【C++】Day nine(3)

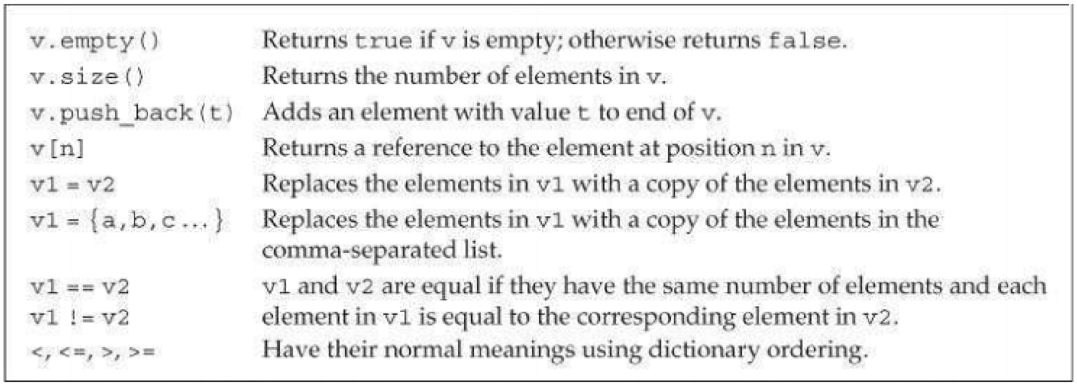
Class	C++
Date	@November 27, 2021
Material	
Series Number	
Summary	

【Ch3】Vector

3.3.3 Other vector Operations

In addition to `push_back`, vector provides only a few other operations, most of which are similar to the corresponding operations on strings.

Table 3.5. vector Operations



`v.empty()`	Returns `true` if `v` is empty; otherwise returns `false`.
`v.size()`	Returns the number of elements in `v`.
`v.push_back(t)`	Adds an element with value `t` to end of `v`.
`v[n]`	Returns a reference to the element at position `n` in `v`.
`v1 = v2`	Replaces the elements in `v1` with a copy of the elements in `v2`.
`v1 = {a,b,c...}`	Replaces the elements in `v1` with a copy of the elements in the comma-separated list.

`v1 == v2`
`v1 != v2`
	`v1` and `v2` are equal if they have the same number of elements and each element in `v1` is equal to the corresponding element in `v2`.

`<, <=, >, >=`	Have their normal meanings using dictionary ordering.

We access the elements of a vector the same way that we access the characters in a string: through their position in the vector.

Note: To use size_type, we must name the type in which it is define. A vector type always includes its element type.

```
vector<int>::size_type; //ok
vector::size_type; //error
```

The relational operators apply a dictionary ordering:

- If the vectors have differing sizes, but the elements that are in common are equal, then the vector with fewer elements is less than the one with mroe elements.
- If the elements have differeing values, then the relationship between the vectors is determined by the relationship between the first elements that differ.

Computing a vector Index

As we've seen, when we use a subscript, we should think about how we know that the indices are in range. Programmers new to C++ sometimes think that subscripting a vector adds elements; it does not. The folloiwng code intends to add ten elements to ivec:

```
vector<int> ivec; //empty vector
for(decltype(ivec.size()) ix = 0; ix != 10; ++ix)
  ivec[ix] = ix; //disaster: ivec has no elements
```

However, it is in error: ivec is an empty vector; there are no elements to subscript!

The right way to write this loop is to use `push_back`:

```
for(decltype(ivec.size()) ix = 0; ix != 10; ++ix)
  ivec.push_back(ix); //ok: adds a new element with value ix
```

Warning: The subscript operator on vector(and string) fetches an existing element; it does not add an element.

3.4 Introducing Iterators

Although we can use subscripts to access the caracters of a string or the elements in a vector, there is a more general mechanism-known as iterators-taat we can use for the same prupose.

Like pointers, iterators give us indiret access to an object. In the case of an iterator, that object is an element in a container or a character in a string. We can use an iterator to fetch an element and iterators have operations to move from one element to another. As with pointers, an iterator may be valid or invalid. A valid iterator either denotes an element or denotes a position one past the last element in a container. All otehr iterator values are invalid.

3.4.1 Using Iterators

Unlike pointers, we do not use the address-of operator to obtrain an iterator. Instead, types that have iterators have members that return iterators. In particular, these types have members named `begin` and `end`. The begin member returns an iterator that dentoes the first element if there is one:

```
auto b = v.begin(), e = v.end(); //b and e have the same type
```

The iterator returned by `end` is an iterator positioned "one past the end" of the associated container. This iterator denotes a nonexistenet element "off the end" of the container. It is used as a marker indicating when we have processed all the elements. The iterator returned by end is often referred to as the off-the-end iterator or abbreviated as "the end iterator."

Note: If the container is empty, begin returns the same iterator as the one returned by end.