

Структура языка программирования

Программные конструкции.

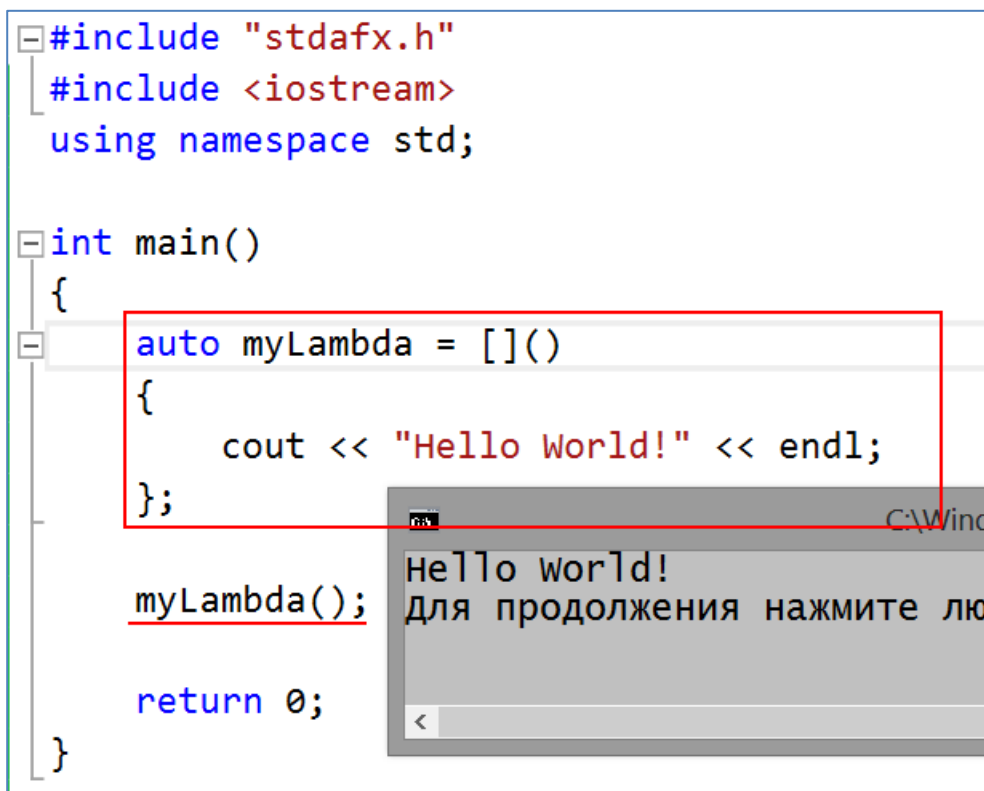
24. Программные конструкции: лямбда-выражение. Лямбда-выражение в программировании — специальный синтаксис для определения функциональных объектов, заимствованный из λ -исчисления, лежит в основе Haskell, ML и других функциональных языков.

Лямбда-выражения поддерживаются во многих языках программирования: Common Lisp, Ruby, Perl, Python, PHP, JavaScript (начиная с ES 2015), C#, F#, Visual Basic .NET, C++, Java, Scala, Kotlin, Object Pascal (Delphi), Нахе и других.

Лямбда-выражения появились в C++11.

Лямбда-выражения представляют собой анонимные функции, которые можно определить в любом месте программы.

Пример простой лямбда-функции:



```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
    auto myLambda = []()
    {
        cout << "Hello World!" << endl;
    };

    myLambda();

    return 0;
}
```

hello world!
Для продолжения нажмите лю

Структура лямбда-выражения:

```
[ маска_переменных ] ( список_параметров ) mutable throw() ->
возвращаемый_тип
{
    /* тело_лямбда-выражения */
}
```

Маска переменных (обязательно).

Лямбда-выражение может получать доступ к переменным вне лямбда-выражения, определенным в той же области видимости, что и лямбда, и использовать их внутри тела.

Маска определяет способ получения параметров (захват переменных) телом лямбда-выражения.

Захват переменных означает, что лямбда может использовать не только переменные, которые передаются в качестве параметров, но и объекты, которые были объявлены вне лямбда-выражения:

- **[a,&b]** *a* захвачена по значению (копия), *b* захвачена по ссылке;
- **[this]** захватывает указатель **this** по значению;
- **[&]** захват всех символов по ссылке;
- **[=]** захват всех символов по значению;
- **[]** ничего не захватывает.

Список параметров (необязательно).

Список параметров лямбда-выражения, аналогичен записи аргументов для обычных функций. Лямбда-выражение может принимать в качестве параметра другое лямбда-выражение.

В C++11 параметры лямбда-выражений требовалось объявлять с указанием конкретных типов. C++14 снимает это ограничение и позволяет объявлять параметры лямбда со спецификатором типа auto.

Начиная с C++14 можно использовать параметры по умолчанию.

mutable (необязательно) — использование **mutable** позволяет модифицировать параметры, переданные по значению.

throw() (необязательно) — спецификация исключений, то есть лямбда-выражения могут выбрасывать исключения.

Возвращаемый тип — определяет возвращаемый тип лямбда-выражения.

