Конструирование программного обеспечения

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

Лекция 11

Функциональные объекты, cv-квалификаторы; контейнеры, ч. 2

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
НИЖНИЙ НОВГОРОД

Функциональные объекты

Уже знаем, что функции в C++ являются «гражданами первого сорта» - имена функции являются одновременно идентификаторами со смыслом «указателя на функцию», их можно присваивать и передавать в другие функции.

```
int my_func(int foo, double bar, char* baz, float& bav) {return 0;}
int my_func2(int foo, double bar, char* baz, float& bav) {return 2 * foo;}
using MyFuncPtr = int(*)(int, double, char*, float&);

void apply_fn(MyFuncPtr fn) {
    float val = 300.0;
    fn(42, 1337.0, nullptr, val);
}

int main() {
    MyFuncPtr func_ptr;
    func_ptr = my_func;
    float val = -1.0;
    int retval = func_ptr(0, 1, nullptr, val);
    apply_fn(func_ptr);
}
```

Функциональные объекты

Помимо указателя на функцию и конструктора класса, в выражении типа *идентификатор* (*аргументы*) могут также участвовать **функциональные объекты** (функторы) - объекты класса, который переопределяет `operator()`

```
class MyFunctor {
public:
    int operator()(int foo, double bar, char* baz, float& bav) {
        return 0;
    }
};
```

```
int main() {
    MyFunctor functor;
    float val = -1.0;
    int retval = functor(0, 1, nullptr, val);
}
```

Функциональные объекты

Помимо указателя на функцию и конструктора класса, в выражении типа *идентификатор* (*аргументы*) могут также участвовать **функциональные объекты** (функторы) - объекты класса, который переопределяет `operator()`

```
class MyFunctor {
public:
    int operator()(int foo, double bar, char* baz, float& bav) {
        return 0;
    }
};
void apply_functor(MyFunctor fn) {
    float val = 300.0;
    fn(42, 1337.0, nullptr, val);
}

int main() {
    MyFunctor functor;
    apply_functor(functor);
}
```

Функциональные объекты

Часто для функциональных объектов используют **struct** вместо **class** - разница лишь в том, что для struct не обязательно указывать **public:**

```
struct MyFunctor {
    int operator()(int foo, double bar, char* baz, float& bav) {
        return 0;
    }
};
void apply_functor(MyFunctor fn) {
    float val = 300.0;
    fn(42, 1337.0, nullptr, val);
}

int main() {
    MyFunctor functor;
    apply_functor(functor);
}
```

cv-квалификаторы

Наименование cv-qualifiers в стандарте используется для обобщения квалификаторов типов `const` и `volatile`:

- `const` - соответствующий идентификатор нельзя модифицировать после создания
- `volatile` - доступы к соответствующему идентификатору запрещено оптимизировать или кэшировать компилятору

```
int main() {
    int n1 = 0;           // non-const object
    const int n2 = 0;     // const object
    int const n3 = 0;     // const object (same as n2)
    volatile int n4 = 0;  // volatile object

    n1 = 1; // ok, modifiable object
//  n2 = 2; // error: non-modifiable object
    n4 = 3; // ok, treated as a side-effect
    const int& r1 = n1; // reference to const bound to non-const object
//  r1 = 2; // error: attempt to modify through reference to const
   const int& r2 = n2; // reference to const bound to const object
}
```

cv-квалификаторы

Наименование cv-qualifiers в стандарте используется для обобщения квалификаторов типов `const` и `volatile`:

- `const` - соответствующий идентификатор нельзя модифицировать после создания
- `volatile` - доступы к соответствующему идентификатору запрещено оптимизировать или кэшировать компилятору

```
int main() {
    int n1 = 0;
    const int n2 = 0;
    int const n3 = 0;
    volatile int n4 = 0;

    n1 = 1; // ok, modifiable object
//  n2 = 2; // error: non-modifiable object
    n4 = 3; // ok, treated as a side-effect
    const int& r1 = n1; // reference to const bound to non-const object
//  r1 = 2; // error: attempt to modify through reference to const
   const int& r2 = n2; // reference to const bound to const object
}
```

