1</code> and <code>c2</code> must be the same type. <code>swap</code> is usually <i>much</i> faster than copying elements from <code>c2</code> to <code>c1</code> .
assign operations not valid for associative containers or array	
<code>seq.assign(b, e)</code>	Replaces elements in <code>seq</code> with those in the range denoted by iterators <code>b</code> and <code>e</code> . The iterators <code>b</code> and <code>e</code> must not refer to elements in <code>seq</code> .
<code>seq.assign(il)</code>	Replaces the elements in <code>seq</code> with those in the initializer list <code>il</code> .
<code>seq.assign(n, t)</code>	Replaces the elements in <code>seq</code> with <code>n</code> elements with value <code>t</code> .
 WARNING	Assignment related operations invalidate iterators, references, and pointers into the left-hand container. Aside from <code>string</code> they remain valid after a <code>swap</code> , and (excepting arrays) the containers to which they refer are swapped.

The assignment operator replaces the entire range of elements in the left-hand container with copies of the elements from the right-hand operand:

```
c1 = c2; //replace the contents of c1 with a copy of the elements in c2
c1 = {a, b, c}; //after the assignment c1 has size 3
```

After the first assignment, the left- and right-hand containers are equal. If the containers had been of unequal size, after the assignment both containers would have the size of the right-hand operand.

Unlike built-in arrays, the library array type does not allow assignment. The left- and right-hand operands must have the same type:

```
array<int, 10> a1 = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
array<int, 10> a2 = { 0 }; //elements all have value 0
a1 = a2; //replaces elements in a1;
a2 = { 0 }; //error: cannot assign to an array from a braced list
```

Because the size of the right-hand operand might differ from the size of the left-hand operand, the array type does not support `assign` and it does not allow assignment from a braced list of values.

Using assign

The assignment operator `=` requires that the left-hand and right-hand operands have the same type. It copies all the elements from the right-hand operand into the left-hand operand.

The sequential containers also define a member named `assign` that lets us assign from a different but compatible type, or assign from a subsequence of a container.

The assign operation replaces all the elements in the left-hand container with (copies of) the elements specified by its arguments.

For example, we can use assign to assign a range of `char*` values from a `vector` into a `list` of `string`:

```
list<string> names;
vector<const char*> old_style;
```

```
names = old_style; //error: container types don't match
//ok: can convert from const char* to string
names.assign(old_style.cbegin(), old_style.cend());
```

Warning: Because the existing elements are replaced, the iterators passed to assign must not refer to the container on which assign is called.

A second version of `assign` takes an integral value and an element value. It replaces the elements in the container with the specified number of elements, each of which has the specified element type:

```
//equivalent to slist1.clear();
//followed by slist.insert(slist1.begin(), 10, "Haiya!");
list<string> slist1(1); //one element, which is empty string
slist1.assign(10, "Haiya!"); //ten elements; each one is "Haiya!"
```

Using swap

The `swap` operation exchanges the contents of two containers of the same type. After the call to swap, the elements in the two containers are interchanged:

```
vector<string> svec1(10); //vector with ten elements
vector<string> svec2(20); //vector with twenty elements
swap(svec1, svec2);
```

After the swap, `svec1` contains 20 `string` elements and `svec2` contains ten.

The swap operation is guaranteed to be fast-the elements themselves are not swapped; internal data structures are swapped.

Note: Excepting array, `swap` does not copy delete, or insert any elements and is guaranteed to run in constant time.

After the swap, pointers, references, and iterators remain bound to the same element they denoted before the swap. However, those elements are now in a different container.

Exercise

Exercises Section 9.2.5

Exercise 9.14: Write a program to assign the elements from a `list` of `char*` pointers to C-style character strings to a `vector` of `strings`.

```
vector<string> vec(10);  
list<char*> list(24);  
vec.assign(list.cbegin(), list.cend());
```