**ЛЕКЦИЯ 5. ФУНКЦИИ. ПРОДОЛЖЕНИЕ**

Содержание

[1. ЛЯМБДА-ВЫРАЖЕНИЕ 1](#_Toc178679401)

[1.1 Понятие и синтаксис лямбда-выражений 1](#_Toc178679402)

[1.2 Примеры лямбда-выражений 2](#_Toc178679403)

[2. АЛГОРИТМЫ 5](#_Toc178679404)

[2.1 Алгоритмы минимум и максимум 5](#_Toc178679405)

[2.2 Алгоритмы поиска и копирования 6](#_Toc178679406)

[2.3 Алгоритмы сортировки, объединения и трансформации 7](#_Toc178679407)

[2.4 Пример с использованием алгоритмов 8](#_Toc178679408)

# ЛЯМБДА-ВЫРАЖЕНИЕ

## Понятие и синтаксис лямбда-выражений

Лямбда-выражения представляют более краткий компактный синтаксис для определения объектов-функций. Формальный синтаксис лямбда-выражения:

**[] (параметры) { действия }**

Лямбда-выражение начинается с квадратных скобок. Затем, как в обычной функции, в круглых скобках идет перечисление параметров - типы и их имена. Также начиная со стандарта C++14 для параметров можно указывать значения по умолчанию. За списком параметров, как и в обычной функции, в фигурных скобках помещаются действия, выполняемые лямбда-выражением.

**int x = 100, y = 200;**

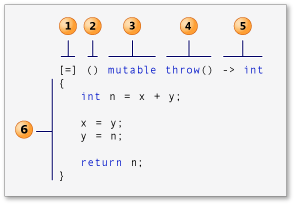
**auto print = [&] { // Capturing object by reference**

**std::cout << ":" << x << "," << y << std::endl;**

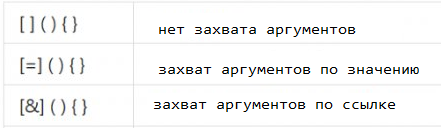
**};**

**print();**

На этом рисунке показаны части лямбда-синтаксиса:



1. Предложение capture (также известное как лямбда-вводное выражение в спецификации C++).



**auto f = [x]() { return x \* 9; }; // OK, 'x' is "captured" by value**

**auto f = [&x]() { return x++; }; // OK, 'x' is "captured" by reference**

1. Список параметров (дополнительные. Также называется лямбда-декларатором
2. Изменяемая спецификация (дополнительные).
3. Спецификация исключений (дополнительные).
4. Тип возвращаемого значения (дополнительные).

**[](int a, int b) -> double {return a + b;}**

1. Лямбда-тело.

