



# Modern C++

## Function objects: Lambda, Bind, Function

Ращенко Елена

27.10.2022

# Структура лекции

- Функциональные объекты в C++98
- Инструменты C++11 и C++14
  - $\lambda$  (lambda)
  - `std::bind`
  - `std::bind` vs `lambda`
- Хранение и передача функций
  - Способы передачи callback функций
  - Обзор `std::function`
  - `std::function` vs `template`
- Идиомы использования

# А ты используешь auto?

```
auto x = 4;  
auto y = 3.37;  
auto ptr = &x;  
std::cout << typeid(x).name(); // int  
std::cout << typeid(y).name(); // double  
std::cout << typeid(ptr).name(); // int *
```

```
auto func() { return 2; }  
auto f = func();
```

```
std::vector<int> data;  
std::vector<int>::iterator it = data.begin();  
auto it = data.begin();
```



# Функциональные объекты в C++98

# Задача “ $18 \leq x < 27$ ”

```
count_if(ages.begin(), ages.end(), ???);
```

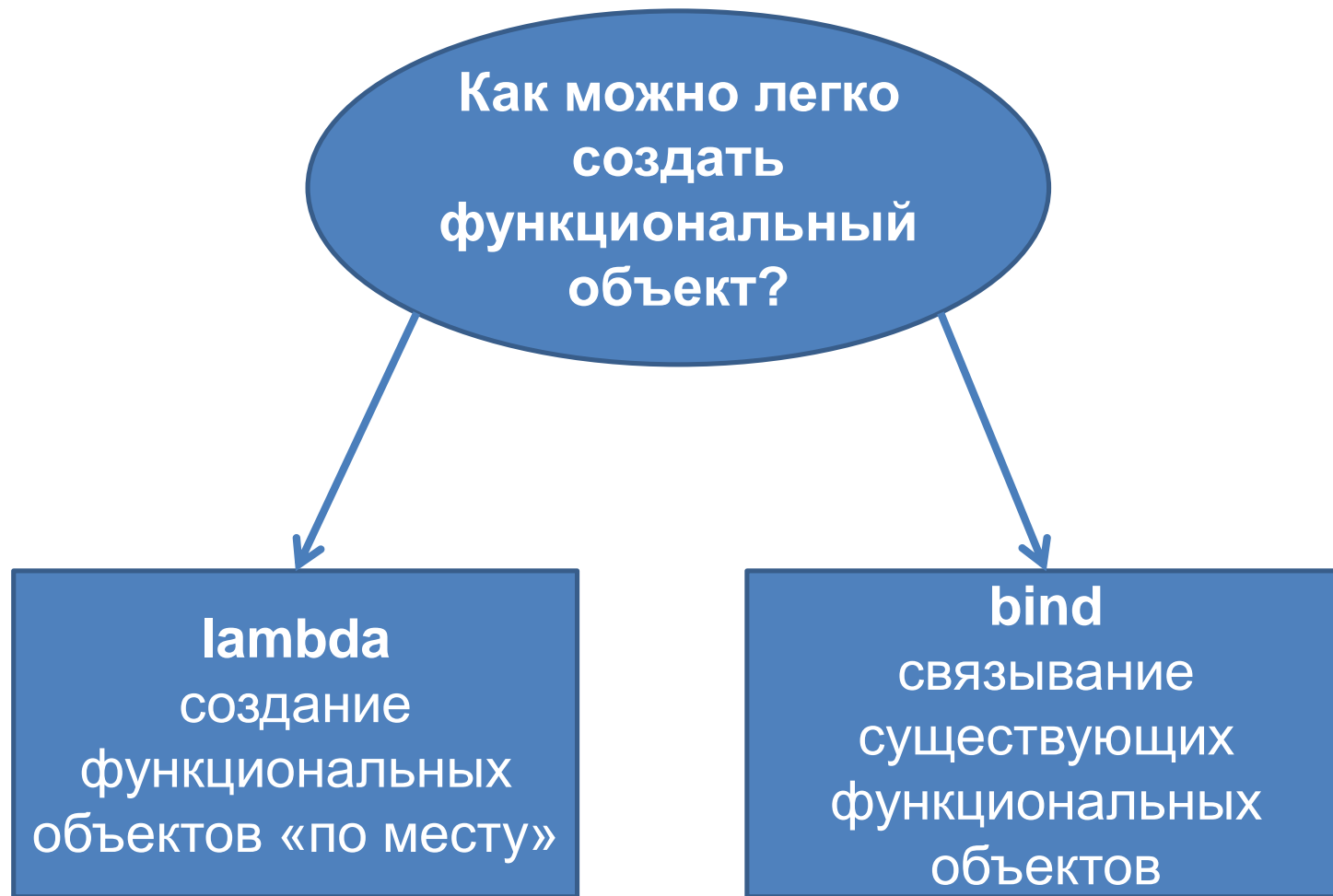
# Задача “ $18 \leq x < 27$ ”

```
struct Between18And27
```

