МИНИСТЕРСТВО РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

«КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ»

РЕФЕРАТ

по дисциплине: Основы алгоритмизации и программирования

на тему: Лямбда-выражения в C++

Выполнил:

студент 1 курса 7 группы

факультет ИТ

Кишко Иван Петрович

Проверил:

доцент, к.т.н.

Белодед Николай Иванович

Минск 2020

Оглавление

[Цели 3](#_Toc38541473)

[Теория 3](#_Toc38541474)

[Вызов лямбда-выражения 3](#_Toc38541475)

[Маска переменных 4](#_Toc38541476)

[Захват по значению 5](#_Toc38541477)

[Захват по ссылке 6](#_Toc38541478)

[Захват this 6](#_Toc38541479)

[Список параметров 7](#_Toc38541480)

[Ключевое слово 8](#_Toc38541481)

[Исключения 9](#_Toc38541482)

[Возвращаемый тип 9](#_Toc38541483)

[Вывод 12](#_Toc38541484)

# 

# Цели

Узнать, что такое лямбда-выражение, познакомиться с синтаксисом, узнать особенности реализации самих выражений и применить их на практике.

# Теория

*Лямбда-выражения(функции)* - безымянные локальные функции, которые можно создавать прямо внутри какого-либо выражения.

Общий синтаксис лямбда-функций:

[маска\_переменных](список\_параметров)ключевое\_слово throw()->возвращаемый\_тип

{

/\*тело лямбда-выражения\*/

};

Обязательно наличие:

1. Маски переменных (хотя бы пустой)
2. Тела лямбда-выражения

Все остальные конструкции являются необязательными

Таким образом простейшее лямбда-выражение выглядит следующим образом:

#include <iostream>

using namespace std;

int main()

{

[]{};

}

## Вызов лямбда-выражения

Существует 2 способа вызова лямбда-выражения

1. Вызов сразу после описания с помощью ( )
2. С помощью вспомогательной переменной

Первый способ вызов:

#include <iostream>

using namespace std;

int main()

{

[] {cout << "Hello,world";}();

}

Результат:



Второй способ:

#include <iostream>

using namespace std;

int main()

{

auto ptr = [] {cout << "Hello,world"; };

ptr();

}

Напомню, что ключевое слово **auto** при инициализации переменной может использоваться вместо типа переменной, чтобы сообщить компилятору, что он должен присвоить тип переменной исходя из инициализируемого значения. В данном случае переменная ptr является объектом с перегруженным оператором ().

При пустых [ ] можно использовать и такой вариант:

#include <iostream>

using namespace std;

int main()

{

void (\*ptr)()= []{cout << "Hello,world"; }; //указатель на функцию

ptr ();

}

Результат:

Маска переменных (обязательна)

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:

[ ] – без захвата переменных из внешней области видимости

[=] – все переменные захватываются по значению

[&] – все переменные захватываются по ссылке

[x,&y] – захват x по значению, а y по ссылке

[&,x] – захват x по значению, а всех остальных переменных по ссылке

[=,&x] – захват x по ссылке, а всех остальных переменных по значению

[this] - захватывает указатель **this** по значению

### Захват по значению

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int year = 2020;

int month = 4;

[=] {cout << "Сейчас " << year << " год и " << month << " месяц"; }();

}

Результат:



Захват по значению != передаче параметра по значению

При выполнении следующего кода возникнет ошибка компиляции:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int price = 100;

float increase = 0.1;

[=] {

price += price \* increase;

cout << "Новая цена = " << price<<endl;

}();

return 0;

}

А все потому, что захваченная переменная по умолчанию является константой. Есть способ, который позволяет изменить эту переменную, но об этом позже.

### Захват по ссылке

Захват по ссылке аналогичен передаче параметра по ссылке (т.е. при изменении захваченной по ссылке переменной, она изменяется и за пределами лямбды):

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int price = 100;

float increase = 0.1;

[&] {

price += price \* increase;

cout << "Новая цена = " << price <<endl;

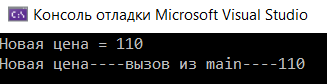
}();

cout << "Новая цена----вызов из main----" << price << endl;

return 0;

}

Результат:



### Захват this

#include <iostream>

using namespace std;

struct Student

{

char name[20];

char surname[20];

unsigned char cours;

void msg()

{

cout << "Вызвалась функция msg\n";

}

void print()

{

[this] {this->msg();}();//теперь лямбда-выражению доступны все переменные и методы структуры

}

};

int main()

{

setlocale(LC\_ALL, "ru");

Student student;

student.print();

}

Результат:



Список параметров (необязательно)

Список параметров лямбда-выражения аналогичен списку параметров для обычных функций. Ниже приведен пример аналогии лямбда-выражения и функции.

