

The Standard C++ Library - Looping on containers

Version 1: Dr. Ofir Pele

Version 2: Dr. Erel Segal-Halevi

Iterators & Containers

```
class NameOfContainer {  
    ...  
    typedef ... iterator; // iterator type  
    iterator begin();      // first element  
    iterator end();        // element after last
```

```
NameOfContainer<...> c
```

```
...
```

```
NameOfContainer<...>::iterator it;  
for( it= c.begin(); it!=c.end(); ++it)  
    // do something that changes *it
```

Iterators & Containers: **c++11**

```
class NameOfContainer {  
    ...  
    typedef ... iterator; // iterator type  
    iterator begin();      // first element  
    iterator end();        // element after last
```

```
NameOfContainer<...> c
```

```
...
```

```
for(auto it= c.begin(); it!=c.end(); ++it)  
    // do something that changes *it
```

Iterators & Containers: **c++11**

```
class NameOfContainer {  
    ...  
    typedef ... iterator; // iterator type  
    iterator begin();      // first element  
    iterator end();        // element after last
```

```
NameOfContainer<...> c
```

```
...
```

```
for(auto& val : c)
```

```
    // do something that changes val
```

const_iterators & Containers

```
class NameOfContainer {  
    ...  
    typedef ... const_iterator; // iterator type  
    const_iterator begin() const;    // first element  
    const_iterator end() const;      // element after last
```

```
    NameOfContainer<...> c
```

```
    ...
```

```
    NameOfContainer<...>::const_iterator it;
```

```
    for( it= c.begin(); it!=c.end(); ++it)
```

```
        // do something that does not change *it
```

const_iterators & Containers: c++11

```
class NameOfContainer {  
    ...  
    typedef ... const_iterator; // iterator type  
    const_iterator cbegin() const;    // first element  
    const_iterator cend() const;      // element after last
```

```
    NameOfContainer<...> c
```

```
    ...
```

```
    for(auto it= c.cbegin(); it!=c.cend(); ++it)  
        // do something that does not change *it
```

const_iterators & Containers: c++11

```
class NameOfContainer {  
    ...  
    typedef ... const_iterator; // iterator type  
    const_iterator cbegin() const;    // first element  
    const_iterator cend() const;      // element after last
```

```
NameOfContainer<...> c
```

```
...
```

```
for(const auto& val : c)
```

```
    // do something that does not change val
```

const_iterators & Containers

...

```
const_iterator cbegin() const;  
const_iterator cend() const;  
const_iterator begin() const;  
const_iterator end() const;
```

...

```
iterator begin();  
iterator end();
```

Note that the `begin()` and `end()` methods that return regular iterator are not **const** methods. i.e: if we get a container by `const` (`const ref`, ...) we can't use these methods. We have to use the methods that return **const_iterator**

Iterators & Map

Suppose we work with:

```
map<string,int> dictionary;  
map<string,int>::iterator it;  
...  
it = dictionary.begin();
```

What is the type of `*it` ?

Pairs

```
template< typename T1, typename T2>
struct pair {
    typedef T1 first_type;
    typedef T2 second_type;

    T1 first;
    T2 second;

    pair( const T1& x, const T2& y )
        : first(x), second(y)
    {}
};
```

Using map iterator (folder 4)

```
map<string,int> dict;  
...  
  
for( auto i = dict.cbegin();  
    i != dict.cend();  
    ++i )  
{  
    cout << i->first << " "  
        << i->second << "\n";  
}
```

Using map iterator

```
map<string,int> dict;
```

```
...
```

```
for( const auto& val : dict) {
```

```
    cout << val.first << " "
```

```
        << val.second << "\n";
```

```
}
```

Iterators and Assoc. Containers (folder 4)

Additional set of operations:

- `iterator C::find(key_type const& key)`

Return iterator to first element with **key**.

Return `end()` if not found

- `iterator C::lower_bound(key_type const& key)`

Return iterator to first element greater or equal to **key**

- `iterator C::upper_bound(key_type const& key)`

Return iterator to first element greater than **key**