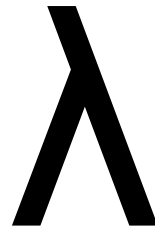
```
{  
    bool operator() (int value) const  
    {  
        return 18 <= value && value < 27;  
    }  
};
```

```
count_if(ages.begin(), ages.end(), Between18And27());
```

# Инструменты C++11 и C++14









# Задача “ $18 \leq x < 27$ ”

```
struct Between18And27
```

```
{  
    bool operator() (int value) const  
    {  
        return 18 <= value && value < 27;  
    }  
};
```

```
count_if(ages.begin(), ages.end(), Between18And27());
```

# Lambda. Задача “ $18 \leq x < 27$ ”

```
struct Between18And27  
{  
    bool operator() (int value) const  
    {  
        return 18 <= value && value < 27;  
    }  
};
```

```
count_if(ages.begin(), ages.end(),  
    [](int value) -> bool  
    {  
        return 18 <= value && value < 27;  
    }));
```

# Синтаксис lambda

`[](){}`

```
[ ] (int value) -> bool
{
    return 18 <= value && value < 27;
}
```

`[capture]`

`(arguments)`

`-> return_type`

`{body}`

# Capture

```
int x = 0;
int y = 0;

cin >> x >> y;

count_if(ages.begin(), ages.end(),
    [] (int value)
    {
        return ??? <= value && value < ???;
    });
```

# Capture

```
int x = 0;
```

```
int y = 0;
```

```
cin >> x >> y;
```

```
count_if(ages.begin(), ages.end(),  
    [] (int value)  
    {  
        return x <= value && value < y;  
    });
```

**error C3493:** 'x' cannot be implicitly captured  
because no default capture mode has been specified

# Capture по значению

```
int x = 0;
```

```
int y = 0;
```

```
cin >> x >> y;
```

```
count_if(ages.begin(), ages.end(),
```

```
    [x, y] (int value)
```

```
{
```

```
    return x <= value && value < y;
```

```
});
```

# Capture по значению

```
int x = 1;  
  
for_each(ages.begin(), ages.end(),  
    [x] (int value)  
    {  
        x *= value;  
    }));
```

**error C3491:** 'x': a by copy capture cannot be modified in a non-mutable lambda



# Capture по значению

```
int x = 1;

for_each(ages.begin(), ages.end(),
    [x] (int value) mutable
    {
        x *= value;
    });

cout << x; // 1
```

# Capture по ссылке

```
int x = 1;

for_each(ages.begin(), ages.end(),
    [&x] (int value) mutable
    {
        x *= value;
    });

cout << x; // 42
```

# Capture по ссылке

```
int x = 1;

for_each(ages.begin(), ages.end(),
    [&x] (int value)
    {
        x *= value;
    });

cout << x; // 42
```

# Опасность передачи по ссылке

```
auto MakeLambda()  
{  
    int x = 0;  
    int y = 0;  
    cin >> x >> y;  
  
    return [&x, &y] (int value)  
    {  
        return x <= value && value < y;  
    };  
}
```

# Висячий указатель

```
auto MakeLambda()  
{  
    int x = 0;  
    int y = 0;  
    cin >> x >> y;  
  
    return [&x, &y] (int value)  
    {  
        return x <= value && value < y;  
    };  
}  
  
count_if(ages.begin(), ages.end(), MakeLambda());  
// Incorrect. Possible crash!!!
```

