Lecture 6 Advanced Containers and Iterators

#CS106L

- **Map Iterators**
- **Further Iterator Usages**
- **Announcements**
- **Iterator Types**

---

make_pair automatically deduces the type!

```
        std::pair<string, int> p{"Phone number", 6507232300};

        std::make_pair("Phone number", 6507232300);
```

`std::multimap` is a map that permits multiple entries with the same key.Doesn't have [] operator.Instead, add elements by calling `.insert` on a key value `std::pair`.

```
        map<int, int> m;

        map<int, int>::iterator i = m.begin();

        map<int, int>::iterator end = m.end();

        while(i != end) {

        //the place of these parentheses is super important

        cout << (*i).first << (*i).second << endl;

        ++i;

 }
```

Iterator Types:

1. Input:For sequential, single-pass input,read only i.e. can only be dereferenced on right side of expression.

- Use cases:`find` and `count` , `input streams`

2. Output:For sequential, single-pass output,write only i.e. can only be dereferenced on left side of expression.

- Use cases:`copy` and `output streams`

3. Forward:Combines input and output iterators, + can make multiple passes.It can read from and write to (if not const iterator)

- Use cases:`replace` and `std::forward_list`

4. Bidirectional:Same as forward iterators, `+` can go backwards with the decrement operator `--`.

- Use cases:`reverse` , `std::map` , `std::set` and `std::list`

5. Random access:Same as bidirectional iterators, + can be incremented or decremented by arbitrary amounts using `+` and `-`.

- Use cases:`std::vector` , `std::deque` , `std::string` and `pointer`

All iterators share a few common traits:

- Can be created from existing iterator
- Can be advanced using `++`
- Can be compared with `==` and `!=`

Q1: IDE warning:no operator "+" matches these operands.

```
#include<iostream>

#include<list>
```

```
using namespace std;

int main(){

    list<int> mylist(10);

    auto it = mylist.begin() + 3;

    cout << *it;

    return 0;

}
```

The program will raise a compile error because the operator `+` is not defined for iterators of `std::list` and `std::forward_list`, so you can't use it to move the iterator forward.

In the `std::list` containers, the elements are stored in a doubly-linked list, which means they are not stored in a linear fashion like an array or a vector. You can't use operator [] or pointer arithmetic to access the elements in the list, and therefore you can't use the operator "+" to move an iterator to a specific position.

To achieve the goal of pointing to the 4th element of the list, you can use the `std::next` function from the `<iterator>` header which returns an iterator that points to the nth element after the current iterator, or use a for loop to iterate through the elements of the list and stop at the 4th element.

Here's an example of how you can use `std::next` function to get an iterator that points to the 4th element of the list:

```
#include<iostream>

#include<list>

#include<iterator>

using namespace std;

int main(){

    list<int> mylist(10);
```

```
    auto it = std::next(mylist.begin(), 3);

    cout << *it;

    return 0;

}
```