Associative Containers

std:map

```
std::map<KeyT, ValueT> m{{key1, value1}, {..}};
```

- Check size: m.size()
- Add item to map: m.emplace(key, value);
- Modify or add item: m[key] = value;
- Check if key presentt: m.count(key) > 0; From C++20, m.contains(key);

```
#include <iostream>
#include <map>
using namespace std;

int main() {
  using StudentList = std::map<int, string>;
  StudentList cpp_students;

  // Inserting data in the students dictionary
  cpp_students.emplace(1509, "Nacho");    // [1]
  cpp_students.emplace(1040, "Pepe");     // [0]
  cpp_students.emplace(8820, "Marcelo");  // [2]

  for (const auto& [id, name] : cpp_students) {
    cout << "id: " << id << ", " << name << endl;
  }

  return 0;
}
```

std::unordered_map

- Implemented as hash table.
- Faster than map because keys are saved in random order.

```cpp
#include <iostream>
#include <unordered_map>
using namespace std;

int main() {
  using StudentList = std::unordered_map<int, string>;
  StudentList cpp_students;

  // Inserting data in the students dictionary
  cpp_students.emplace(1509, "Nacho");    // [2]
  cpp_students.emplace(1040, "Pepe");     // [1]
  cpp_students.emplace(8820, "Marcelo");  // [0]

  for (const auto& [id, name] : cpp_students) {
    cout << "id: " << id << ", " << name << endl;
  }

  return 0;
}
```

Iterating over maps

```cpp
for (const auto& kv : m) {
  const auto& key = kv.first;
  const auto& value = kv.second;
  // Do important work.
}
```

New in C++17

```cpp
std::map<char, int> my_dict{{'a', 27}, {'b', 3}};
for (const auto& [key, value] : my_dict) {
  cout << key << " has value " << value << endl;
```