Скомпилированная программа в ассемблер-представлении

```
main:
    movl    $0, -4(%rsp) # volatile int n4 = 0;
    movl    $3, -4(%rsp) # n4 = 3;
    xorl    %eax, %eax   # return 0 (implicit)
    ret
```

cv-квалификаторы и функции

```
using MyFuncPtr = int (*)(int);

int wrong_fn(int& arg) {return 0;}
int ok_fn(const int arg) {return 0;}
int cv_arg_ok_fn(const volatile int arg) {return 0;}
const int another_wrong_fn(int arg) {return 0;}
const volatile int yet_another_wrong_fn(int arg) {return 0;}

int main()
{
    MyFuncPtr func_ptr;
    // func_ptr = wrong_fn; // error C2440: '=': cannot convert from ...
    func_ptr = ok_fn;
    func_ptr = cv_arg_ok_fn;
    // func_ptr = another_wrong_fn; // error C2440: '=': cannot convert from ...
    // func_ptr = yet_another_wrong_fn; // error C2440: '=': cannot convert from ...
}
```

cv-qualifiers в аргументах функции не являются частью ее типа...

... также могут быть указаны для возвращаемого значения, но это не имеет какого-то осмысленного эффекта

cv-квалификаторы и функции

```
using MyFuncPtr = int (*)(int);

int ok_fn(const int arg) {return 0;}
int ok_fn(int arg) {return 0;} // error C2084: function 'int ok_fn(const int)’
                               // already has a body
int ok_fn(volatile int arg) {return 0;} // error C2084: ...

int main()
{
    MyFuncPtr func_ptr;
    func_ptr = ok_fn; // error C2568: '=': unable to resolve function overload
}
```

Поскольку cv-qualifiers в аргументах функции не являются частью ее типа, нельзя перегружать функцию, поменяв лишь cv-квалификатор ее аргумента(-ов).

cv-квалификаторы и методы

```
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() {
        return std::atan(y / x);
    }
    void rotate(double angle) {
        double tmp_x = x * std::cos(angle)
                     + y * std::sin(angle);
        double tmp_y = -x * std::sin(angle)
                      + y * std::cos(angle);
        x = tmp_x; y = tmp_y;
    }
};
```

```
Point2D pt_2d = {0.0, 1.0};
std::cout << pt_2d.get_angle() << '\n';
const double M_PI_2 = 1.57079632679489;
pt_2d.rotate(M_PI_2);
std::cout << pt_2d.get_angle() << '\n';
```

```
1.5708
6.12323e-17
```

cv-квалификаторы и методы

```cpp
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() const {
        return std::atan(y / x);
    }
    void rotate(double angle) const {
        double tmp_x = x * std::cos(angle)
                     + y * std::sin(angle);
        double tmp_y = -x * std::sin(angle)
                      + y * std::cos(angle);
        x = tmp_x; y = tmp_y;
    }
};
```

```cpp
Point2D pt_2d = {0.0, 1.0};
std::cout << pt_2d.get_angle() << '\n';
const double M_PI_2 = 1.57079632679489;
pt_2d.rotate(M_PI_2);
std::cout << pt_2d.get_angle() << '\n';
```

error C3490: 'x' cannot be modified because it is being accessed through a const object

cv-квалификаторы и методы

```
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() const {
        return std::atan(y / x);
    }
    void rotate(double angle) {
        double tmp_x = x * std::cos(angle)
                     + y * std::sin(angle);
        double tmp_y = -x * std::sin(angle)
                      + y * std::cos(angle);
        x = tmp_x; y = tmp_y;
    }
};
```

```
const Point2D pt_2d = {0.0, 1.0};
std::cout << pt_2d.get_angle() << '\n';
const double M_PI_2 = 1.57079632679489;
pt_2d.rotate(M_PI_2);
```

```
error C2662: 'void
Point2D::rotate(double)': cannot convert
'this' pointer from ‘const Point2D' to
'Point2D &'
note: Conversion loses qualifiers
```

cv-квалификаторы и методы

```
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() const {
        return std::atan(y / x);
    }
    void rotate(double angle) {
        double tmp_x = x * std::cos(angle)
                     + y * std::sin(angle);
        double tmp_y = -x * std::sin(angle)
                      + y * std::cos(angle);
        x = tmp_x; y = tmp_y;
    }
};
```

```
volatile Point2D pt_2d = {0.0, 1.0};
std::cout << pt_2d.get_angle() << '\n';
```

```
error C2662: 'double
Point2D::get_angle(void) const': cannot
convert 'this' pointer from 'volatile
Point2D' to 'const Point2D &'
note: Conversion loses qualifiers
```

cv-квалификаторы и методы

```cpp
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() const volatile {
        return std::atan(y / x);
    }
    void rotate(double angle) {
        double tmp_x = x * std::cos(angle)
                     + y * std::sin(angle);
        double tmp_y = -x * std::sin(angle)
                      + y * std::cos(angle);
        x = tmp_x; y = tmp_y;
    }
};
```

```cpp
volatile Point2D pt_2d = {0.0, 1.0};
std::cout << pt_2d.get_angle() << '\n';
```