# Capture “this”

```
class MyClass
{
public:
    void ProcessInt(int a) { ... }

    void ProcessVectorOfInts(const vector<int>& va)
    {
        for_each(va.begin(), va.end(),
            [] (int a)
            {
                ProcessInt(a);
            });
    }
};
```

**error C2352:** 'MyClass::ProcessInt' : illegal call of non-static member function

# Capture “this”

```
class MyClass
{
public:
    void ProcessInt(int a) { ... }

    void ProcessVectorOfInts(const vector<int>& va)
    {
        ...
        ProcessInt(777);
        ...
    }
};
```



# Capture “this”

```
class MyClass
{
public:
    void ProcessInt(int a) { ... }

    void ProcessVectorOfInts(const vector<int>& va)
    {
        ...
        this->ProcessInt(777);
        ...
    }
};
```

# Capture “this”

```
class MyClass
{
public:
    void ProcessInt(int a) { ... }

    void ProcessVectorOfInts(const vector<int>& va)
    {
        for_each(va.begin(), va.end(),
            [] (int a)
            {
                ProcessInt(a);
            });
    }
};
```

# Capture “this”

```
class MyClass
{
public:
    void ProcessInt(int a) { ... }

    void ProcessVectorOfInts(const vector<int>& va)
    {
        for_each(va.begin(), va.end(),
            [this] (int a)
            {
                ProcessInt(a);
            });
    }
};
```

# Много параметров в capture

```
int firstCoolParam = 1;

for_each(va.begin(), va.end(),
    [firstCoolParam] (int& i_value)
    {
        i_value += firstCoolParam;
    });
```

# Много параметров в capture

```
int firstCoolParam = 1;
int secondCoolParam = 1;

for_each(va.begin(), va.end(),
    [firstCoolParam, secondCoolParam]
    (int& i_value)
    {
        i_value +=
            firstCoolParam * secondCoolParam;
    });
```

# Default capture

```
int firstCoolParam = 1;
int secondCoolParam = 1;

for_each(va.begin(), va.end(),
    [=] (int& i_value)
    {
        i_value +=
            firstCoolParam * secondCoolParam;
    });
```

Не рекомендуется!

# Default capture

```
int firstCoolParam = 1;
int secondCoolParam = 1;

for_each(va.begin(), va.end(),
    [&] (int& i_value)
    {
        firstCoolParam += i_value;
        secondCoolParam *= i_value;
    });
```

Не рекомендуется (есть исключения)!



# Lambda type

```
???? lambda = [&x, y] (int val) { return ++x + y + val; };
```

```
class ???  
{  
    public:  
        ???(int& _x, int& _y) : x{_x} , y{_y} {}  
  
        int operator()(int val) const  
        {  
            return ++x + y + val;  
        }  
  
    private:  
        int& x;  
        int y;  
};
```

[cppinsights.io](http://cppinsights.io)

# Lambda type

```
auto lambda = [&x, y] (int val) { return ++x + y + val; };
```

```
class ???  
{  
    public:  
        ???(int& _x, int& _y) : x{_x} , y{_y} {}  
  
        int operator()(int val) const  
        {  
            return ++x + y + val;  
        }  
  
    private:  
        int& x;  
        int y;  
};
```

# Generic lambdas (C++14)

```
auto print1 = [](auto a) { cout << a ; };
```

```
print1("aaa");  
print1(15);
```

```
auto sum = [](auto a, auto b) { return a + b; };
```

# Capture initializer (C++14)

```
auto lambda = [x = GetValue()] (auto a)
{
    return a < x;
};
```

```
std::set<int> cont = GetHugeContainer();
auto isInContainer =
    [c = std::move(cont)](int i)
    {
        return c.contains(i); // contains is from C++20
    };
```

# Lambda with template param(C++20)

```
auto sumGen =  
    [](auto fir, auto sec) { return fir + sec; }; // C++14  
  
sumGen(4.1, 1);  
  
auto sumTem =  
    []<class T>(T fir, T sec) { return fir + sec; }; // C++20  
  
sumTem(4.1, 1); // error
```

