Performance

The unordered associative container type is more optimized compared to its ordered sibling.

Performance - that's the simple reason - why the unordered associative containers were so long missed in C++. In the example below, one million randomly created values are read from a 10 million big `std::map` and `std::unordered_map`. The impressive result is that the linear access time of an unordered associative container is 20 times faster than the access time of an ordered associative container. That is just the difference between constant and logarithmic complexity *O(log n)* of these operations.

> **Note:** The given code might take more time to execute than normal.

```cpp
#include <chrono>
#include <iostream>
#include <map>
#include <random>
#include <unordered_map>

static const long long mapSize= 10000000;
static const long long accSize= 1000000;

int main(){

  std::cout << std::endl;

  std::map<int,int> myMap;
  std::unordered_map<int,int> myHash;

  for ( long long i=0; i < mapSize; ++i ){
    myMap[i]=i;
    myHash[i]= i;
  }

  std::vector<int> randValues;
  randValues.reserve(accSize);

  // random values
  std::random_device seed;
  std::mt19937 engine(seed());
  std::uniform_int_distribution<> uniformDist(0,mapSize);
  for ( long long i=0 ; i< accSize ; ++i) randValues.push_back(uniformDist(engine));
```

```cpp
  auto start = std::chrono::system_clock::now();
  for ( long long i=0; i < accSize; ++i){

    myMap[randValues[i]];
  }
  std::chrono::duration<double> dur= std::chrono::system_clock::now() - start;
  std::cout << "time for std::map: " << dur.count() << " seconds" << std::endl;

  auto start2 = std::chrono::system_clock::now();
  for ( long long i=0; i < accSize; ++i){
    myHash[randValues[i]];
  }
  std::chrono::duration<double> dur2= std::chrono::system_clock::now() - start2;
  std::cout << "time for std::unordered_map: " << dur2.count() << " seconds" << std::endl;

  std::cout << std::endl;

}
```

Performance Comparison