

Проектирование больших систем на C++

Коноводов В. А.

кафедра математической кибернетики ВМК
vkonovodov@gmail.com

Лекция 3
19.09.2020

Лямбда-выражения

```
[capture] (params) mutable exception_attribute -> ret {body}  
[capture] (params) -> ret {body}  
[capture] (params) {body}  
[capture] {body}
```

Пример:

```
std::vector<int> v = {-1, -2, -3, -4, -5, 1, 2, 3, 4, 5};  
std::sort(v.begin(), v.end(), [](int l, int r) {  
    return l * l < r * r;  
});
```

Лямбда-выражения

- ▶ `[]` — без захвата переменных
- ▶ `[=]` — все переменные захватываются по значению
- ▶ `[&]` — все переменные захватываются по ссылке
- ▶ `[x]` — захват `x` по значению
- ▶ `[&x]` — захват `x` по ссылке
- ▶ `[x, &y]` — захват `x` по значению, `y` по ссылке
- ▶ `[=, &x, &y]` — захват всех переменных по значению, но `x, y` — по ссылке
- ▶ `[&, x]` — захват всех переменных по ссылке, кроме `x`
- ▶ `[this]` — для доступа к переменной класса

return

Вот так не работает:

```
auto cmp = [&data](int a, int b) {  
    if (a == 12)  
        return -1;  
    return data[a] < data[b];  
};
```

return

Вот так не работает:

```
auto cmp = [&data](int a, int b) {  
    if (a == 12)  
        return -1;  
    return data[a] < data[b];  
};
```

А вот так всё хорошо:

```
auto cmp = [&data](int a, int b) -> int {  
    if (a == 12)  
        return -1;  
    return data[a] < data[b];  
};
```

mutable

Те элементы, которые захвачены по значению, автоматически становятся константами внутри лямбды.

```
auto cmp = [&d, d1](int a, int b) mutable -> int {  
    d[0] = d1[0] = 0;  
    if (a == 12)  
        return -1;  
    return d[a] < d[b];  
};
```

Упражнение

Отсортировать массив, не испортив его — вывести перестановку, сохранив исходные данные.

```
const int a[] = {3, 5, 2, 8, 15, 12, -1, 3, 4, 7}; // ...
size_t n = sizeof(a) / sizeof(a[0]);
std::vector<size_t> idx(n);
for (int i = 0; i < n; ++i) {
    idx[i] = i;
}

// ... ?

for (const auto &i : idx) {
    std::cout << a[i] << " ";
}
std::cout << std::endl;
}
```

Захват данных класса

```
class T {  
    private:  
        std::vector<int> Data;  
    public:  
        T(const std::vector<int>& data)  
            :Data(data)  
        {}  
        void Do() {  
            auto f = [this](int a, int b) {  
                return Data[a] < Data[b];  
            };  
        }  
};
```

Захват данных класса

```
class T {  
    private:  
        std::vector<int> Data;  
    public:  
        T(const std::vector<int>& data)  
            :Data(data)  
        {}  
        void Do() {  
            auto f = [=](int a, int b) {  
                return Data[a] < Data[b];  
            };  
        }  
};
```

Опасности захвата по умолчанию

```
using T = std::vector<std::function<bool(int)>>;
```

```
void AddFunc(T& funcs) {  
    static int x = 2;  
    funcs.emplace_back(  
        [=](int v) { return v % x == 0;}  
    );  
    ++x;  
}
```

```
int main() {  
    T funcs;  
    AddFunc(funcs);  
    AddFunc(funcs);  
    funcs[0](5);  
    funcs[1](5);  
}
```

Захватывать явно — заметнее ошибки.

Захватывать указатель тоже может быть опасно.

sizeof

```
int main()
{
    auto f = [](){ return 1; };
    std::cout << sizeof(f) << std::endl;
}
```

sizeof

```
int main()
{
    auto f = [](){ return 1; };
    std::cout << sizeof(f) << std::endl;
}
```

All objects in C++ must have a minimum size of 1 under the standard.