# Lambda with template param(C++20)

```
auto sumGen =  
    [](auto fir, auto sec) { return fir + sec; }; // C++14  
  
template<class T1, class T2>  
auto operator()(T1 fir, T2 sec) const  
{  
    return fir + sec;  
}
```

# Lambda with template param(C++20)

```
auto sumTem =  
    [<class T>(T fir, T sec) { return fir + sec; }]; // C++20  
  
template<class T>  
auto operator()(T fir, T sec) const  
{  
    return fir + sec;  
}
```





std::bind

# Использование std::bind

```
#include <functional>
```

```
using namespace std;
```

```
using namespace std::placeholders;
```

# Использование std::bind

```
void f(int a, int b, int c, int d)
{
    cout << a << b << c << d << endl;
}
```

Мы хотим получить **g(int x)**, такую что:

```
void g(int x)
{
    f(7, 6, 5, x);
}
```

```
auto g = bind(f, 7, 6, 5, _1);
g(4); // 7654
```

# Использование std::bind

```
void f(int a, int b, int c, int d);
```

```
auto g2 = bind(f, _1, _2, 5, 6);  
g2(3, 4);  
// 3456
```

```
auto g0 = bind(f, 1, 2, 3, 4);  
g0();  
// 1234
```

```
auto g3 = bind(f, _2, _1, _1, _3);  
g3(7, 8, 9);  
// 8779
```

# std::bind. Задача “x < 27”

```
template<class T = void>  
struct less; // STL: Functional objects
```

```
count_if(ages.begin(), ages.end(),  
         bind(less<int>(), _1, 27))
```

# std::bind. Задача “ $18 \leq x < 27$ ”

```
count_if(ages.begin(), ages.end(),  
    bind(logical_and<bool>(),  
        bind(greater_equal<int>(), _1, 18),  
        bind(less<int>(), _1, 27)));
```

```
// Note: inner bind expressions  
// are invoked eagerly and  
// returned value is passed to outer one
```

# std::bind. Задача “ $x \leq a < y$ ”

```
int x = 0;
```

```
int y = 0;
```

```
cin >> x >> y;
```

```
count_if(ages.begin(), ages.end(),  
         bind(logical_and<bool>(),  
             bind(greater_equal<int>(), _1, x),  
             bind(less<int>(), _1, y)));
```

# std::bind. Методы класса

```
struct Multiplier
{
    Multiplier (int multiplier) :
        m_multiplier(multiplier)
    {}

    void CountValue(int value) const
    {
        cout << value * m_multiplier << endl;
    }

    int m_multiplier;
};
```



# std::bind. Методы класса

```
Multiplier m1_5(5);  
Multiplier m1_20(20);
```

```
auto g1 = bind(&Multiplier::CountValue, m1_5, _1)  
g1(7);  
// 35
```

```
auto g2 = bind(&Multiplier::CountValue, _1, _2)  
g2(m1_20, 7);  
// 140
```

```
auto g0 = bind(&Multiplier::CountValue, m1_5, 10)  
g0();  
// 50
```

# std::bind. Методы класса

```
vector<int> collInt;  
Multiplier m1_5(5);  
  
for_each(collInt.begin(), collInt.end(),  
    bind(&Multiplier::CountValue, m1_5, _1));  
  
vector<Multiplier> collMul;  
  
for_each(collMul.begin(), collMul.end(),  
    bind(&Multiplier::CountValue, _1, 10));
```

# std::bind. Методы класса

```
vector<Multiplier*> coll;
```

```
for_each(coll.begin(), coll.end(),  
    bind(&Multiplier::CountValue, _1, 10));
```

```
vector<shared_ptr<Multiplier>> coll;
```

```
for_each(coll.begin(), coll.end(),  
    bind(&Multiplier::CountValue, _1, 10));
```

# std::bind. Передача параметров

```
struct Accumulator
{
    void AddValue(int i_val) { m_summ += i_val; }

    int GetSumm() const { return m_summ; }

private:
    int m_summ = 0;
};
```