OK

```
1.5708
```

cv-квалификаторы и перегрузка методов

```cpp
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() const {
        std::cout << "Const" << '\n';
        return std::atan(y / x);
    }
    double get_angle() {
        std::cout << "Non-const" << '\n';
        return std::atan(y / x);
    }
};
```

```cpp
Point2D pt_2d = {0.0, 1.0};
std::cout << pt_2d.get_angle() << '\n';
const Point2D pt_2d_c = {-1.0, -1.0};
std::cout << pt_2d_c.get_angle() << '\n';
```

```
Non-const
1.5708
Const
0.785398
```

Указание cv-qualifiers для **методов** так, как указано выше, делает соответствующие квалификаторы частью типа метода; методы можно перегружать, меняя лишь cv-qualifier метода, без изменения аргументов и возвращаемых значений. Разрешение перегрузки происходит согласно cv-квалификаторам объекта, метод у которого вызывается.

const-корректность

```
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() {
        return std::atan(y / x);
    }
};
```

```
void print_angle(Point2D& pt) {
    std::cout << pt.get_angle();
}

int main()
{
    Point2D pt_2d = {0.0, 1.0};
    print_angle(pt_2d);
    return 0;
}
```

const-корректность заключается в своевременном и полном указании квалификатора const для аргументов, которым не следует по смыслу меняться в функции, или методов, которые по смыслу не должны менять состояние текущего объекта.

const-корректность

```
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() {
        return std::atan(y / x);
    }
};
```

```
void print_angle(Point2D& pt) {
    std::cout << pt.get_angle();
}

int main()
{
    const Point2D pt_2d = {0.0, 1.0};
    print_angle(pt_2d);
    return 0;
}
```

```
error C2664: 'void print_angle(Point2D &)':
cannot convert argument 1 from 'const
Point2D' to 'Point2D &'
note: Conversion loses qualifiers
```

const-корректность

```cpp
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() {
        return std::atan(y / x);
    }
};
```

```cpp
void print_angle(const Point2D& pt) {
    std::cout << pt.get_angle();
}

int main()
{
    const Point2D pt_2d = {0.0, 1.0};
    print_angle(pt_2d);
    return 0;
}
```

const-корректность

```
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() {
        return std::atan(y / x);
    }
};
```

```
void print_angle(const Point2D& pt) {
    std::cout << pt.get_angle();
}

int main()
{
    const Point2D pt_2d = {0.0, 1.0};
    print_angle(pt_2d);
    return 0;
}
```

```
error C2662: 'double
Point2D::get_angle(void)': cannot
convert 'this' pointer from 'const
Point2D' to 'Point2D &’
note: Conversion loses qualifiers
```

const-корректность

```cpp
#include <iostream>
#include <cmath>

class Point2D {
public:
    double x, y;
    double get_angle() const {
        return std::atan(y / x);
    }
};
```

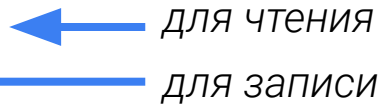
```cpp
void print_angle(const Point2D& pt) {
    std::cout << pt.get_angle();
}

int main()
{
    const Point2D pt_2d = {0.0, 1.0};
    print_angle(pt_2d);
    return 0;
}
```

OK

const-корректность

```
template<class T>
class DynArray {
    /* ... */
    const T& operator[](size_t idx) const;
    T& operator[](size_t idx);
    /* ... */
}
```



std::unordered_map для пользовательских типов

```
#include <unordered_map>
#include <string>
#include <iostream>

class RGBColor {
    public: uint8_t r, g, b;
};
using ColorNameMap = std::unordered_map<RGBColor, std::string>;
using ColorNameMapEntry = std::pair<RGBColor, std::string>;

int main() {
    ColorNameMap color_map = {
        {{255, 0, 0}, "Red"},
        {{0, 255, 0}, "Green"},
        {{255, 255, 255}, "White"}};

    RGBColor color = {255, 0, 0};
    std::cout << color_map[color];
}
```