Инкапсуляция сложной инициализации в лямбду

Пример:

```
TSomeType obj;  
for (auto i = 2; i <= N; ++i) {  
    obj += some_func(i);  
}  
// далее obj не меняется
```

Инкапсуляция сложной инициализации в лямбду

Пример:

```
TSomeType obj;  
for (auto i = 2; i <= N; ++i) {  
    obj += some_func(i);  
}  
// далее obj не меняется
```

Вынесем

```
const TSomeType obj = [&]{  
    TSomeType val;  
    for (auto i = 2; i <= N; ++i) {  
        val += some_func(i);  
    }  
    return val;  
}();
```

C++17: constexpr-lambdas

```
constexpr auto add(int y) {  
    return [=](int x) { return x + y;};  
}  
  
int main() {  
    constexpr auto inc = add(5);  
    static_assert(inc(3) == 8);  
}
```

C++17: if-init expressions

```
auto x = get_result();  
if (x == 1) {  
}
```

```
// C++17:  
if (auto x = get_result(); x == 1) {  
}
```

hashset, hashmap

```
template<
    typename Key,
    typename Value,
    typename Hash = std::hash<Key>,
    typename KeyEqual = std::equal_to<Key>,
    typename Allocator =
        std::allocator<std::pair<const Key, Value>>
> class unordered_map;
```

hashmap: случай Key = const char*

```
std::unordered_map<const char*, unsigned> table;  
const char* x = "ABC";
```

```
std::string s = "ABC";  
const char* y = s.c_str();
```

```
table[x] = 1;  
// strcmp(x, y) == 0  
// table.find(y) == table.end()  
// std::hash<const char*>(x) == 13930663984845506963  
// std::hash<const char*>(y) == 12394125337132834388
```

hashmap: случай Key = const char*

Функция для хэширования:

```
struct THashString {  
    void hashCombine(size_t& seed, const char v) const {  
        seed ^= v + 0x9e3779b9 + (seed << 6) + (seed >> 2);  
    }  
  
    size_t operator() (char const* p) const {  
        size_t hash = 0;  
        for (; *p; ++p) {  
            hashCombine(hash, *p);  
        }  
        return hash;  
    }  
};
```

hashmap: случай Key = const char*

Сравнение ключей:

```
struct TCompString {  
    bool operator() (const char* p1, const char* p2) const {  
        return strcmp(p1, p2) == 0;  
    }  
};
```

hashmap: случай Key = const char*

```
std::unordered_map<const char*, unsigned,  
                  THashString, TCompString> table;  
const char* x = "ABC";  
  
std::string s = "ABC";  
const char* y = s.c_str();  
  
table[x] = 1;  
// strcmp(x, y) == 0  
// table.find(y) != table.end()  
// THashString()(x) == 11093822720383  
// THashString()(y) == 11093822720383
```

C++17: std::optional

Если всё хорошо, то есть значение, иначе — значения нет.

```
std::optional<int> GetCount(const int param) {  
    const static std::map<int, int> MAPINT_PARAMS = ....;  
    auto it = MAPINT_PARAMS.find(param);  
    if (it != MAPINT_PARAMS.end()) {  
        return it->second;  
    } else {  
        // return std::nullopt;  
        // return std::optional<int>();  
        return {};  
    }  
}  
  
int main() {  
    int param = 3;  
    if (auto count = GetCount(param)) {  
        std::cout << "res = " << *count << std::endl;  
    }  
}
```

C++17: std::optional

Существует возможность взять std::hash от std::optional.

```
#include <iostream>
#include <optional>
#include <string>
#include <unordered_set>
int main()
{
    std::unordered_set<std::optional<std::string>> s = {
        "ABC", "DEF", std::nullopt
    };

    for(const auto& item : s) {
        std::cout << item.value_or("nothing") << ' ';
    }
}
```

Задача

Реализовать `TStaticAssert` без использования C++11. Ловим ошибки [на этапе компиляции](#):

```
TStaticAssert<1 == 1> assert_one_is_one;
```

std::function

```
struct Foo {  
    void print(int i) const { std::cout << i << std::endl; }  
};  
void print_num(int i) {  
    std::cout << i << std::endl;  
}  
int main() {  
    std::function<void(int)> f_display = print_num;  
    f_display(1);  
  
    std::function<void()> f_display_2 = []() { print_num(2); };  
    f_display_2();  
  
    std::function<void(const Foo&, int)> f_add_display =  
        &Foo::print;  
    f_add_display(Foo(), 1);  
}
```

Функторы в <functional>

```
greater
less
greater_equal
less_equal
equal_to
not_equal_to
plus
minus
multiply
divide
negate -
modulus
logical_and
logical_or
logical_not
```