# std::bind. Передача параметров

```
Accumulator acc;  
  
for_each( ages.begin(), ages.end(),  
    bind(  
        &Accumulator::AddValue, acc, _1));  
  
cout << acc.GetSumm() << endl; // 0
```

# std::bind. Передача параметров

```
Accumulator acc;  
  
for_each( ages.begin(), ages.end(),  
    bind(  
        &Accumulator::AddValue, acc, _1));  
  
cout << acc.GetSumm() << endl;
```

*constructor*  
*copy constructor*  
*copy constructor*  
*destructor*  
*destructor*  
*0*  
*destructor*

# std::bind. Передача параметров

```
Accumulator acc;  
  
for_each( ages.begin(), ages.end(),  
    bind(  
        &Accumulator::AddValue, ref(acc), _1) );  
  
cout << acc.GetSum() << endl; // 197
```

*constructor*  
*197*  
*destructor*



# To bind or not to bind std::bind vs lambda



# “To bind or not to bind”. Сложность

```
void print(ostream& os, size_t i)
{
    os << i << endl;
}
```

```
map<string, vector<float>> m;
```

```
for_each(m.begin(), m.end(),
    bind(print, cout,
        bind(&vector<int>::size,
            bind(
                &map<string, vector<int>>::value_type::second, _1)))));
```

# “To bind or not to bind”. Сложность

```
void print(ostream& os, size_t i)
{
    os << i << endl;
}

map<string, vector<float>> m;

for_each(m.begin(), m.end(),
    [] (const auto& x)
    {
        print(cout, x.second.size());
    });
```

# “To bind or not to bind”. Перепутзка

```
void process(int a) { /*do int processing*/ }  
void process(double a) { /*do double processing*/ }
```

```
auto l = []() { process(1); };
```

```
auto b = bind(process, 1);
```

error C2672: 'std::bind': no matching overloaded function found

# “To bind or not to bind”. Перепутка

```
void process(int a) { /*do int processing*/ }  
void process(double a) { /*do double processing*/ }  
  
auto l = []() { process(1); };  
  
auto b = bind(  
    static_cast<void(*)>(int)>(process), 1);
```

# “To bind or not to bind”. Аргумент за?

```
struct B
{
    int f(int a, int b, int c)
    {
        return a + b + c;
    }
} b;
```

```
auto f = bind(&B::f, ref(b), 1, -1, _1);
```

```
auto f1 = [&b](int c)
{
    return b.f(1, -1, c);
};
```

# “To bind or not to bind”. Аргумент за?

std::bind:

return f(0);

B::f(int, int, int):

add esi, edx

lea eax, [rsi+rcx]

ret

main:

sub rsp, 24

xor ecx, ecx

mov edx, -1

mov esi, 1

lea rdi, [rsp+15]

call B::f(int, int, int)

add rsp, 24

ret

Lambda:

return f1(0);

main:

xor eax, eax

ret



# Callback

```
void Alarm(int i_time, ??? i_callback)
{
    ::Sleep(i_time);
    i_callback ???;
}
```



# Callback (function pointer)

```
void DoBeep()  
{  
    // make beep-beep-beep  
}  
  
void DoBlink()  
{  
    // make blink-blink-blink  
};
```

# Callback (function pointer)

```
void Alarm(int i_time, void(*i_callback)(void) )  
{  
    ::Sleep(i_time);  
    if (i_callback)  
    {  
        (*i_callback)();  
    }  
}
```

```
Alarm(3, DoBeep);  
Alarm(3, DoBlink);
```

# Callback (interface)

```
struct IAlarmObserver
{
    virtual void OnAlarm() = 0;
};
```

```
struct Beeper :
    IAlarmObserver
{
    void OnAlarm() override
    {
        // make beep-beep
    }
};
```

```
struct Blinker :
    IAlarmObserver
{
    void OnAlarm() override
    {
        // make blink-blink
    }
};
```

# Callback (interface)

```
void Alarm(int i_time, IAlarmObserver& i_callback)
{
    ::Sleep(i_time);
    i_callback.OnAlarm();
}
```

```
Beeper beeper;
Alarm(3, beeper);
```

```
Blinker blinker;
Alarm(3, blinker);
```