Compile-time error:

error C2280: 'std::hash<_Kty>::hash(const std::hash<_Kty> &)': attempting to reference a deleted function

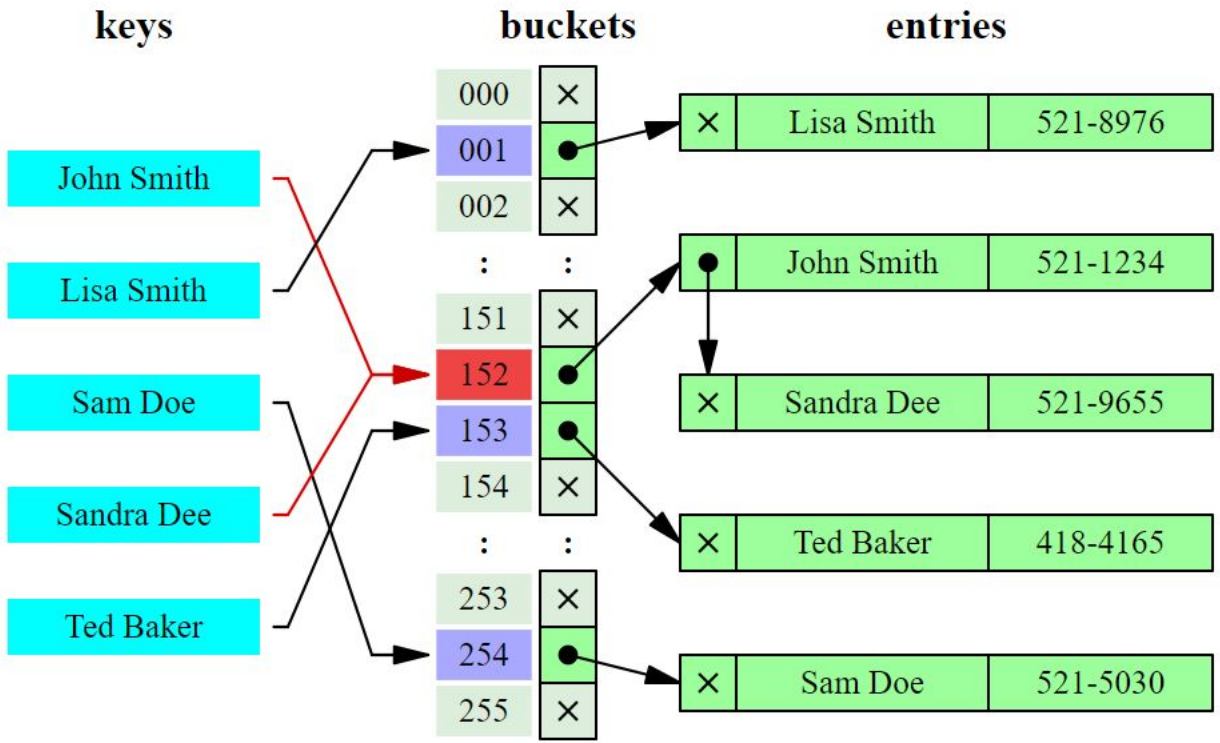
with

[

_Kty=RGBColor

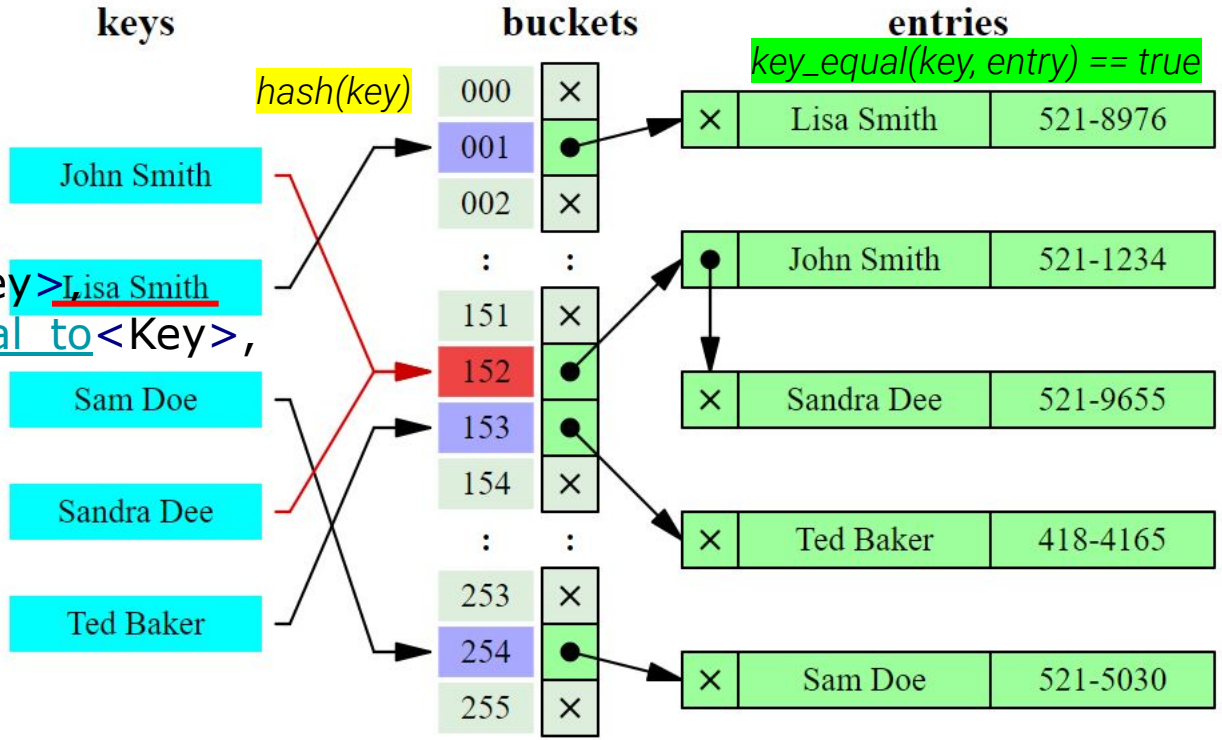
]

