

Лямбда-вирази

Ніколайчук Михайло
ФВЕ, 2-й курс магістратури

Джерела

Документація	http://en.cppreference.com/w/cpp/language/lambda
Більшість прикладів коду	https://habrahabr.ru/post/66021
Деякі мінорні приклади	https://stackoverflow.com/questions/16501/what-is-a-lambda-function
Всі прикладу у повному варіанті	https://github.com/Physmatik/lambda

Зміст

- Походження терміну «лямбда-функція»
 - Короткий огляд їх властивостей
- Лямбда-вирази у C++
 - Загальний синтаксис
 - Тип, що повертається
 - Список захоплення
 - Наявність стану у лямбда-виразів
 - `this`` у лямбдах

Що таке «лямбда»?



Anonymous lambda function

Що таке «лямбда»?

Термін вперше з'явився у «[Лямбда-численні](#)» (Алонзо Черч, 1930-ті).
Позначав просто функцію. Зараз його використовують чисто за інерцією.

Syntax	Name	Description
a	Variable	A character or string representing a parameter or mathematical/logical value
$(\lambda x.M)$	Abstraction	Function definition (M is a lambda term). The variable x becomes bound in the expression.
$(M\ N)$	Application	Applying a function to an argument. M and N are lambda terms.

Лямбди взагалі

```
>> a = [1, 2, 3, 4, 5, 6]
>> print(map(lambda x: x * x, a))
[1, 4, 9, 16, 25, 36]
```

```
>> a = [1, 2, 3, 4, 5, 6]
>> print(filter(lambda x: x % 2 == 0, a))
[2, 4, 6]
```

```
def adder(x):
    return lambda y: x + y

add5 = adder(5)

add5(1) # 6
```

На практиці «лямбдами» називають функції, у яких немає імен.

Найчастіше їх використовують як «блоки» коду, які треба виконати деінде (в смолтоку, наприклад, «лямбди» так і називаються – «блоки»).

Крім того, вони дозволяють вам робити елегантні замикання (як зліва).

** Якщо ви незнайомі з функційним програмуванням, це може здатись вам дивним і незручним. Ні. Це круто і зручно.*

Анонімні функції у C++

скоро::програмувати<'на'> &языке << !C++ ^будут* << _так?;

Загальний синтаксис

```
[ captures ] ( params ) specifiers(optional) exception attr -> ret { body }
```

[captures]	-- список захоплення
(params)	-- параметри, що передаються у функцію
specifiers	-- mutable and/or constexpr
exception	-- <i>noexcepts(optional) or exception specification</i>
attr	-- <u>attribute specification</u> (implementation-specific)
-> ret	-- return type

Приклади:

```
auto glambda = [](auto a, auto&& b) { return a < b; }; // C++14 or later
int x = 4;
auto y = [&r=x, x=x+1] mutable ()->int { r+=2; return x*x; }(); //x->6, y=25
```


Що за магія і як воно працює?

// ЛІСТИНГ 1

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <vector>

using namespace std;
int main()
{
    vector<int> srcVec;
    for (int val = 0; val < 10; val++) {
        srcVec.push_back(val);
    }

    for_each(srcVec.begin(), srcVec.end(),
        [](int n){ cout << n << " "; }
    );

    cout << endl;