# Callback (template)

```
template<typename TCallback>
void Alarm(int i_time, const TCallback& i_callback)
{
    ::Sleep(i_time);
    i_callback();
}
```

```
Alarm(3, DoBeep);
```

```
CBeeper beeper;
Alarm(3, beeper);
```

```
Alarm(3, []() { /*do stuff*/ });
```

```
Blinker blinker;
Alarm(3, bind(&Blinker::OnAlarm, blinker));
```

# std::function

```
bool SomeFunc (int first, int second);
```

```
function<bool (int, int)> f1 = SomeFunc;  
f1(25, 27);
```

```
function<bool (int, int)> f2 = less<int>();  
f2(25, 27);
```

```
function<bool (int)> f3 = bind(less<int>(), _1, 27);  
f3(25);
```

# std::function

```
function< bool (int, int) > f = SomeFunc;  
f(12, 15);
```

```
function< bool (int, int) > f2;  
f2(12, 15); // bad_function_call exception
```

```
if (f2)  
{  
    //...  
}
```

```
f2 = f;  
f2 = nullptr;
```

# Callback

```
void Alarm(int i_time,  
           const function<void (void)>& i_callback)  
{  
    ::Sleep(i_time);  
    if (i_callback) i_callback();  
}
```

```
Alarm(3, DoBeep);
```

```
CBeeper beeper;  
Alarm(3, beeper);
```

```
Alarm(3, []() { /*do stuff*/ });
```

```
Blinker blinker;  
Alarm(3, bind(&Blinker::OnAlarm, blinker));
```



# std::function. Передача параметров

```
class Accumulator
{
public:
    void AddValue(int i_val) { m_summ += i_val; }

    int GetSumm() const { return m_summ; }

private:
    int m_summ = 0;
};
```

# std::function. Передача параметров

```
class Accumulator
{
public:
    void operator()(int i_val) { m_summ += i_val; }

    int GetSumm() const { return m_summ; }

private:
    int m_summ = 0;
};
```

# function

Accumulator acc;

function< void (int) > f1 = acc;

function< void (int) > f2 = acc;

f1(10);

f2(10);

cout << acc.GetSumm() << endl;

// 0

# function

Accumulator acc;

```
function< void (int) > f1 = ref(acc);
```

```
function< void (int) > f2 = ref(acc);
```

```
f1(10);
```

```
f2(10);
```

```
cout << acc.GetSumm() << endl;
```

```
// 20
```

# Недостатки function

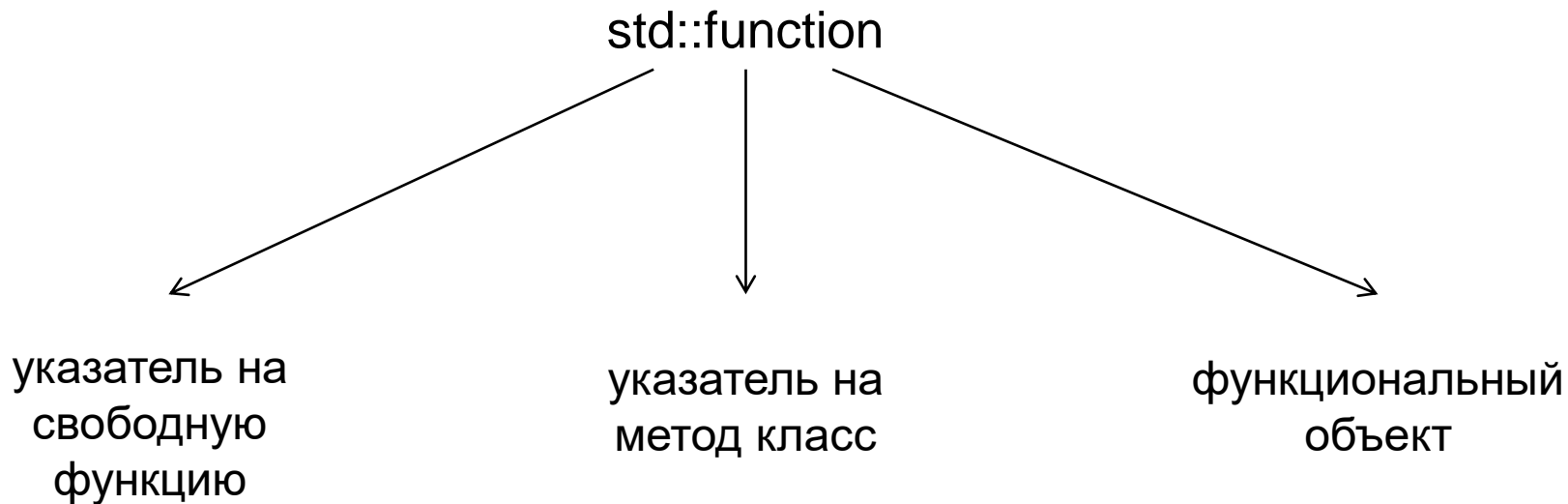
- 1) Размер объекта function в **восемь** раз больше указателя на функцию.