std::unordered_map



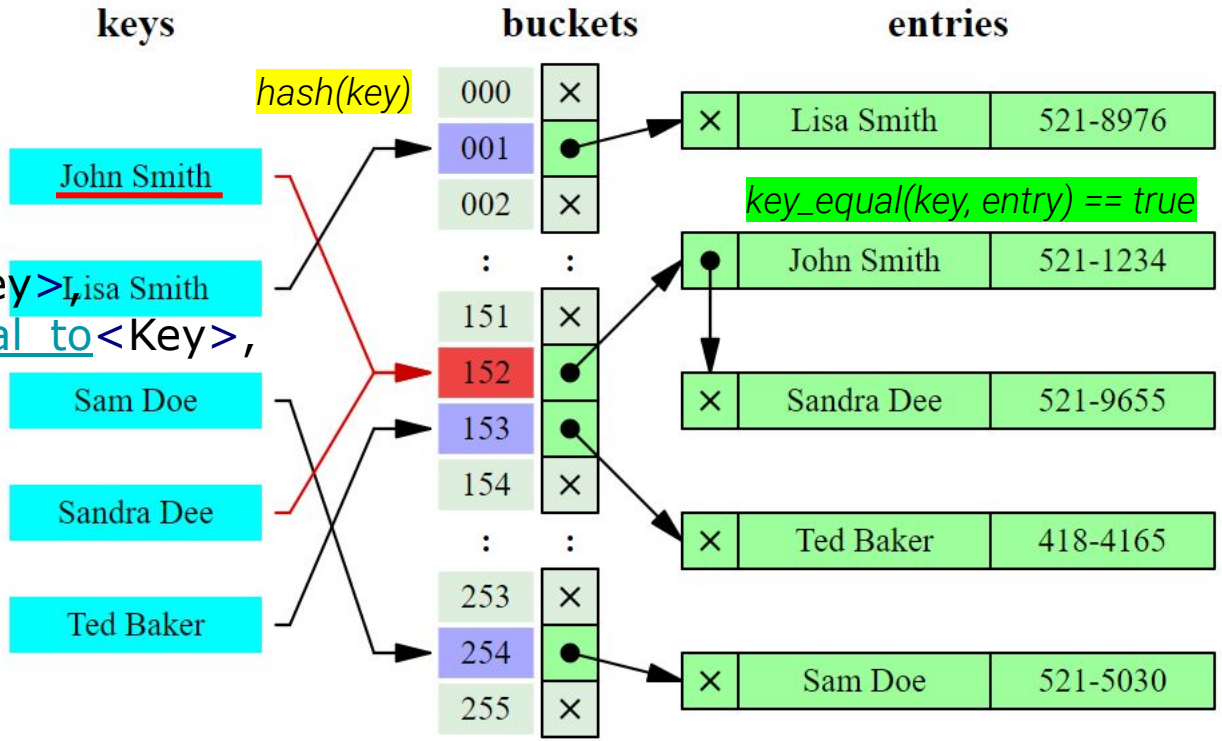
std::unordered_map

```
template< class Key,
    class T,
    class Hash = std::hash<Key>,
    class KeyEqual = std::equal_to<Key>,
    class Allocator = ...>
class unordered_map;
```



