Associative Containers: Performance Comparison

Modern C++ has eight associative containers, but we should use std::map and std::unordered_map. Why? Let's disucss in this lesson.

WE'LL COVER THE FOLLOWING

In 95% of our use-cases, we use `std::map` or `std::unordered_map`. In fewer cases, we do not need the value associated with the key. Before we begin this lesson and give an overview of numbers to both associative containers, here is one simple rule to remember:

> 💡 If we want to have a container with ordered key/value associations, use `std::map`; if not, use a `std::unordered_map`.

A Phone Book

The eight variations are like different versions of a phone book. What is a phone book? A phone book is a sequence of key/value pairs. We use the keys (family names) to get the values (phone numbers).

The family names of a phone book can be ordered or unordered. The phone book can have a phone number associated with the family name or not. It can either have only one family name or multiple identical family names. If we want to store our mobile number and our landline number in a phone book, we are using two identical keys.

The reason for this lesson is not to explain the associative containers. Rather, we want to discuss how the access time to an ordered associative container is

logarithmic, but the access time to an unordered associative container is *amortized constant*.

Performance of an std::map and an std::unordered_map

What does amortized constant access time mean for an unordered associative container, such as `std::unordered_map`? It means that our query for a phone number is independent of the size of the phone book. Let's look at a performance test.

We have a phone book with roughly 89,000 entries. We will increase its size successively by ten until it has approximately 89,000,000 entries. After each step, we will ask for every phone number, meaning that we will randomly use all family names.

The file below shows us a section of the initial phone book. We can see the name/number pairs separated by a colon and the name separated from the number by a comma.

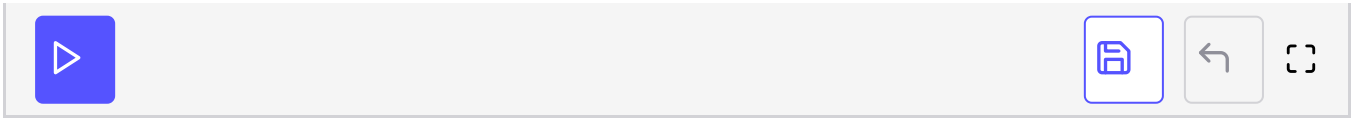
telebook.txt

Map

```
// telephoneBook.cpp

#include <chrono>
#include <fstream>
#include <iostream>
#include <map>
#include <random>
#include <regex>
#include <sstream>
#include <string>
#include <unordered_map>
#include <vector>

using map = std::map<std::string, int>;                    // (1)

std::ifstream openFile(const std::string& myFile){

  std::ifstream file(myFile, std::ios::in);
  if ( !file ){
```

```
    std::cerr << "Can't open file "+ myFile + "!" << std::endl;
    exit(EXIT_FAILURE);
  }

  return file;

}

std::string readFile(std::ifstream file){

    std::stringstream buffer;
    buffer << file.rdbuf();

    return buffer.str();

}

map createTeleBook(const std::string& fileCont){

    map teleBook;

    std::regex regColon(":");

    std::sregex_token_iterator fileContIt(fileCont.begin(), fileCont.end(), regColon, -1);
    const std::sregex_token_iterator fileContEndIt;

    std::string entry;
    std::string key;
    int value;
    while (fileContIt != fileContEndIt){                                    // (2)
        entry = *fileContIt++;
        auto comma = entry.find(",");                                       // (3)
        key = entry.substr(0, comma);
        value = std::stoi(entry.substr(comma + 1, entry.length() -1));
        teleBook[key] = value;                                              // (4)
    }
    return teleBook;

}

std::vector<std::string> getRandomNames(const map& teleBook){

    std::vector<std::string> allNames;
    for (const auto& pair: teleBook) allNames.push_back(pair.first);   // (5)

    std::random_device randDev;
    std::mt19937 generator(randDev());

    std::shuffle(allNames.begin(), allNames.end(), generator);         // (6)

    return allNames;
}

void measurePerformance(const std::vector<std::string>& names, map& m){

    auto start = std::chrono::steady_clock::now();
    for (const auto& name: names) m[name];                              // (7)
    std::chrono::duration<double> dur= std::chrono::steady_clock::now() - start;
    std::cout << "Access time: " << dur.count() << " seconds" << std::endl;

}

int main(){
```

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

    // get the filename
    std::string myFile{"telebook.txt"};

    std::ifstream file = openFile(myFile);

    std::string fileContent = readFile(std::move(file));

    map teleBook = createTeleBook(fileContent);

    std::cout << "teleBook.size(): " <<  teleBook.size() << std::endl;

    std::vector<std::string> randomNames = getRandomNames(teleBook);
    std::cout << "Map -> ";
    measurePerformance(randomNames, teleBook);

    std::cout << std::endl;