совет: используйте const function& для передачи:

```
void Alarm(int i_time,  
    const function<void (void)>& i_pCallback)  
{  
    ...  
}
```

- 2) Для выполнения требуется **несколько** вложенных вызовов, в отличие от одного у указателя на функцию.

# Достоинства std::function



# std::function vs template

```
class Base
{
    virtual process(const
        std::function<void()>& i_callback)
    {}
};
```

# std::function vs template

```
void beep(){ /*do beep*/ }
```

```
vector<function<void()>> actions;
```

```
actions.push_back(beep);
```

```
actions.push_back([](){/*do stuff*/});
```

```
actions.push_back(bind(plus<int>(), 1, 2));
```

```
for (const auto& f : actions) { f(); }
```



# Практическое правило

```
class A
{
    std::function<void()> m_callback;
};
```

```
template<typename T>
void foo(const T& i_callback);
```



# Идиомы использования

# Неверные типы!

```
auto l = [](int x) -> int
{
    return x * 0.5;
};
cout << l(3); // 1
```

```
function<void()> f = [](){} // performance
auto f = [](){};
```

# STL algorithms

```
transform(points.begin(), points.end(),  
           points.begin(),  
           [](const Point& p)  
           { return Point(p.x + 10, p.y + 10); }));
```

```
all_of(points.begin(), points.end(),  
        [&circle](const Point& p)  
        { return circle.Contains(p); }));
```

```
students.erase(  
    remove_if(students.begin(), students.end(),  
              [](const Student& x){ return !x.isListening(); } ),  
    students.end());
```

# Callback (async)

```
struct Query
{
    Query(function<void(std::vector<int>)> callback);
    function<void(std::vector<int>)> callback;
};
```

```
Query query(
    [](std::vector<int> data)
    {
        cout << "data arrived!!!";
        /* do something with data */
    });
```





# IILE

## (immediately invoked lambda expression)

```
string str;  
if (...)   
    str = "beep";  
else   
    str = "blink";  
  
foo(str);
```

```
foo([&]{  
    if (...)   
        return "beep";  
    else   
        return "blink";  
})();
```



# IILE

## (immediately invoked lambda expression)

```
string str;  
if (...)   
    str = "beep";  
else  
    str = "blink";
```

```
/* do something  
   with str */
```

```
const string str = [&]{  
    if (...)   
        return "beep";  
    else  
        return "blink";  
}();
```

```
/* do something  
   with str */
```



# Отложенные вычисления

```
const auto calcBigObject = [](){ ... };  
if (...)  
{  
    auto o = calcBigObject();  
    ...  
}  
else if (...) { ... }  
else  
{  
    auto o = calcBigObject();  
    ...  
}
```



# Дополнительные ресурсы

- Book: Effective Modern C++, Scott Meyers
- Useful resources:
  - <https://gcc.godbolt.org/>
  - <https://cppinsights.io/>
- Jason Turner [YouTube channel](#):
  - [C++ Weekly](#)
  - [C++ Lambdas](#)
- Habr:
  - Лямбды: от C++11 до C++20  
[Часть 1](#), [Часть 2](#)



# Q&A