## Примеры лямбда-выражений

**Пример 1. Лямбда-выражение вычисления суммы двух значений**

**auto sum{ [](int a, int b) {return a + b; } };**

**// вызываем лямбда-выражение**

**std::cout << sum(10, 23) << std::endl; // 33**

**// присваиваем результат лямбда-выражения переменной**

**int result{ sum(1, 4) };**

**Пример 2. Предложение capture**

**#include <functional>**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int i = 3;**

**int j = 5;**

**// The following lambda expression captures i by value and**

**// j by reference.**

**function<int(void)> f = [i, &j] { return i + j; };**

**// Change the values of i and j.**

**i = 22;**

**j = 44;**

**// Call f and print its result.**

**cout << f() << endl; //47**

**}**

**Пример 3. Вложенные лямбда-выражения**

**#include <iostream>**

**using namespace std;**

**int main() {**

**// The following lambda expression contains a nested lambda**

**// expression.**

**int timestwoplusthree = [](int x) {return [](int y) { return y \* 2;}**

**(x)+3; }(5);**

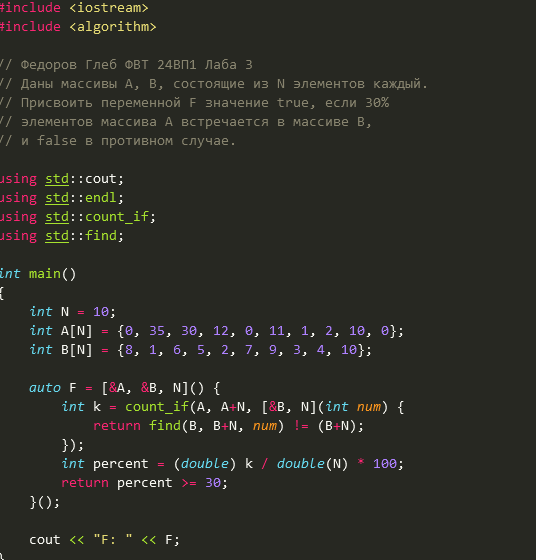
**// Print the result.**

**cout << timestwoplusthree << endl; // результат: 13**

**}**

Внутреннее лямбда-выражение умножает его аргумент на 2 и возвращает результат. Внешнее лямбда-выражение вызывает внутреннее лямбда-выражение с использованием его аргумента и добавляет к результату 3

**Пример 4. Лямбда-выражение как параметр функции**



**Пример 5. Лямбда-выражение как параметр функции**

**#include <iostream>**

**#include <algorithm>**

**#include <cmath>**

**void abssort(int x[], unsigned n);**

**int main() {**

**int numbers[]{ 1, 4, 3, 5, 2 };**

**for (const auto& n : numbers) {**

**std::cout << n << "\t";**

**}**

**std::cout << std::endl;**

**abssort(numbers, 5);**

**for (const auto& n : numbers) {**

**std::cout << n << "\t";**

**}**

**std::cout << std::endl;**

**return 0;**

**}**

**void abssort(int x[], unsigned n) {**

**std::sort(x, x + n,**

**[](int a, int b) {// лямбда-выражение**

**return (std::abs(a) < std::abs(b));**

**}**

**);**

**}**

**Пример 6.Универсальное л*ямбда-выражение***

**#include <iostream>**

**int main()**

**{**

**auto print = [](const auto& value) {std::cout << value << std::endl; };**

**print("Hello");**

**print(4);**

**print(45.6789);**

**}**

# АЛГОРИТМЫ

**Алгоритм** — это функция, которая производит некоторые действия над элементами контейнера (контейнеров). Алгоритмы в STL не являются методами классов и даже не являются дружественными функциями по отношению к контейнерам.

В стандарте языка C++ алгоритмы — это **независимые шаблонные функции**.

Их можно использовать при работе как с обычными массивами C++, контейнерами и с пользовательскими классами. Все алгоритмы STL можно разделить на четыре категории:

* немодифицирующие операции с последовательностями;
* модифицирующие операции с последовательностями;
* алгоритмы, связанные с сортировкой;
* алгоритмы работы с множествами и пирамидами;

Примеры алгоритмов приведены в таблице 1.

Таблица 1 – Некоторые типичные алгоритмы STL

|  |  |
| --- | --- |
| **Алгоритм** | **Назначение** |
| **find** | Возвращает первый элемент с указанным значением |
| **count** | Считает количество элементов, имеющих указанное значение |
| **equal** | Сравнивает содержимое двух контейнеров и возвращает true, если все соответствующие элементы эквивалентны |
| **search** | Ищет последовательность значений в одном контейнере, которая соответствует такой же последовательности в другом |
| **copy** | Копирует последовательность значений из одного контейнера в другой (или в другое место того же контейнера) |
| **swap** | Обменивает значения, хранящиеся в разных местах |
| **iter\_swap** | Обменивает последовательности значений, хранящиеся в разных местах |
| **fill** | Копирует значение в последовательность ячеек |
| **sort** | Сортирует значения в указанном порядке |
| **merge** | Комбинирует два сортированных диапазона значений для получения наибольшего диапазона |
| **accumulate** | Возвращает сумму элементов в заданном диапазоне |
| **for\_each** | Выполняет указанную функцию для каждого элемента контейнера |

