【C++】Day46

Class	C++
Date	@January 26, 2022
Material	
Series Number	
Summary	

【Ch11】Associative Container

11.3.3 Erasing Elements

The associative containers define three versions of erase, which are described in the table below.

`c.erase(k)`	Removes every element with key `k` from `c`. Returns `size_type` indicating the number of elements removed.
`c.erase(p)`	Removes the element denoted by the iterator `p` from `c`. `p` must refer to an actual element in `c`; it must not be equal to `c.end()`. Returns an iterator to the element after `p` or `c.end()` if `p` denotes the last element in `c`.
`c.erase(b, e)`	Removes the elements in the range denoted by the iterator pair `b`, `e`. Returns `e`.

The associative containers supply an additional erase operation that takes a `key_type` argument. This version removes all the elements, if any, with the given key and returns a count of how many elements were removed.

We can use this version to remove a specific word from `word_count` before printing the results:

```
if(word_count.erase(removal_word))
  std::cout << "ok: " << removal_word << " removed\n" << std::endl;
else
  std::cout << "oops: " << removal_word << " not found" << std::endl;
```

11.3.4 Subscripting a map

The `map` and `unordered_map` containers provide the subscript operator and a corresponding `at` function, which are described below.

`c[k]`	Returns the element with key `k`; if `k` is not in `c`, adds a new, value-initialized element with key `k`.
`c.at(k)`	Checked access to the element with key `k`; throws an `out_of_range` exception (§ 5.6, p. 193) if `k` is not in `c`.

The `set` types do not support subscripting because there is no “value” associated with a key in a set.

We cannot subscript a `multimap` or an `unordered_multimap` because there may be more than one value associated with a given key.

The `map` subscript takes an index and fetches the value associated with that key. However, unlike other subscript operators, if the key is not already present, a new element is created and inserted into the map for that key. The associated value is value initialized.

For example, when we write

```
map<string, size_t> word_count; //empty map
//insert a value-initialized elemenet with key Anna; then assign 1 to its value.
word_count["Anna"] = 1;
```

the following steps take place:

- `word_count` is searched for the element whose key is Anna. The element is not found.
- A new key-value pair is inserted into `word_count`. The key is a `const string` holding `Anna`. The value is value initialized, meaning in this case that value is 0.
- The newly inserted element is fetched and is given the value 1.

Because the subscript operator might insert an element, we may use subscript only on a map that is not const.

Note: Subscripting a map behaves quite differently from subscripting an array or vector: Using a key that is not already present adds an element with that key to the map.

11.3.5 Accessing Elements

The associative containers provide various ways to find a given element, which are described in the table below.

`lower_bound` and `upper_bound` not valid for the unordered containers. Subscript and `at` operations only for `map` and `unordered_map` that are not `const`.

`c.find(k)`	Returns an iterator to the (first) element with key `k`, or the off-the-end iterator if `k` is not in the container.
`c.count(k)`	Returns the number of elements with key `k`. For the containers with unique keys, the result is always zero or one.
`c.lower_bound(k)`	Returns an iterator to the first element with key not less than `k`.
`c.upper_bound(k)`	Returns an iterator to the first element with key greater than `k`.
`c.equal_range(k)`	Returns a `pair` of iterators denoting the elements with key `k`. If `k` is not present, both members are `c.end()`.

Which operation to use depends on what problem we are trying to solve.

- If all we care about is whether a particular element is in the container, it is probably best to use `find`.
- For the containers that can hold only unique keys, it probably doesn't matter whether we use `find` or `count`.

  However, for the containers with multiple keys, `count` has to do more work: If the element is present, it still has to count how many elements have the same key.

If we don't need the count, it's still best to use `find`:

```
set<int> iset = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
iset.find(1); //returns an iterator that refers to the element with key == 1
iset.find(10); //returns an iteator == iset.end()
```

```
iset.count(1); //returns 1
iset.count(10); //returns 0
```

Using find Instead of Subscript for maps

For the `map` and `unordered_map` types, the subscript operator provides the simplest method of retrieving a value.

However, as we've just seen, using a subscript has an important side effect: If that key is not already in the map, then subscript inserts an element with that key.

Sometimes, we wan to know if an element with a given key is present without changing the map. We cannot use the subscript operator to determine whether an element is present, because the subscript operator inserts a new element if the key is not already here.

In such cases, we should use `find`:

```
if(word_count.find("Arthur") == word_count.end())
  cout << "Arthur is not in the map" << endl;
```

Find Elements in a multimap or multiset

For the containers that allow multiple keys, finding an element is more complicated: There may be many elements with the given key.

When a `multimap` or `multiset` has multiple elements of a given key, those elements will be adjacent within the container.

For example, we might want to print all the books by a particular author. We can solve this problem in three different ways. The most obvious way uses find and count:

```
string search_item("Alain de Botton"); //author we'll look for
auto entries = authors.count(searh_item); //number of elements
auto iter = authors.find(search_item);
while(entries) {
```

```
  cout << iter->second << endl;
  ++iter;
  --entries; //keep track of how many we've printed
}
```

Note: We are guaranteed that iterating across a `multimap` or `multiset` returns all the elements with a given key in sequence.