Associative Containers and Iterators

Associative Containers is a data structure that can associate values
data is accessed using **key** instead of index

```
std::map<T1,T2>
std::set<T>
std::unordered_map<T1, T2>
std::unordered_set<T>
```

A trick learned from this class

```
map<string, int>
string key;
if the key is existed,
    map[key] will return the corresponding value
else
    map[key] will automatically insert <key, 0> pair and return 0;

//if you want to check whether a key is existed in map,
case 1: if (map.find(key) != map.end())//I usually do this
case 2: if (map.count(key))//same effect but much more elegent
/*
However, case1 is more descriptive and more efficient.
Since map.find(key) will return as long as the key is found
map.count(key) has to traverse the whole map before return
*/
```

Iterators

The standard interface to loop through things

Associate containers has no notion of indexing

- Iterator let us view a `non-linear` collection in a `linear` manner

Get the value of iterator by using dereference operator `*`
Advance iterator by using operator `++`

Uses of Iterators

```
Sort: std::sort(vec.begin(), vec.end();

Find:
vec<int>::iterator it = std::find(vec.begin(), vec.end());
if(it != vec.end()) {
cout << "Found: " << *it << endl;
} else {
cout << "Element not found!" << endl;
}

Range:
set<int>::iterator i = mySet.lower_bound(7);
set<int>::iterator end = mySet.lower_bound(26);
```

	[a, b]	[a, b)	(a, b]	(a, b)
begin	lower_bound(a)	lower_bound(a)	upper_bound(a)	upper_bound(a)
end	upper_bound(b)	lower_bound(b)	upper_bound(b)	lower_bound(b)

lower_bound(a): smallest integer that is equal to or greater than a

$$x = \text{lower\_bound}(a), \text{where } x \geq a$$

upper_bound(a): smallest integer that is strictly greater than a

$$x = \text{upper\_bound}(a), \text{where } x > a$$

auto

`auto` is a C++11 features that uses `type reduction`
Asks the compiler to figure out the type for you.

When to use it?

- Use it whenever the type is **obivous**, e.g. iterator
- In places where only the compiler knows the type

Range Based for Loop

```
for (auto item : container) {
    //do something
}
```