

## - Exercise

In this exercise, you will use different types of map to solve the problem.

### WE'LL COVER THE FOLLOWING ^

- Task 1
- Task2

## Task 1 #

In the code below, use `std::unordered_set` instead of `std::unordered_map`

## Task2 #

In the code below, use `std::unordered_multiset` instead of `std::unordered_multimap`.

```
#include <iostream> //unorderedMapMultimap.cpp
#include <map>
#include <unordered_map>

int main(){

    std::cout << std::endl;

    long long home= 497074123456;
    long long mobile= 4916046123356;

    // constructor
    std::unordered_multimap<std::string, long long> multiMap{{"grimm", home}, {"grimm", mobile}}
    std::unordered_map<std::string, int> uniqMap{{"bin", 1}, {"root", 0}, {"nobody", 65834}, {"

    // show the unordered maps
    std::cout << "multiMap: ";
    for(auto m : multiMap) std::cout << '{' << m.first << ", " << m.second << '}'

    std::cout << std::endl;

    std::cout << "uniqMap: ";
    for(auto u : uniqMap) std::cout << '{' << u.first << ", " << u.second << '}'
    std::cout << std::endl;
```

```

std::cout << std::endl;

// insert elements
long long work= 4970719754513;

multiMap.insert({"grimm", work});
// will not work
// multiMap["grimm-jaud"]=4916012323356;

uniqMap["lp"]=4;
uniqMap.insert({"sshd", 71});

std::map<std::string, int> myMap{{"ftp", 40}, {"rainer", 999}};
uniqMap.insert(myMap.begin(), myMap.end());

// show the unordered maps
std::cout << "multiMap: ";
for(auto m : multiMap) std::cout << '{' << m.first << ", " << m.second << '}'

std::cout << std::endl;

std::cout << "uniqMap: ";
for(auto u : uniqMap) std::cout << '{' << u.first << ", " << u.second << '}'
std::cout << std::endl;

std::cout << std::endl;
// search for elements

// only grimm
auto iter= multiMap.equal_range("grimm");
std::cout << "grimm: ";
for(auto itVal= iter.first; itVal !=iter.second;++itVal){
    std::cout << itVal->second << " ";
}

std::cout << std::endl;

std::cout << "multiMap.count(grimm): " << multiMap.count("grimm") << std::endl;

auto it= uniqMap.find("root");
if ( it != uniqMap.end()){
    std::cout << "uniqMap.find(root): " << it->second << std::endl;
    std::cout << "uniqMap[root]: " << uniqMap["root"] << std::endl;
}

// will create a new entry
std::cout << "uniqMap[notAvailable]: " << uniqMap["notAvailable"] << std::endl;

std::cout << std::endl;

// remove
int numMulti= multiMap.erase("grimm");
int numUniq= uniqMap.erase("rainer");

std::cout << "Erased " << numMulti << " times grimm from multiMap." << std::endl;
std::cout << "Erased " << numUniq << " times rainer from uniqMap." << std::endl;

// all
multiMap.clear();
uniqMap.clear();

```

```
std::cout << std::endl;

std::cout << "multiMap.size(): " << multiMap.size() << std::endl;
std::cout << "uniqMap.size(): " << uniqMap.size() << std::endl;

std::cout << std::endl;

}
```



---

You will find the solutions to these tasks in the following lesson.