【C++】Day32(2)

Class	C++
Date	@January 4, 2022
Material	
Series Number	
Summary	Container Size Operation&Container Comparison

【Ch9】Sequential Container

9.2.6 Container Size Operations

With one exception, the container types have three size-related operations.

- The size member returns the number of elements in the container
- `empty` returns a bool that is true if size is zero and false otherwise
- `max_size` returns a number that is greater than or equal to the number of elements a container of that type can contain.

9.2.7 Relational Operators

Every container type supports the equality operators(== and !=); all the containers except the unordered associative containers also support the relational opertaors(>, >=, < and <=)

The right- and left-hand operands must be the same kind of container and must hold elements of the same type. That is, we can compare a vector<int> only with another vector<int>. We cannot compare a vector<int> with a list<int> or a vector<double>.

Comparing two containers performs a pairwise comparison of the elements. These operators work similarly to the string relationals:

- If borth containers are the same size and all the elements are equal, then the two containers are equal; otherwise, they are unequal

- If the containers have different sizes but every element of the smaller one is equal to the corresponding element of the larger one, then the smaller one is less than the other
- If neither container is an initial subsequence of the other, then the comparison depends on compparing the first unequal elements.

The following examples illustrate how these opeartors work:

```
vector<int> v1 = { 1, 3, 5, 7, 9, 12 };
vector<int> v2 = { 1, 3, 9};
vector<int> v3 = { 1, 3, 5, 7 };
vector<int> v4 = { 1, 3, 5, 7, 9, 12};
v1 < v2; //true: v1 and v2 differ at element [2]
v1 < v3; //false: all elements are equal, but v3 has fewer of them
v1 == v4; //true: each element is equal and v1 and v4 have the same size()
v1 == v2; //false: v2 has fewer elements than v1
```

Note: We can use a relational operator to compare two containers only if the appropriate comparison operator is defined for the element type.

Exercise

Exercise 9.15: Write a program to determine whether two `vector<int>`s are equal.

```
bool isGreater(const vector<int> &vec1, const vector<int> &vec2) {
  return vec1 == vec2;
}

int main(int argc, char **argv) {
  vector<int> vec1(10, 10);
  vector<int> vec2(10, 0);
  if(isGreater(vec1, vec2))
    cout << "Equal";
  return 0;
}
```

Exercise 9.16: Repeat the previous program, but compare elements in a `list<int>` to a `vector<int>`.

```
bool isGreater(const vector<int> &vec1, const list<int> &list) {
  vector<int> vec(list.cbegin(), list.cend());
  return vec1 == vec;
}

int main(int argc, char **argv) {
  vector<int> vec1;
  list<int> list1;
  int temp;

  cout << "Please enter the elements of vec<int>: \n";
  while(cin >> temp)
    vec1.push_back(temp);
  cin.clear();
  cout << "Please enter the element of list<int>: \n";
  while(cin >> temp)
    list1.push_back(temp);

  vector<int>::const_iterator vit = vec1.cbegin();
  list<int>::const_iterator lit = list1.cbegin();
  while(vit != vec1.cend() && lit != list1.cend()) {
    if(*vit < *lit) {
      cout << "List is greater. \n";
      return 0;
    } else if(*vit > *lit) {
      cout << "Vector is greater. \n";
      return 0;
    }
    ++vit, ++ lit;
  }
  if(vec1.size() == list1.size())
    cout << "Vector and list are equal. \n";
  else if(lit == list1.cend())
    cout << "List is less";
  else
    cout << "vector is less";
  return 0;
}
```

Exercise 9.17: Assuming `c1` and `c2` are containers, what (if any) constraints does the following usage place on the types of `c1` and `c2`?

```
if (c1 < c2)
```

1. c1 and c2 needs to be container of the same type(same container type and element type)

2. The type of the elements in the container must be comparable.(< defined for the type)