Associative Containers

In this lesson, we will learn about ordered and unordered associative containers.

WE'LL COVER THE FOLLOWING

A Simple Performance Comparison

Before we take a deeper look into associative containers, let's examine the interface of the hash tables, which are called unordered associative containers. We will first compare the performance of the associative containers.

Admittedly, the eight variations of associative containers can be confusing. In addition, the unordered name component of the new variations is not easy to read or write. On the one hand, the unordered associative containers were too late for the C++98 standard. On the other hand, they were missed so badly that most of the architectures implemented them by themselves. Therefore, the simple names have already been used, and the C++ standardization committee must use more elaborate names, that follow a simple pattern.

The table below shows the entire system. It includes the access time for the ordered containers (logarithmic) and the unordered container (amortized constant).

We can also look at the table from a different perspective. A `std::unordered_set` supports the interface of a `std::set`. The same holds for the pairs `std::unordered_map` and `std::map`, `std::unordered_multiset` as wel as `std::multiset`, `std::unordered_multimap` and `std::multimap` as well. This means, if an `std::map` is too slow, we can quite easily use an `std::unordered_map`.

Associative Container	Keys sorted	Value available	Multiple identical keys	Access time	Standard
`std::set`	yes	no	no	logarithmic	C++98
`std::unordered_set`	no	no	no	constant	C++11
`std::map`	yes	yes	no	logarithmic	C++98
`std::unordered_map`	no	yes	no	constant	C++11
`std::multiset`	yes	no	yes	logarithmic	C++98
`std::unordered_multiset`	no	no	yes	constant	C++11
`std::multimap`	yes	yes	yes	logarithmic	C++98
`std::unordered_multimap`	no	yes	yes	constant	C++11

The Eight Variations

To get an ordering of the eight variations, we start with the classical ordered associative containers. We can easily apply the ordering to the unordered associative containers.

To get the data into ordered associative containers, we must answer two questions regarding the keys:

1. Does the key have an associated value?
2. Are several identical keys possible?

By answering these two questions, we get the four different ordered associative containers: `std::set`, `std::multiset`, `std::map`, and `std::multimap`. We can apply the two questions to the unordered associative containers, meaning that we get the containers `std::unordered_set`, `std::unordered_multiset`, `std::unordered_map`, and `std::unordered_multimap`.

If the name of the container has the component

- **map**, it has an associated value.
- **multi**, it can have more than one identical key.
- **unordered**, its keys are not sorted.

If two container names only differ in the name component **unordered**, they will have a similar interface. The container without the name component unordered supports an interface that is a subset of the unordered container.

Performance Matters

- The access time of the
  - ordered associative containers is **logarithmic**.
  - unordered associative containers is **amortized constant**.

Ordered Associative Containers

The small difference between ordered and unordered associative containers is that the keys of the classical associative containers are ordered. By default, the smaller relation (<) is used. Therefore, the containers are sorted in increasing order.

This ordering generates the following consequences for the ordered associative containers.

- The key has to support an ordering relation.
- Associative containers are typically implemented in balanced binary search trees.
- The access time to the key and to the value is logarithmic.

The most commonly used ordered associative container is `std::map`.

```
map<int, string> int2String{ {3,"three"},{2,"two"},{1,"one"},{5,"five"},{6
,"six"},{4,"four"},{7,"seven"} };
```

Unordered Associative Containers

The key idea of the unordered associative containers is that the **key** is mapped with the help of the hash function onto the bucket. You can find the **key/value pair** in the bucket.

We must introduce a few terms before we describe the characteristics of unordered associative containers.

Hash value: The value you get if you apply the hash function onto the key.

Collision: If different keys are mapped to the same hash value, we will have a collision. The unordered associative containers have to deal with this situation.

