

Map

- We can use `insert()` and `erase()` as we did with `std::set`
- We need to check the return value from `insert()`
- Because the data is stored using `std::pair`, we need to use `make_pair()` for the new element

```
m.insert(make_pair(k, v));
```

- With C++11, we can use brace initialization

```
m.insert( {k, v} );
```

Map Example

```
#include <map>                                     // Header file for std::map

map<string, int> scores;                             // Create an instance of std::map
scores.insert(make_pair("Maybelline", 86));          // Add some elements to it
scores.insert( {"Graham", 78} );

for (auto it: scores) {                             // Use a range-for loop
    cout << it.first << " has a score of " << it.second << endl;
}

for (auto [key,value] : scores) {                   // C++17
    cout << key << " has a score of " << value << endl;
}
```

Map operations

- Unlike list and set, map supports subscripting.
- However, this works differently from vector and arrays. If the element does not already exist, it is created
 - `scores["Graham"];` *// Fetches the value with key "Graham"*
 - `scores["Graham"] = 66;` *// Changes the element with key "Graham"*
 - `scores["Grace"] = 66;` *// Creates a new element with key "Grace"*
- Need to be careful to avoid accidentally overwriting data
- Not recommended!

- As with `std::set`, we can use read-only algorithms with maps
- These take a key and return an iterator to the pair of data items in the element
- The first member of the pair will be the element key and the second member will be the element value

```
auto p = scores.find("Graham");  
if (p != scores.end())  
    cout << "Element with key " << p->first << " has value " << p->second << endl;
```