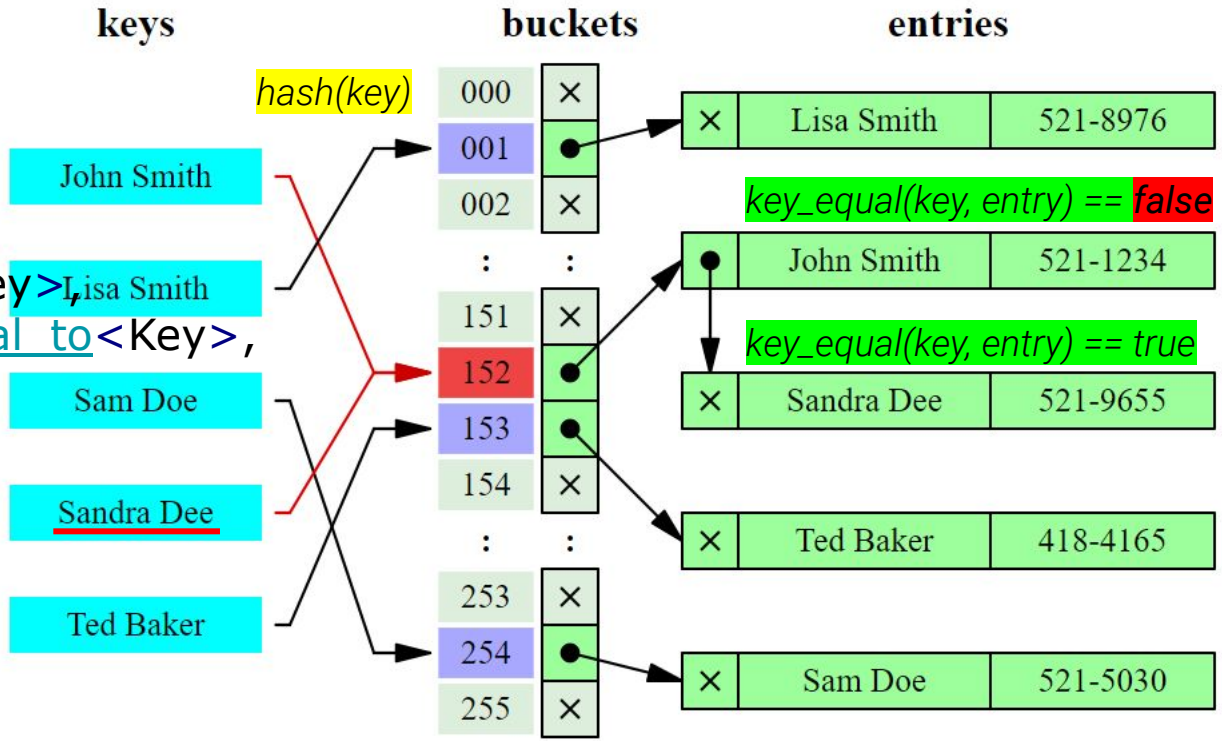
std::unordered_map

```
template< class Key,
    class T,
    class Hash = std::hash<Key>,
    class KeyEqual = std::equal_to<Key>,
    class Allocator = ...>
class unordered_map;
```



std::unordered_map

```
template< class Key,
    class T,
    class Hash = std::hash<Key>,
    class KeyEqual = std::equal_to<Key>,
    class Allocator = ...>
class unordered_map;
```



std::unordered_map для пользовательских типов

std::hash

Defined in header <functional>

```
template< class Key >
struct hash;
```
(since C++11)

`std::hash` - класс (в виде `struct`), определяющий `operator()(const Key&) const;` оператор должен вычислить хэш пришедшего объекта.