## Алгоритмы минимум и максимум

**Пример 7. Функции минимума и максимума**

**#include <iostream>**

**#include <algorithm>**

**int main()**

**{**

**int numbers[]{ 1, 2, 3, 4, 5 };**

**std::cout<<"Min:"<<\*std::min\_element(numbers, numbers+5) << std::endl;**

**std::cout<<"Max:"<<\*std::max\_element(numbers, numbers+5) << std::endl;**

**// функция std::minmax\_element()возвращает результат**

**// в виде объекта std::pair<iterator, iterator>**

**auto p = std::minmax\_element(numbers, numbers+5);**

**std::cout << "Min: " << \*p.first << std::endl;**

**std::cout << "Max: " << \*p.second << std::endl;**

**}**

## Алгоритмы поиска и копирования

**Пример 8. Функции поиска и копирования**

**#include <iostream>**

**#include <string>**

**#include <algorithm> //для алгоритмов**

**#include <functional>**

**using namespace std;**

**void print(int i) { cout << i << " "; }**

**bool isDon(string name) {return name == "Дмитрий";}**

**int main() {**

**setlocale(LC\_ALL, "Russian");**

**int arr[] = { 22, 33, 33, 66, 55, 88, 77, 11 };**

**int\* ptr;**

**// find: найти первый объект, значение которого равно данному**

**ptr = find(arr, arr + 8, 33); //найти первое вхождение 33**

**cout << "Первый объект со значением 33 найден в позиции "**

**<< (ptr - arr);**

**cout << endl<<endl;**

**// find\_if: ищет первое вхождение объекта по заданному условию**

**string names[] = { "Сергей", "Татьяна", "Елена", "Дмитрий",**

**"Михаил", "Владимир" };**

**string\* sptr;**

**sptr = find\_if(names, names + 6, isDon);**

**if (sptr == names + 6)**

**cout << "Дмитрия нет в списке.\n";**

**else**

**cout << "Дмитрий записан в позиции " << (sptr - names)**

**<< " в списке";**

**cout << endl<<endl;**

**//count: считает количество объектов, имеющих данное значение**

**int n = count(arr, arr + 8, 33); //сколько раз встречается 33**

**cout << "Число 33 встречается " << n << " раз в массиве ";**

**cout << endl<<endl;**

**// search: ищет последовательность, заданную одним**

**//контейнером, в другом контейнере**

**int pattern[] = { 33, 66, 55 };**

**ptr = search(arr, arr + 8, pattern, pattern + 3);**

**if (ptr == arr + 8) // если после последнего**

**cout << "Совпадения не найдено\n";**

**else**

**cout << "Совпадение в позиции " << (ptr - arr);**

**cout << endl<<endl;**

**// копирование массивов**

**const int N = 5;**

**int copy\_numbers[N];**

**copy(numbers, numbers+N, copy\_numbers);**

**for (int& i : copy\_numbers) {**

**cout << i << ' ';**

**}**

**cout << endl;**

**copy\_n(numbers, N, copy\_numbers);**

**for (int& i : copy\_numbers)**

**cout << i << ' ';**

**cout << endl;**

**}**

## Алгоритмы сортировки, объединения и трансформации

**Пример 9. Функции сортировки и трансформации**

**#include <iostream>**

**#include <algorithm> //для алгоритмов**

**#include <functional>**

**using namespace std;**

**void print(int i) { cout << i << " "; }**

**int main()**

**{**

**//sort: сортировка**

**sort(arr, arr + 8);**

**//for (int j = 0; j < 8; j++) // вывести отсортированный**

**// cout << arr[j] << ' '; //массив**

**for\_each(arr, arr + 8, print);**

**cout << endl<<endl;**

**//sort:сортировка с заданной функцией сравнения**

**sort(numbers, numbers+5,**

**[](int a, int b) {// лямбда-выражение**

**return (std::abs(a) < std::abs(b));**

**}**

**);**

**for (const auto& n : numbers) {**

**cout << n << "\t";**

**}**

**cout << endl;**

**// merge: соединение двух контейнеров в третий**

**int src1[] = { 2, 3, 4, 6, 8 };**

**int src2[] = { 1, 3, 5 };**

**int dest[8];**

**merge(src1, src1 + 5, src2, src2 + 3, dest);**

**copy(dest, dest + 8, ostream\_iterator<int>(cout, " "));**

**cout << endl<<endl;**

**// transform: преобразует каждый элемент контейнера**

**// и помещает результат в другой контейнер (или в тот же)**

**transform(arr, arr + 8, arr, dest, plus<>{});**

**copy(dest, dest + 8, ostream\_iterator<int>(cout, " "));**

**cout << endl;**

**return 0;**

**}**

## Пример с использованием алгоритмов

**Пример 10. Обобщенный пример использования функций STL**

**#include <iostream>**

**#include <string>**

**#include <algorithm> //для алгоритмов**

**using namespace std;**

**void print(int i) { cout << i << " "; }**

**bool isDon(string name) {return name == "Дмитрий";}**

**bool is\_even(int n) { return n % 2 == 0; }**

**int main()**

**{**

**setlocale(LC\_ALL, "Russian");**

**int arr[] = { 22, 33, 33, 66, 55, 88, 77, 11 };**

**int\* ptr;**

**// find: найти первый объект, значение которого равно данному**

**ptr = find(arr, arr + 8, 33); //найти первое вхождение 33**

**cout << "Первый объект со значением 33 найден в позиции " <<**

**(ptr - arr);**

**cout << endl<<endl;**

**// find\_if: ищет первое вхождение объекта по заданному условию**

**string names[] = { "Сергей", "Татьяна", "Елена", "Дмитрий",**

**"Михаил", "Владимир" };**

**string\* sptr;**

**sptr = find\_if(names, names + 6, isDon);**

**//sptr = find\_if(names, names + 5, [](string s){**

**//return s == "Дмитрий"; } );**

**if (sptr == names + 6)**

**cout << "Дмитрия нет в списке.\n";**

**else**

**cout << "Дмитрий записан в позиции " << (sptr - names)<<**

**" в списке";**

**cout << endl<<endl;**

**//count: считает количество объектов, имеющих данное значение**

**int n = count(arr, arr + 8, 33); // сколько раз встречается 33**

**cout << "Число 33 встречается " << n << " раз в массиве ";**

**cout << endl<<endl;**

**// search: ищем последовательность, заданную одним**

**//контейнером, в другом контейнере**

**int pattern[] = { 33, 66, 55 };**

**ptr = search(arr, arr + 8, pattern, pattern + 3);**

**if (ptr == arr + 8) // если после последнего**

**cout << "Совпадения не найдено\n";**

**else**

**cout << "Совпадение в позиции " << (ptr - arr);**

**cout << endl<<endl;**

**//sort: сортировка**

**sort(arr, arr + 8);**

**//for (int j = 0; j < 8; j++) // вывести отсортированный**

**// cout << arr[j] << ' '; //массив**

**for\_each(arr, arr + 8, print);**

**//copy(arr, arr + 8, ostream\_iterator<int>(cout, " "));**

**cout << endl<<endl;**

**// merge: соединение двух контейнеров в третий**

**int src1[] = { 2, 3, 4, 6, 8 };**

**int src2[] = { 1, 3, 5 };**

**int dest[8];**

**merge(src1, src1 + 5, src2, src2 + 3, dest);**

**//for\_each(dest, dest + 8, [](int i) {cout << i << " "; });**

**copy(dest, dest + 8, ostream\_iterator<int>(cout, " "));**

**cout << endl<<endl;**

**// transform: преобразует каждый элемент контейнера**

**// и помещает результат в другой контейнер (или в тот же)**

**transform(arr, arr + 8, dest, [](int i) {return i \* 2;});**

**copy(arr, arr + 8, ostream\_iterator<int>(cout, " "));**

**cout << endl;**

**transform(arr, arr + 8, arr, dest, plus<>{});**

**copy(dest, dest + 8, ostream\_iterator<int>(cout, " "));**

**cout << endl;**

**int numbers[]{ 