}
```

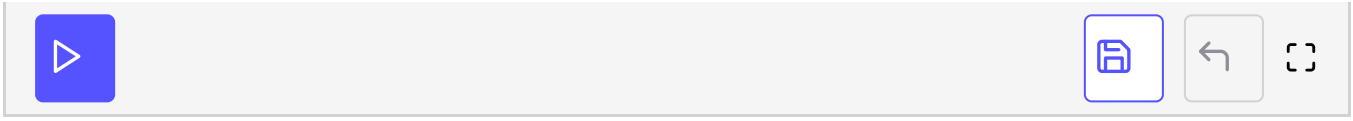


Unordered Map

```
// telephoneBook.cpp

#include <chrono>
#include <fstream>
#include <iostream>
#include <map>
#include <random>
#include <regex>
#include <sstream>
#include <string>
#include <unordered_map>
#include <vector>

using map = std::unordered_map<std::string, int>;                    // (1)

std::ifstream openFile(const std::string& myFile){

  std::ifstream file(myFile, std::ios::in);
  if ( !file ){
    std::cerr << "Can't open file "+ myFile + "!" << std::endl;
    exit(EXIT_FAILURE);
  }
  return file;

}

std::string readFile(std::ifstream file){

    std::stringstream buffer;
    buffer << file.rdbuf();
```

```
    return buffer.str();

}


map createTeleBook(const std::string& fileCont){

    map teleBook;

    std::regex regColon(":");

    std::sregex_token_iterator fileContIt(fileCont.begin(), fileCont.end(), regColon, -1);
    const std::sregex_token_iterator fileContEndIt;

    std::string entry;
    std::string key;
    int value;
    while (fileContIt != fileContEndIt){                                // (2)
        entry = *fileContIt++;
        auto comma = entry.find(",");                                   // (3)
        key = entry.substr(0, comma);
        value = std::stoi(entry.substr(comma + 1, entry.length() -1));
        teleBook[key] = value;                                          // (4)
    }
    return teleBook;

}

std::vector<std::string> getRandomNames(const map& teleBook){

    std::vector<std::string> allNames;
    for (const auto& pair: teleBook) allNames.push_back(pair.first);   // (5)

    std::random_device randDev;
    std::mt19937 generator(randDev());

    std::shuffle(allNames.begin(), allNames.end(), generator);         // (6)

    return allNames;
}

void measurePerformance(const std::vector<std::string>& names, map& m){

    auto start = std::chrono::steady_clock::now();
    for (const auto& name: names) m[name];                              // (7)
    std::chrono::duration<double> dur= std::chrono::steady_clock::now() - start;
    std::cout << "Access time: " << dur.count() << " seconds" << std::endl;

}

int main(){

    std::cout << std::endl;

    // get the filename
    std::string myFile{"telebook.txt"};

    std::ifstream file = openFile(myFile);

    std::string fileContent = readFile(std::move(file));

    map teleBook = createTeleBook(fileContent);
```

```
    std::cout << "teleBook.size(): " <<  teleBook.size() << std::endl;

    std::vector<std::string> randomNames = getRandomNames(teleBook);
    std::cout << "Unordered Map -> ";
    measurePerformance(randomNames, teleBook);

    std::cout << std::endl;

}
```



Explanation

Let's start in the main program. There are several steps. First, we open the file, (`telebook.txt`) and read the content. Then, we create a phone book (`std::map` and `std::unordered_map`), get an arbitrary permutation of the family names, and finally make the performance test. We used the code tabs to compare the performance of `std::map` and `std::unordered_map`.

Line 14 is interesting to note. `std::unordered_map` supports a superset of the interface of a `std::map`. This makes it easier for us to make our performance test. This was accomplished by first using

```
map = std::unordered_map<std::string, int>;
```

We then changed the line in the second code tab by using

```
map = std::map<std::string, int>;
```

The following functions are also interesting to note:

- `createTeleBook`
  - the while loop iterates over all name/number tokens, created by the regular expression `regColon` (line 48)
  - each token is separated by the comma (line 50)
  - in the end, the name/number pair is added to the phone book (line 53)
- `getRandomNames`
  - puts all names onto a vector (line 62)
  - shuffles the names (line 67)

- `measurePerformance`
    - asks for each name in the phone book (line 75)

Run both the codes and see how the access time in `std::unordered_map` is faster than in `std::map`.

Let's take a look at the examples of associative containers in the next lesson.