

Nested Maps

Nested Maps

- We can create a map whose values are another map

- A map with key int and value string

```
map<int, string>
```

- Which is in turn the value of another map with key int

```
map<int, map<int, string>>
```

// Before C++11 we had to leave a gap between the >>

```
map<int, map<int, string> >
```

- We can initialize a vector by putting a list of the initial element values inside braces

```
vector<int> odd_nums = { 1, 3, 5, 7, 9 };
```

- We can initialize a map in a similar way, with a list of pair elements inside braces

```
// Level map - key is position, value is the object at that position
```

```
map<int, string> level_one_map = {  
    { 1, "player" },    // First data pair  
    { 10, "door" }     // Second data pair  
};
```

Nested Maps

- A map which contains another map is a useful data structure
 - Level map has position as key and player/object as value
 - Game map has level number as key and level map as value

// Game map

```
map<int, map<int, string>> game_map = {  
    {1, level_one_map},      // First element is first level number and map  
    {2, level_two_map}      // Second element is second level number and map  
};
```

Using Statement

- A using declaration makes the code easier to read

```
using level_map = map<int, string>;  
// typedef map<int, string> level_map;    // Before C++11  
  
map<int, level_map> game_map = {  
    {1, level_one_map},  
    {2, level_two_map}  
};
```

Iterating over nested map

- Iterating over the game map gives us pairs
 - The first element is the level number
 - The second element is the level map
- We can iterate over the elements in this level map

```
for (auto level: game_map) {  
    for (auto elem: level.second) {  
        cout << elem.first << ", " << elem.second << endl;  
    }  
}
```

Adding Elements to Nested Map

- We can add elements with insert()

```
level_map level_two_map = {                                     // Create another level map
    {5, "player"},
    {10, "monster"}
};
game_map.insert( {2, level_two_map} );                         // Add the level map to the game map

auto level = game_map.find(2);                                  // Find the level 2 data

if (level != game_map.end()) {
    level->second.insert( {3, "magic wand"} );                 // Add another object to the level 2 map
}
```

Removing Elements from Nested Map

- We can remove elements with `erase()`

// Remove element with key 10 from level 2

```
auto level = game_map.find(2);
```

// Get iterator to level 2

```
if (level != game_map.end()) {
```

```
    auto ten = level->second.find(10);
```

// Get element with key 10

```
    if (ten != level->second.end())
```

```
        level->second.erase(ten);
```

```
}
```


Summary

- We can use list initialization to populate a map
- A map can have another map as its value
- using declarations make complex types easier to work with
- We can perform operations on a nested map
 - The outer map works similarly to a single map
 - To access the inner maps, iterate over the outer map and use the second member of the iterator pair
 - We can then use this second member to access the elements of the inner map