Функторы в <functional>

```
greater
less
greater_equal
less_equal
equal_to
not_equal_to
plus
minus
multiply
divide
negate -
modulus
logical_and
logical_or
logical_not
```

```
sort(a, a + 5, std::greater<int>());
transform(a, a + 5, b, ostream_iterator<int> (cout, " "),
          plus<int>());
```

Связыватели в C++98

```
remove_copy_if (a,  
                a + 5,  
                ostream_iterator<int> (cout, " "),  
                bind2nd(less<int>(), 3));
```

Связыватели в C++98

Но вот так не работает:

```
class MyCmp {  
    public:  
        bool operator()(int a, int b) const {  
            return a > b;  
        }  
};
```

```
remove_copy_if (a.begin(),  
                a.end(),  
                b.begin(),  
                std::bind2nd(MyCmp(), 3));
```

Связыватели в C++98

А так ok:

```
class MyCmp : public std::binary_function<int, int, bool>{  
    public:  
        bool operator()(int a, int b) const {  
            return a > b;  
        }  
};
```

```
remove_copy_if (a.begin(),  
                a.end(),  
                b.begin(),  
                std::bind2nd(MyCmp(), 3));
```

Отступление: указатели на методы внутри класса

```
class C {  
    private:  
        int a, b;  
    public:  
        C(int a, int b) : a(a), b(b) {};  
        void f() const { ... }  
        void g() const { ... }  
};
```

Хотим сделать указатель на функцию f из класса,

```
void (*ptr) ();           //  
ptr = &C::f;              // так нельзя
```

Отступление: указатели на методы внутри класса

```
class C {  
    private:  
        int a, b;  
    public:  
        C(int a, int b) : a(a), b(b) {};  
        void f() const { ... }  
        void g() const { ... }  
};
```

Хотим сделать указатель на функцию f из класса,

```
void (*ptr) ();           //  
ptr = &C::f;              // так нельзя  
void (C::*ptr)() const;  
ptr = &C::f;              // ok
```

C++11: std::bind

```
int f(int a, int b) {  
    return a - b;  
}
```

```
int main() {  
    auto g = std::bind(f, 3, std::placeholders::_2);  
    std::cout << g(1, 4) << std::endl;  
    return 0;  
}
```

C++11: std::bind

```
int f(int a, int b) {  
    return a - b;  
}
```

```
int main() {  
    auto g = std::bind(f, 3, std::placeholders::_2);  
    std::cout << g(1, 4) << std::endl;  
    return 0;  
}  
-1
```

C++11: std::bind

```
int f(int a, int b) {  
    return a - b;  
}
```

```
int main() {  
    auto g = std::bind(f, 3, std::placeholders::_2);  
    std::cout << g(1, 4) << std::endl;  
    return 0;  
}  
-1
```

```
std::bind(f, _2, _1, 2, 3, 4) (x, y);  // f(y, x, 2, 3, 4)
```

C++11: std::bind

```
int f(int a, int b) {  
    return a - b;  
}  
  
int g(int x) {  
    return x * x;  
}  
  
int main() {  
    auto f2 = std::bind(  
        f,  
        std::bind(  
            g,  
            std::placeholders::_2  
        ),  
        std::placeholders::_1  
    );  
    std::cout << f2(3, 7) << std::endl;  
}
```

C++17: std::invoke

Вызывает callable-объект с заданными аргументами.

```
class C {
    int a;
public:
    C(int a) : a(a) {}
    void f(int k) const {
        std::cout << a << " " << k << std::endl;
    }
};

void print_int(int i) { std::cout << i << std::endl; }

int main()
{
    std::invoke(print_int, 117);
    std::invoke([]() { print_int(117); });
    const C obj(117);
    std::invoke(&C::f, obj, 42);
}
```

C++17: std::apply

Вызывает callable-объект с аргументами, переданными в виде tuple.

```
int add(int first, int second, int third) {  
    return first + second + third;  
}  
  
int main()  
{  
    std::cout << std::apply(add, std::make_tuple(1, 2, 3));  
    // 6  
}
```

C++20: std::bind_front

Вызов callable-объекта с параметрами, первые из которых берутся из аргументов std::bind_front.

Вызов

```
std::bind_front(f, bound_args...)(call_args...)
```

эквивалентен

```
std::invoke(f, bound_args..., call_args....)
```

Пример:

```
int minus(int a, int b){  
    return a - b;  
}  
  
int main()  
{  
    auto f = std::bind_front(minus, 117);  
    std::cout << f(1);    // 116  
}
```