- Solution

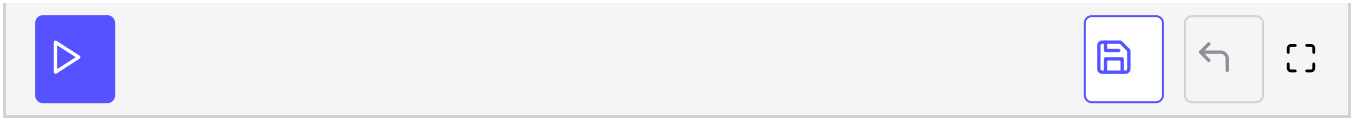
The solution to the tasks in the associative containers exercise is explained in this lesson.

WE'LL COVER THE FOLLOWING

Solution

```
// unorderedSetMultiset.cpp
#include <iostream>
#include <set>
#include <unordered_set>

int main(){

  std::cout << std::endl;

  // constructor
  std::unordered_multiset<int> multiSet{1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1};
  std::unordered_set<int> uniqSet(multiSet.begin(), multiSet.end());

  // show the difference
  std::cout << "multiSet: ";
  for(auto m : multiSet) std::cout << m << " ";

  std::cout << std::endl;

  std::cout << "uniqSet: ";
  for(auto s : uniqSet) std::cout << s << " ";

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

  // insert elements
  multiSet.insert(-1000);
  uniqSet.insert(-1000);

  std::set<int> mySet{-5, -4, -3, -2, -1};
  multiSet.insert(mySet.begin(), mySet.end());
  uniqSet.insert(mySet.begin(), mySet.end());

  // show the difference
  std::cout << "multiSet: ";
  for(auto m : multiSet) std::cout << m << " ";
```

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

  std::cout << "uniqSet: ";
  for(auto s : uniqSet) std::cout << s << " ";

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

  // search for elements
  auto it = uniqSet.find(5);
  if (it != uniqSet.end()){
    std::cout << "uniqSet.find(5): " << *it << std::endl;
  }

  std::cout << "multiSet.count(5): " << multiSet.count(5) << std::endl;

  std::cout << std::endl;

  // remove
  int numMulti = multiSet.erase(5);
  int numUniq = uniqSet.erase(5);

  std::cout << "Erased " << numMulti << " times 5 from multiSet." << std::endl;
  std::cout << "Erased " << numUniq << " times 5 from uniqSet." << std::endl;

  // all
  multiSet.clear();
  uniqSet.clear();

  std::cout << std::endl;

  std::cout << "multiSet.size(): " << multiSet.size() << std::endl;
  std::cout << "uniqSet.size(): " << uniqSet.size() << std::endl;

  std::cout << std::endl;

}
```



Explanation

- In lines 11-12, we defined an `std::unordered_multiset` named `multiSet` and an `std::unordered_set` named `uniqSet`. `std::unordered_multiset` can have multiples instances of the same value, but `std::unordered_set` can have only one instance of one specific value.
- Values in both of these sets are arranged according to their hash values, making it difficult to find any particular order in it.
- In lines 16 and 21, we print the values for both sets, and you can clearly see the differences in the values present in both sets.

- In lines 26-27, we add -1000 to both `multiSet` and `uniqSet` using the built-in function `insert()`.
- In line 29, we defined a `std::set<int> mySet` and then inserted the values in the set to the unordered sets `multiSet` and `uniqSet`. There is no particular order in which the values are present in these sets. The only difference is that the values can be repeated in `std::unordered_multiset` but not in `std::unordered_set`.
- In line 45, `uniqSet.find(5)` returns the pointer to the value 5 in the `uniqSet`, and we have used `auto` for its type deduction.
- In line 50, `multiSet.count(5)` returns the number of instances of value 5 in `multiSet`.
- In lines 55 and 56, we erased all the instances of 5 in `multiSet` and `uniqSet`.
- 5 is erased twice in `multiSet` and once in `uniqSet`.

In the next lesson, we will introduce you to templates in modern C++.