    return EXIT_SUCCESS;
}
```

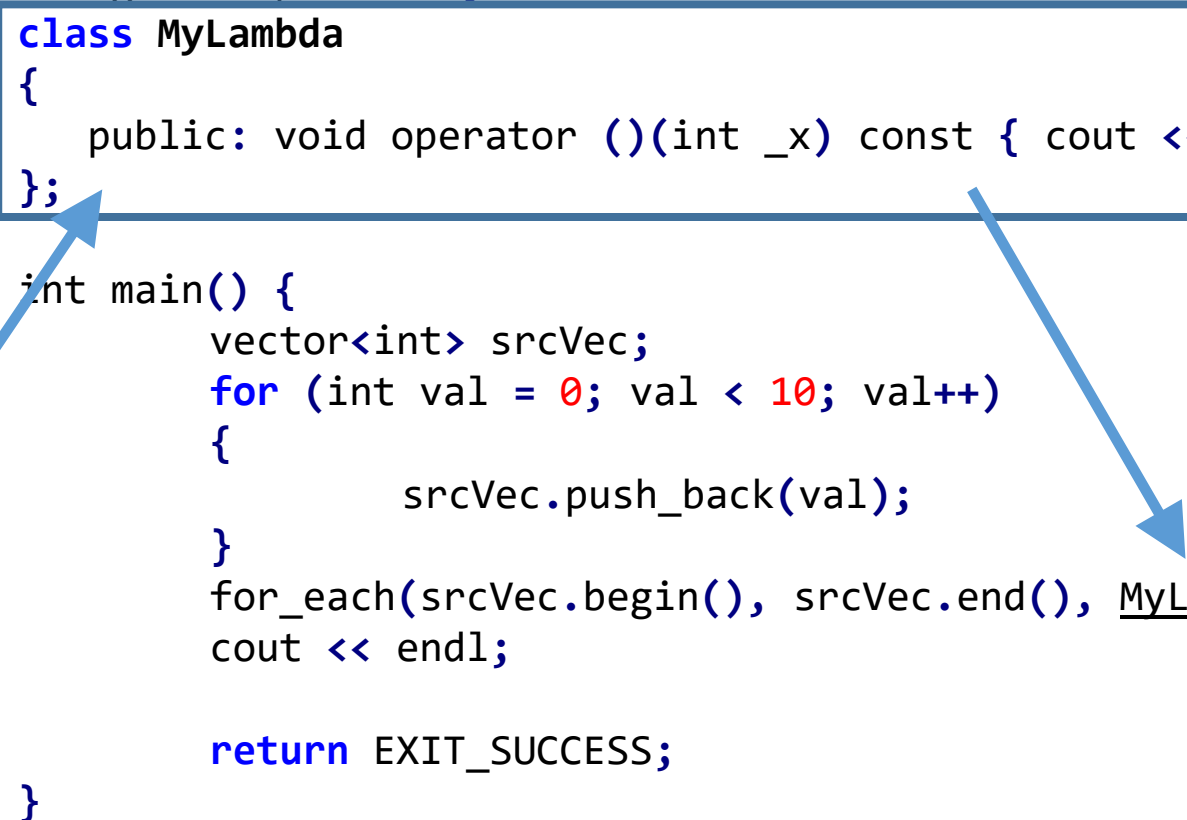
// ЛІСТИНГ 2

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <vector>

using namespace std;
class MyLambda
{
    public: void operator()(int _x) const { cout << _x << " "; }
};

int main() {
    vector<int> srcVec;
    for (int val = 0; val < 10; val++)
    {
        srcVec.push_back(val);
    }
    for_each(srcVec.begin(), srcVec.end(), MyLambda());
    cout << endl;

    return EXIT_SUCCESS;
}
```



Тип, що повертається

За замовчуванням повертається тип `void`. В принципі, якщо компілятор здатний однозначно вивести тип, що повертається, то його можна не вказувати (*хаскелізація плюсів?*).

```
// listing 3
// ...
int result =
    count_if(srcVec.begin(), srcVec.end(), [] (int _n) {
        return (_n % 2) == 0;
    });

cout << result << endl;
// ...
```

Коли компілятору треба підказка

```
// listing 4
// ...

transform(srcVec.begin(), srcVec.end(),
          back_inserter(destVec), [] (int _n) -> double {
    if (_n < 5)
        return _n + 1.0;
    else if (_n % 2 == 0)
        return _n / 2.0;
    else
        return _n * _n;
});

// ...
```

Захоплення змінних (const by default)

```
// listing 5
// ...

cin >> lowerBound >> upperBound;
int result =
    count_if(srcVec.begin(), srcVec.end(),
        [lowerBound, upperBound] (int _n) {
            return lowerBound <= _n && _n < upperBound;
        });
cout << result << endl;

// ...
```

Неконстантный стан лямбд

```
// listing6
// ...

vector<int> srcVec;
int init = 0;
generate_n(back_inserter(srcVec), 10, [init] () mutable
{
    return init++;
});

ostream_iterator<int> outIt(cout, " ");
copy(srcVec.begin(), srcVec.end(), outIt);
cout << endl << "init: " << init << endl;

// 0 1 2 3 4 5 6 7 8 9
// init: 0
```

Зовнішні об'єкти by-reference

```
// listing7
// ...

vector<int> srcVec;
int init = 0;
generate_n(back_inserter(srcVec), 10, [&] () mutable
{
    return init++;
});

ostream_iterator<int> outIt(cout, " ");
copy(srcVec.begin(), srcVec.end(), outIt);
cout << endl << "init: " << init << endl;

// 0 1 2 3 4 5 6 7 8 9
// init: 10
```

Всі види захоплень

[]	no capture
[=]	all by-value
[&]	all by-reference
[x, y]	`x` and `y` by-value
[&x, &y]	`x` and `y`
[&in, out]	`in` by-reference , `out` by-value
[&, out1, out2]	all by-reference , except for `out1` and `out2`
[&, x, &y]	all by-reference , except for `x`

Лямбди і приватність

```
// listing8
class MyMegaInitializer
{
public:
    // ...
    void initializeVector(vector<int> & _vec)
    {
        for_each(_vec.begin(), _vec.end(), [this] (int & _val) mutable
        {
            _val = m_val;
            m_val *= m_power;
        });
    }
private:
    int m_val, m_power;
};
```


Додаткові хитрі можливості

- Лямбди можна повертати як результат функції (і глибше)

```
return [] (int x) -> int
{
    cout << "you result is " << rand() % x;
};
```

- Починаючи з C++14 можна використовувати auto майже усюди

```
auto add = [] (auto x, auto y) {
    return x + y;
};
```