1, -3, 2, -4, 5 };**

**for (const auto& n : numbers) {**

**cout << n << "\t";**

**}**

**cout << '\n';**

**//копирует четные элементы numbers в массив copy\_numbers**

**int copy\_numbers[5];**

**auto copyind = copy\_if(begin(numbers), end(numbers),**

**begin(copy\_numbers), is\_even);**

**for (int i = 0; i < copyind - copy\_numbers; i++)**

**cout << copy\_numbers[i] << " ";**

**cout <<endl;**

**//копирует четные элементы numbers в выходной поток**

**copy\_if(begin(numbers), end(numbers),**

**ostream\_iterator<int>(cout, " "),**

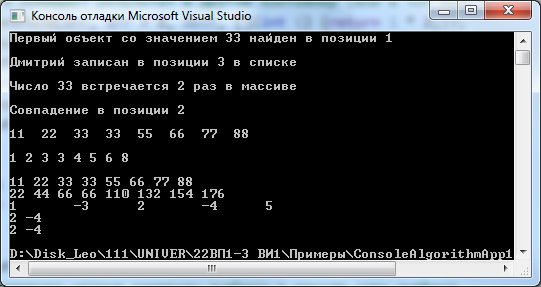
**[](int x) { return x % 2 == 0; }**

**);**

**cout << '\n';**

**return 0;**

**}**



# РЕКУРСИВНЫЕ ФУНКЦИИ

## Понятие рекурсивной функции

*Рекурсивная функция* – это такая функция, которая в своем коде вызывает сама себя непосредственно либо косвенно через другую функцию

При каждом рекурсивном вызове запоминаются предыдущие значения внутренних локальных переменных и полученных параметров функции. Чтобы следующий шаг рекурсивного вызова отличался от предыдущего, значение, как минимум одного из параметров функции, должно быть изменено. Остановка процесса рекурсивных вызовов функции происходит, когда изменяемый параметр достиг некоторого конечного значения.

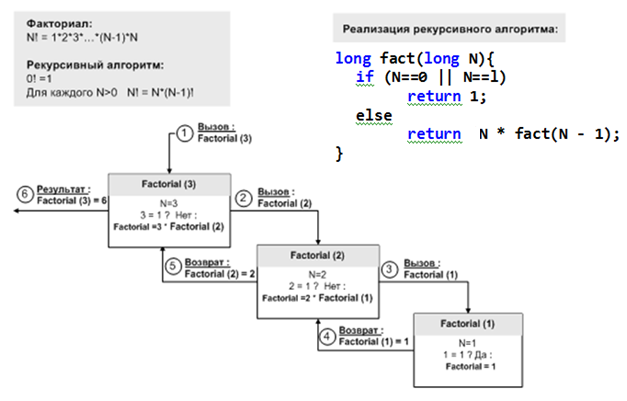
Чтобы циклический процесс преобразовать в рекурсивный, нужно определить:

* условие прекращения последовательности рекурсивных вызовов функции;
* формулу следующего элемента, который используется в рекурсивном процессе;
* список параметров, которые передаются в рекурсивную функцию. Один из параметров обязательно есть итератор (счетчик), изменяющий свое значение. Другие параметры являются дополнительными.

## Схема рекурсии вычисления факториала

Рассмотрим рекурсию на классическом примере вычисления факториала:

**Способ 1:**



**Способ 2:**

**#include <iostream>**

**using namespace std;**

**long fact1(long, long);**

**int main()**

**{**

**const long N = 5;**

**long i = 1;**

**cout << fact1(N, i) << endl;**

**return 0;**

**}**

**// вычисление факториала**

**long fact1(long n, long i)**

**{**

**if (i == n)**

**return n; // условие завершения рекурсивного процесса**

**else**

**return i \* fact1(n, i + 1); // переход к следующему числу i+1**

**}**

## Примеры простых рекурсивных функций

**Пример 11. Вычисление чисел Фибоначчи**

Числа Фибоначчи**: a0=0; a1=1; аn = an-1 + an-2; n>=2**

**#include <iostream>**

**using namespace std;**

**int fib(int);**

**int main(){**

**const int N=10;**

**for (int i = 0; i < N; ++i)**

**cout << fib(i) << " ";**

**cout << endl;**

**return 0;**

**}**

**int fib(int n) {**

**if (n == 0)**

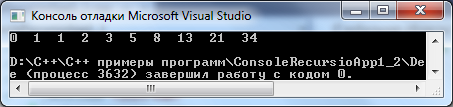
**return 0;**

**if (n == 1)**

**return 1;**

**if(n>=2)**

**return fib(n - 1) + fib(n - 2);**

**}**

**Пример 12. Вычисление суммы элементов одномерного массива**

**#include <iostream>**

**using namespace std;**

**int sum(int, int [], int );**

**int main(){**

**const int N = 5;**

**int a[N] = { 1, 2, 3, 4, 5};**

**cout << sum(0, a, N) << endl;**

**return 0;**

**}**

**int sum(int i, int x[], int n) { //сумма элементов массива**

**//i - итератор**

**if (i == n - 1)**

**return x[i];**

**else**

**return x[i] + sum(i + 1, x, n);**

**}**

**Пример 13. Вычисление суммы знакопеременного ряда**

Вычислить сумму членов ряда **x – x^3/3! + x^5/5! – х^7/7! + ...** с точностью **eps** < 0.