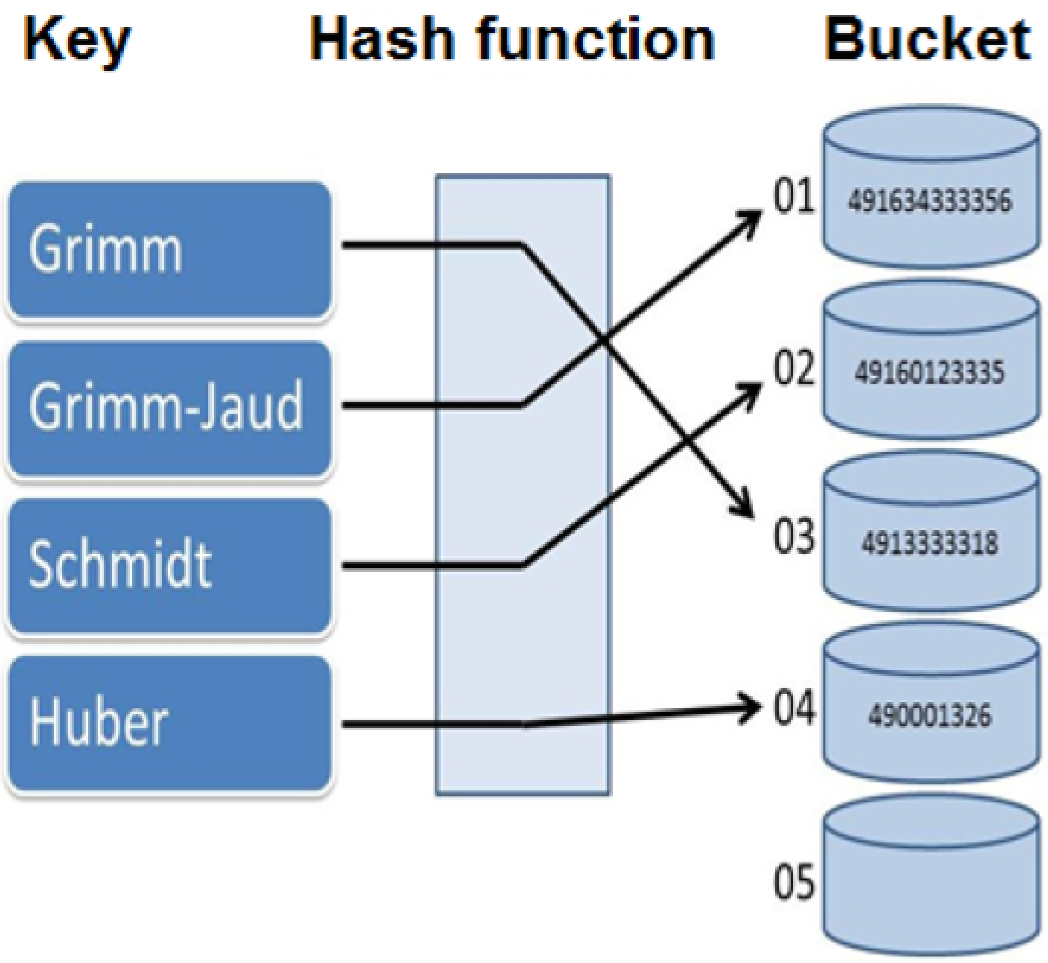
The usage of the hash function has very notable consequences for the unordered associative container.

- The keys must support an equal comparison in order to deal with collisions.
- The hash value of a key must be available.
- The execution of a hash function is a constant. Therefore, the access time to the keys of an unordered associative container is constant. For simplicity, we ignored collisions.

`std::map` the most commonly used ordered associative container `std::unordered_map` is the most commonly used unordered associative container.

```
std::unordered_map<std::string, int> {{"Grimm", 4916343333},
    {"Grimm-yaud", 491601233}, {"Schmidt", 49133318}, {"Huber", 4900013}};
```

The graphic shows the mapping of the keys to their bucket by using the hash function.



In the next lesson, we will learn about the functionality of hash functions.