#include <iostream>

void sum(int a, int &b)

{

std::cout << "вызов функции sum\n";

std::cout << "a = " << a << " b = " << b << std::endl;

std::cout <<"сумма = "<< a + b << std::endl;

std::cout <<"a = "<<a<<", измененное b = "<<++b << std::endl;//инкрементируем b

std::cout << "====================" << std::endl;

}

int main()

{

setlocale(LC\_ALL, "ru");

int x = 15,y=23;

auto ptr=[](int a, int &b) {

std::cout << "вызов лямбды\n";

std::cout << "a = " << a << " b = " << b << std::endl;

std::cout <<"сумма = "<<a + b << std::endl;

std::cout << "a = " << a << ", измененное b = "

<< ++b <<std::endl;//инкрементируем b

};

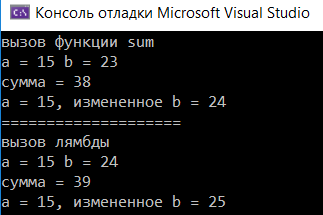
sum(x, y);

ptr(x, y);

return 0;

}

Результат:



Ключевое слово (необязательно)

**mutable** **—** использование mutable позволяет модифицировать константные параметры, переданные **по значению.**

Вспомним, что при выполнении нижеприведенного кода компилятор выдает ошибку

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int price = 100;

float increase = 0.1;

[=] {

price += price \* increase;

cout << "Новая цена = " << price<<endl;

}();

return 0;

}



Модифицируем код с помощью ключевого слово **mutable**

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int price = 100;

float increase = 0.1;

[=]()mutable {

price += price \* increase;

cout << "Новая цена = " << price << endl;

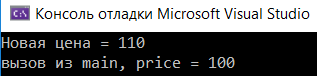
}();

cout << "вызов из main, price = " << price;

return 0;

}

Результат:



Исключения (необязательно)

**throw() —** спецификация исключений, то есть лямбда-выражения   
могут выбрасывать исключения, указанные в круглых скобках ().Если скобки пустые, то исключения не выбрасываются, а попытка выбросить исключение при пустых скобках приводит к неопределенному поведению. В финальном варианте стандарта С++ 11 throw-спецификации объявлены устаревшими.

При указании ключевого слово **noexcept** вместо **throw()** лямбда-выражение теряет возможность выбрасывать исключения (рекомендуется).

#include <iostream>

int main()

{

setlocale(LC\_ALL, "ru");

try {

[]()noexcept(true) {throw "вызвано исключение"; }();

}

catch (const char\* str)

{

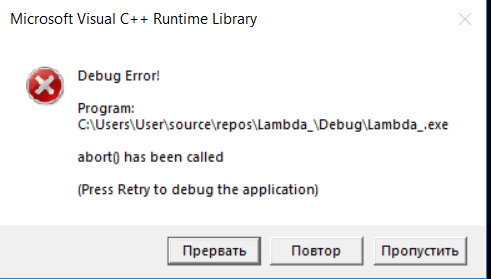
std::cout << str << '\n';

}

return 0;

}

Результат:



Возвращаемый тип(необязательно)

**– >**(стрелка)определяет возвращаемый тип лямбда-выражения.

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

auto a = []()->int {return 5; }();

cout << typeid(a).name() << endl; //функция, позволяющая узнать тип данных переменной

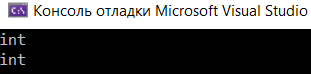
auto b = []()->int {return 5.6; }();//преобразовывается в return (int)5.6;

cout << typeid(b).name() << endl;

return 0;

}

Результат:



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.

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

auto a = [](){return 5; }();

cout << typeid(a).name() << endl; //функция, позволяющая узнать тип данных переменной

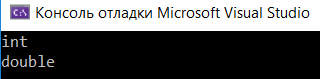
auto b = [](){return 5.6; }();//double

cout << typeid(b).name() << endl;

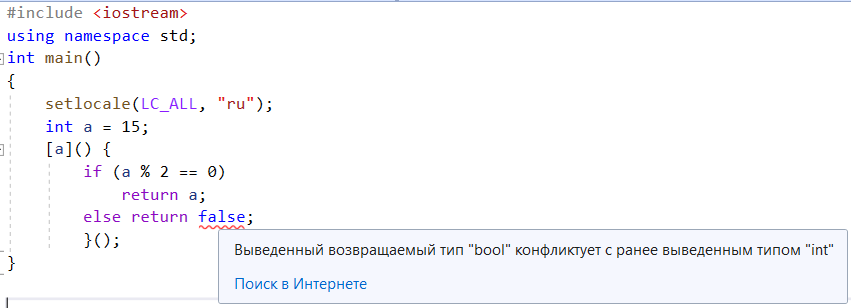
return 0;

}

Результат:



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



Укажем тип возвращаемого значения явно:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int a = 15;

auto ptr=[a]()->int{

if (a % 2 == 0)

return a;

else return false;//то же самое что и return (int)false;

}();

cout<<ptr<<" "<< typeid(ptr).name() << endl;

return 0;

}

Результат:



Лямбда также может возвращать лямбда выражение:

#include <iostream>

int main()

{

setlocale(LC\_ALL, "ru");

int a = 13, b = -3;

auto ptr = [] {

return [](auto a,auto b) {std::cout << "a + b = "<<a+b; };

}();

ptr(a,b);

}

Результат:



# Вывод

Лямбды выражения хороши и позволяют улучшить читабельность, но они не заменяют обычные функции для всех случаев. Используйте обычные функции для нетривиальных случаев.