Будем вычислять каждое слагаемое (**a**) через предыдущее:

**a = a\*(– x^2)/((2\*n)\*(2\*n+1)), где n = 1, 2, 3, 4, 5,...**

Рекурсивная функция должна принимать следующие параметры:

**a** – предыдущее слагаемое

**x** – аргумент функции

**eps** – точность

**n** – итератор

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**double sum(double, double, double, int);**

**int main() {**

**double x = 1;**

**cout << x + sum(x, x, 0.01, 1) << endl;**

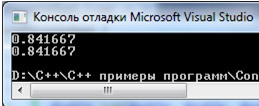
**cout << x - x \* x \* x / 6 + x \* x \* x \* x \* x / 120 << endl;**

**return 0;**

**}**

**double sum(double a, double x, double eps, int n) { //сумма ряда**

**a = a \* (-x \* x) / ((2 \* n) \* (2 \* n + 1));**

** if (abs(a) < eps)**

**return a;**

**else**

**return a + sum(a, x, eps,**

**n + 1);**

**}**

**Пример 14. Backtracking. Задача о рюкзаке**

**Постановка задачи:**

Имеется N предметов. Известны их вес и стоимость. Какие предметы надо положить в рюкзак, чтобы их вес не превышал допустимый, а стоимость была максимальна

**Алгоритм:**

1. Поместить **i** – й предмет в рюкзак
2. Пересчитать общий суммарный вес и суммарную стоимость (с учетом добавленного предмета)
3. Если полученный вес не превышает допустимый, тогда
4. Если суммарная стоимость больше текущей максимальной, то

* вариант заполнения рюкзака запоминается как наилучший
* максимальная текущая стоимость получает значение суммарной стоимости
* максимальный суммарный вес получает значение суммарного веса

1. Рекурсивно вызываем функцию с параметром i – й для рассмотрения новых вариантов
2. После возвращения из функции предмет вынимается из рюкзака, суммарная стоимость и суммарный вес уменьшаются на параметры вынутого предмета
3. Продолжается перебор вариантов с рассмотрения **i+1**-го предмета

**#include <iostream>**

**using namespace std;**

**const int N = 5;**

**int w[N] = { 1, 2, 3, 4, 5 }; //вес**

**int c[N] = { 1, 3, 5, 5, 1 }; //стоимость**

**int u[N], //массив выбранных предметов: 1-выбран, 0-не выбран**

**maxU[N], //результат**

**maxC = 0, //максимальная стоимость**

**maxW=0, //максимальный вес**

**totalC = 0, //текущая суммарная стоимость**

**totalW = 0, //текущий суммарный вес**

**bagSize = 8;//допустимый вес**

**void AddNew(int);**

**int main()**

**{**

**AddNew(-1);**

**cout << "maxC: "<<maxC << endl;**

**cout << "maxW: "<<maxW << endl;**

**for (int i = 0; i < N; ++i)**

**if(maxU[i]==1)**

**cout << i<<" ";**

**cout << endl;**

**return 0;**

**}**

**void AddNew(int last)**

**{**

**for (int i = last + 1; i < N; i++) {**

**if ((u[i] == 0) && (totalW + w[i] < bagSize)) {**

**u[i] = 1;**

**totalC += c[i];**

**totalW += w[i];**

**if (totalC > maxC) {**

**for (int j = 0; j < N; ++j)**

**maxU[j] = u[j];**

**maxC = totalC;**

**maxW = totalW;**

**}**

**AddNew(i);**

**totalC -= c[i];**

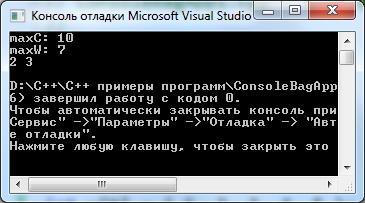
**totalW -= w[i];**

**u[i] = 0;**

**}**

**}**

**}**

****