## **Emplace Enhancements for Maps and Unordered Maps**

Here we'll work with some of the new methods available in maps.

```
WE'LL COVER THE FOLLOWINGtry_emplace Methodinsert_or_assign Method
```

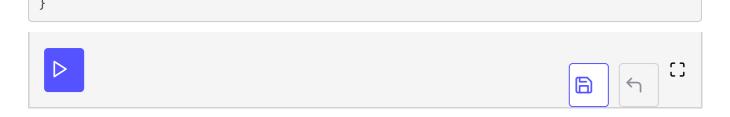
With C++17 you get two new methods for maps and unordered maps:

- try\_emplace() if the object already exists, then it does nothing;
   otherwise, it behaves like emplace().
- emplace() might move from the input parameter when the key is in the
  map, that's why it's best to use find() before such emplacement.
- insert\_or\_assign() gives more information than operator [] as it
  returns if the element was newly inserted or updated and also works
  with types that have no default constructor.

```
try emplace Method #
```

Here's an example:

```
#include <iostream>
#include <string>
#include <map>
int main() {
    std::map<std::string, int> m; m["hello"] = 1;
    m["world"] = 2;
    // C++11 way:
    if (m.find("great") == std::end(m)) m["great"] = 3;
    // the lookup is performed twice if "great" is not in the map
    // C++17 way:
    m.try_emplace("super", 4);
    m.try_emplace("super", 4);
    m.try_emplace("hello", 5); // won't emplace, as it's
    // already in the map
    for (const auto& [key, value] : m)
        std::cout << key << " -> " << value << '\n';</pre>
```



The behaviour of try\_emplace is important in a situation when you move elements into the map:

```
#include <iostream>
#include <string>
#include <map>
int main() {
  std::map<std::string, std::string> m;
  m["Hello"] = "World";
  std::string s = "C++";
  m.emplace(std::make_pair("Hello", std::move(s)));
  // what happens with the string 's'?
  std::cout << s << '\n';
  std::cout << m["Hello"] << '\n';</pre>
  s = "C++";
  m.try_emplace("Hello", std::move(s));
  std::cout << s << '\n';
  std::cout << m["Hello"] << '\n';</pre>
}
```

The code tries to replace ["Hello", "World"] into ["Hello", "C++"].

If you run the example string s after emplace is empty, and the value "World" is not changed into "C++"!

try\_emplace does nothing in the case where the key is already in the container, so the string is unchanged.

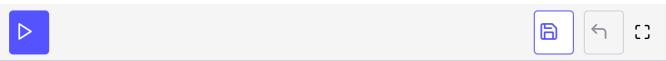
```
insert_or_assign Method #
```

The second function <code>insert\_or\_assign</code>, inserts a new object in the map or assigns the new value. But as opposed to operator [], it also works with non-default constructible types.

For example:

```
#include <iostream>
#include <map>
#include <string>
```

```
struct User {
    std::string name;
    User(std::string s) : name(std::move(s)) {
        std::cout << "User::User(" << name << ")\n";</pre>
    }
    ~User() {
        std::cout << "User::~User(" << name << ")\n";</pre>
    User(const User& u) : name(u.name) {
        std::cout << "User::User(copy, " << name << ")\n";</pre>
    }
    friend bool operator<(const User& u1, const User& u2) {</pre>
        return u1.name < u2.name;
    }
};
int main() {
    std::map<std::string, User> mapNicks;
    //mapNicks["John"] = User("John Doe"); // error: no default ctor for User()
    auto [iter, inserted] = mapNicks.insert_or_assign("John", User("John Doe"));
    if (inserted)
        std::cout << iter->first << " entry was inserted\n";</pre>
    else
        std::cout << iter->first << " entry was updated\n";</pre>
}
```



In the example, above we cannot use the operator [] to insert a new value into the container, as it doesn't support types with non-default constructors. We can do it with the new function.

insert\_or\_assign returns a pair of <iterator, bool>. If the boolean value is true, it means the element was inserted into the container. Otherwise, it was reassigned.

**Extra Info:** See more information in Splicing Maps and Sets P0083 and New emplacement routines N4279.

Now, we'll examine how emplace now returns something.