

The Hash Function

Here, we will understand why hash functions are essential in unordered associative pairs.

The reason for the constant access time of the unordered associative container is the hash function. The hash function maps the key to its value the so-called has value. A hash function is good if it produces as few collisions as possible and equally distributes the keys onto the buckets. Because the execution of the hash function takes a constant amount of time, the access of the elements is in the base case also constant.

The hash function

- is already defined for the built-in types like boolean, natural numbers and floating point numbers,
- is available for `std::string` and `std::wstring`,
- generates for a C string `const char` a hash value of the pointer address,
- can be defined for *user-defined* data types.

For *user-defined* types which are used as a key for an unordered associative container, you have to keep two requirements to keep in mind. They need a hash function and have to be comparable to equal.

```
// unorderedMapHash.cpp
#include <iostream>
#include <unordered_map>

struct MyInt{
    MyInt(int v):val(v){}
    bool operator== (const MyInt& other) const {
        return val == other.val;
    }
    int val;
};

struct MyHash{
    std::size_t operator()(MyInt m) const {
        std::hash<int> hashVal;
        return hashVal(m.val);
    }
}
```



```

};

std::ostream& operator << (std::ostream& st, const MyInt& myIn){
    st << myIn.val ;
    return st;
}

int main(){
typedef std::unordered_map<MyInt, int, MyHash> MyIntMap;
MyIntMap myMap{{MyInt(-2), -2}, {MyInt(-1), -1}, {MyInt(0), 0}, {MyInt(1), 1}};

for(auto m : myMap) std::cout << "{" << m.first << "," << m.second << " } ";
    // {MyInt(1),1} {MyInt(0),0} {MyInt(-1),-1} {MyInt(-2),-2}

std::cout << myMap[MyInt(-2)] << std::endl;    // -2
return 0;
}

```



A custom hash function