1. Accepts a single parameter of type Key.
2. Returns a value of type `std::size_t` that represents the hash value of the parameter.
3. Does not throw exceptions when called.
4. For two parameters k1 and k2 that are equal, `std::hash<Key>()(k1) == std::hash<Key>()(k2)`.
5. For two different parameters k1 and k2 that are not equal, the probability that `std::hash<Key>()(k1) == std::hash<Key>()(k2)` should be very small, approaching `1.0/std::numeric_limits<std::size_t>::max()`.

std::unordered_map для пользовательских типов

Standard specializations for basic types

Defined in header <functional>

```
template<> struct hash<bool>;
template<> struct hash<char>;
template<> struct hash<signed char>;
template<> struct hash<unsigned char>;
template<> struct hash<char8_t>;          // C++20
template<> struct hash<char16_t>;
template<> struct hash<char32_t>;
template<> struct hash<wchar_t>;
template<> struct hash<short>;
template<> struct hash<unsigned short>;
template<> struct hash<int>;
template<> struct hash<unsigned int>;
template<> struct hash<long>;
template<> struct hash<long long>;
template<> struct hash<unsigned long>;
template<> struct hash<unsigned long long>;
template<> struct hash<float>;
template<> struct hash<double>;
template<> struct hash<long double>;
template<> struct hash<std::nullptr_t>;
template< class T > struct hash<T*>;
```

Standard specializations for library types

std::hash<std::string>	(C++11)
std::hash<std::u8string>	(C++20)
std::hash<std::u16string>	(C++11)
std::hash<std::u32string>	(C++11)
std::hash<std::wstring>	(C++11)
std::hash<std::pmr::string>	(C++17)
std::hash<std::pmr::u8string>	(C++20)
std::hash<std::pmr::u16string>	(C++17)
std::hash<std::pmr::u32string>	(C++17)
std::hash<std::pmr::wstring>	(C++17)

hash support for strings
(class template specialization)

`std::hash` - шаблонный класс, имеющий специализации, правильным образом вычисляющие хэш для большинства встроенных типов. Для неспециализированного шаблона operator() «удален» специально, поскольку его действие не определено.

std::unordered_map для пользовательских типов

rgbcolor.hpp (вариант 1)

```
class RGBColor {
    public: uint8_t r, g, b;
    bool operator==(const RGBColor& rhs) const {
        return this->r == rhs.r && this->g == rhs.g && this->b == rhs.b;
    }

};

template<>
struct std::hash<RGBColor> {
    std::size_t operator()(const RGBColor& c) const
    {
        std::size_t h1 = std::hash<uint8_t>{}(c.r);
        std::size_t h2 = std::hash<uint8_t>{}(c.g);
        std::size_t h3 = std::hash<uint8_t>{}(c.g);
        return h1 ^ (h2 << 1) ^ (h3 << 2);
    }
};
```

std::unordered_map для пользовательских типов

```
#include <unordered_map>
#include <string>
#include <iostream>
#include "rgbcolor.hpp“ // <- contains std::hash specialization too

using ColorNameMap = std::unordered_map<RGBColor, std::string>;
using ColorNameMapEntry = std::pair<RGBColor, std::string>;

int main() {
    ColorNameMap color_map = {
        {{255, 0, 0}, "Red"},
        {{0, 255, 0}, "Green"},
        {{255, 255, 255}, "White"}};

    RGBColor color = {255, 0, 0};
    std::cout << color_map[color];
}
```

```
Red
```

std::unordered_map для пользовательских типов

rgbcolor.hpp (вариант 2)

```
class RGBColor {
    public: uint8_t r, g, b;
    bool operator==(const RGBColor& rhs) const {
        return this->r == rhs.r && this->g == rhs.g && this->b == rhs.b;
    }

};

struct MyHasher // instead of std::hash specialization
{
    std::size_t operator()(const RGBColor& c) const
    {
        std::size_t h1 = std::hash<uint8_t>{}(c.r);
        std::size_t h2 = std::hash<uint8_t>{}(c.g);
        std::size_t h3 = std::hash<uint8_t>{}(c.g);
        return h1 ^ (h2 << 1) ^ (h3 << 2);
    }
};
```