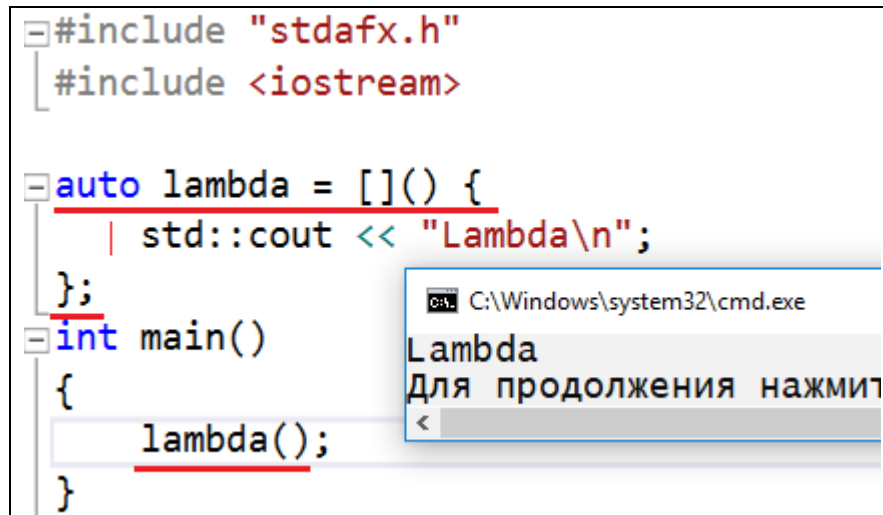
Если у лямбда-выражения нет возвращаемого значения, то по умолчанию устанавливается **void**.

Если в лямбда-выражении есть return, то компилятор вычисляет тип возвращаемого значения автоматически на основании return-выражения.

Если же в лямбда-выражении присутствует if или switch (или другое ветвление) и, соответственно, несколько return, то требуется явное указание конечного возвращаемого типа в виде: -> тип_возвращаемого_параметра, перед началом тела функции.

Тело лямбда-выражения (обязательно).

Пример:



```
#include "stdafx.h"
#include <iostream>

auto lambda = []() {
    | std::cout << "Lambda\n";
};

int main()
{
    lambda();
}
```

где

- | | |
|------------------------------|-------------------------------------|
| auto myLambda = []() {тело}; | - именованное лямбда- выражение |
| myLambda (); | - вызов именованного лямбда-функции |

Более сложное применение:

```
#include "stdafx.h"
#include <algorithm>
#include <iostream>
#include <vector>

int main()
{
    setlocale(LC_ALL, "Russian");
    std::vector<int> v{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    // определить количество целых чисел в std::vector равных заданному.
    int target1 = 3;
    int num_items1 = std::count(v.begin(), v.end(), target1);
    std::cout << "число: " << target1 << " количество: " << num_items1 << '\n';
    // лямбда-выражение для подсчета элементов, кратных 3.
    int num_items3 = std::count_if(v.begin(), v.end(), [](int i) {return i % 3 == 0; });
    std::cout << "количество элементов, кратных 3: " << num_items3 << '\n';
    std::cout << std::endl << " Лямбда, захват переменных" << std::endl;
    // лямбда-выражение, захват переменных
    for (auto i : v) [i]() {if (i % 3 == 0) std::cout << i << " "; }();

    std::cout << std::endl << " Лямбда с параметрами" << std::endl;
    // Вывод значений кратных 3. Значения будем передавать как параметр у обычной функции
    for (auto i : v) [(auto i) {if (i % 3 == 0) std::cout << i << " "; }(i);

    std::cout << std::endl << " Лямбда без параметров" << std::endl;
    // Вывод значений кратных 3.
    // Значения передаются через захват переменных при этом пропустим скобки для добавления па
    for (auto i : v) [i] {if (i % 3 == 0) std::cout << i << " "; }();
}
```

C:\Windows\system32\cmd.exe
число: 3 количество: 1
количество элементов, кратных 3: 3
Лямбда, захват переменных
3 6 9
Лямбда с параметрами
3 6 9
Лямбда без параметров
3 6 9 Для продолжения нажмите любую клавишу . . .

Начиная с C++14 можно использовать параметры по умолчанию:

```
#include "stdafx.h"
#include <iostream>
int main()
{
    setlocale(LC_ALL, "Russian");
    int total = 5;
    int addict = 1;
    auto lam = [&, addict](int coeff = 5)mutable
    {
        total += addict*coeff;
        addict += 22; // переменную можно изменять, но она ограничена телом лямбда-функции
                     // и сохраняется в ней
        std::cout << "(lambda) addict = " << addict << std::endl;
    };
    std::cout << "(main) addict до lambda = " << addict << std::endl;
    lam();
    std::cout << "(main) addict после lambda = " << addict << std::endl;
}
```

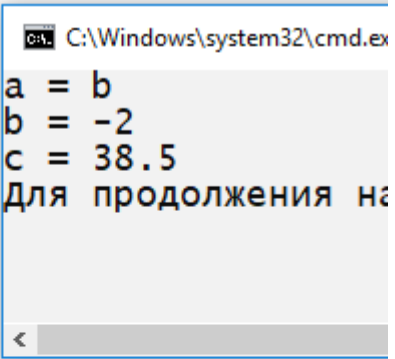
C:\Windows\system32\cmd.exe
(main) addict до lambda = 1
(lambda) addict = 23
(main) addict после lambda = 1
Для продолжения нажмите любую клавишу . . .

Лямбда-функции можно использовать вместо шаблонных функций:

```
#include "stdafx.h"
#include <iostream>

int main()
{
    auto add = [](auto x, auto y) -> char {
        return x + y;
    };
    auto sub = [](auto x, auto y) {
        return x - y;
    };
    auto mul = [](auto x, auto y) {
        return x * y;
    };

    auto a = add('B', 32);
    auto b = sub(5, 7);
    auto c = mul(5.5, 7);
    std::cout << "a = " << a << std::endl;
    std::cout << "b = " << b << std::endl;
    std::cout << "c = " << c << std::endl;
}
```



C:\Windows\system32\cmd.exe
a = b
b = -2
c = 38.5
Для продолжения на