std::unordered_map для пользовательских типов

```
#include <unordered_map>
#include <string>
#include <iostream>
#include "rgbcolor.hpp“ // <- without std::hash specialization, but with MyHasher


using ColorNameMap = std::unordered_map<RGBColor, std::string, MyHasher>;
using ColorNameMapEntry = std::pair<RGBColor, std::string>;

int main() {
    ColorNameMap color_map = {
        {{255, 0, 0}, "Red"},
        {{0, 255, 0}, "Green"},
        {{255, 255, 255}, "White"}};

    RGBColor color = {255, 0, 0};
    std::cout << color_map[color];
}
```

```
Red
```

std::unordered_map

```
#include <unordered_map>
#include <string>
#include <iostream>
#include "rgbcolor.hpp“ // <- without std::hash specialization, but with MyHasher


using ColorNameMap = std::unordered_map<RGBColor, std::string, MyHasher>;
using ColorNameMapEntry = std::pair<RGBColor, std::string>;

int main() {
    ColorNameMap color_map = {
        {{255, 0, 0}, "Red"},
        {{0, 255, 0}, "Green"},
        {{255, 255, 255}, "White"}};

    RGBColor color = {255, 0, 0};
    std::cout << color_map[color];
}
```

```
Red
```

std::unordered_map - инвалидация итераторов

Operations	Invalidated
All read only operations, `swap`, `std::swap`	Never
`clear`, `rehash`, `reserve`, `operator=`	Always
`insert`, `emplace`, `emplace_hint`	Only if causes rehash
`erase`	Only to the element erased

std::unordered_map - инвалидация итераторов

Operations	Invalidated
All read only operations, `swap`, `std::swap`	Never
`clear`, `rehash`, `reserve`, `operator=`	Always
`insert`, `emplace`, `emplace_hint`	Only if causes rehash
`erase`	Only to the element erased

std::unordered_map - инвалидация итераторов

std::unordered_map<Key,T,Hash,KeyEqual,Allocator>::rehash

```
void rehash( size_type count );     (since C++11)
```

Sets the number of buckets to `count` and rehashes the container, i.e. puts the elements into appropriate buckets considering that total number of buckets has changed. If the new number of buckets makes load factor more than maximum load factor (`count < size() / max_load_factor()`), then the new number of buckets is at least `size() / max_load_factor()` .

std::unordered_map - инвалидация итераторов

Hash policy

`load_factor` (C++11)	returns average number of elements per bucket (public member function)
`max_load_factor` (C++11)	manages maximum average number of elements per bucket (public member function)
`rehash` (C++11)	reserves at least the specified number of buckets and regenerates the hash table (public member function)
`reserve` (C++11)	reserves space for at least the specified number of elements and regenerates the hash table (public member function)

std::unordered_map - множитель нагрузки

```
float max_load_factor() const;          (1)   (since C++11)
void max_load_factor( float ml );       (2)   (since C++11)
```

Manages the maximum load factor (number of elements per bucket). The container automatically increases the number of buckets if the load factor exceeds this threshold.

1) Returns current maximum load factor.

2) Sets the maximum load factor to `ml`.

std::unordered_map - множитель нагрузки

```
int main()
{
    ColorNameMap color_map = {
      {{255, 0, 0}, "Red"},
     {{0, 255, 0}, "Green"},
     {{255, 255, 255}, "White"}};
     std::cout << "Max load factor is:"
               << color_map.max_load_factor() << '\n';
     std::cout << "LF: " << color_map.load_factor() << '\n';

     for (size_t i = 0; i < 10; i++) {
         uint8_t val = i;
         color_map[RGBColor{val, val, val}] = std::string("");
         std::cout << "LF: "
                   << color_map.load_factor() << '\n';
     }

     return 0;
}
```

```
Max load factor is:1
LF: 0.375
LF: 0.5
LF: 0.625
LF: 0.75
LF: 0.875
LF: 1
LF: 0.140625
LF: 0.15625
LF: 0.171875
LF: 0.1875
LF: 0.203125
```