Лямбда функции

Лямбда функции

```
#include <iostream>
#include <vector>


int main()
{
    std::vector<int> v = {5, -1, 4, -2, -3};
    std::sort(v.begin(), v.end());
    for (int i : v){
        std::cout << i << " ";
    }
}
```

А если хотим более сложную сортировку?

Лямбда функции

```
[ списки захвата ] (аргументы) -> возвращаемое значение
{
        описание метода
}
```

Лямбда-выражение позволяет нам определить анонимную функцию внутри другой функции. Тип, который возвращает лямбда функция называется замыканием

Решение проблемы

```
#include <iostream>
#include <vector>
int main()
{
    std::vector<int> v = {5, -1, 4, -2, -3};
    std::sort(v.begin(), v.end());
    for (int i : v){
        std::cout << i << " ";
    }
    std::cout << std::endl;
    auto cmp = [](int x, int y) {return x*x < y*y;};
    std::sort(v.begin(), v.end(), cmp);
        for (int i : v){
        std::cout << i << " ";
    }
}
```

Лямбда функции

```
template <typename T>
void print_vec(const T& x)
{
    cout << x << " ";
}
```

```
47 94 68 64 77 34 46 4 69 44
```

```
auto f = [](auto value)
{
       print_vec(value);
};
for_each(vec1.begin(), vec1.end(), f);
```

Лямбда функции

Упростим….

```
for_each(vec1.begin(), vec1.end(), [](auto value){cout << value << " "; });
```

Придумаем себе проблему

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


int main() {
    std::vector<int> numbers = {15, 3, 22, 8, 17, 9, 31, 6, 12};
    int min_value = 10;
    std::cout << "Исходный массив: ";
    for (int n : numbers) std::cout << n << " ";
    std::cout << std::endl;

    std::sort(numbers.begin(), numbers.end(), [](int a, int b) {
        // Сначала числа >= min_value, потом остальные
        if (a < min_value && b >= min_value) return false;
        if (a >= min_value && b < min_value) return true;
        return a < b; // Внутри групп сортируем обычно
    });

    std::cout << "После сортировки (числа >= " << min_value << " сначала): ";
    for (int n : numbers) std::cout << n << " ";
    std::cout << std::endl;
}
```

Списки захвата

Пустой список захвата `[]`

```
int x = 10;
auto lambda = []() {
    // return x; // CE
    return 42;
};
```

Захват по значению `[=]`

```
int a = 5, b = 10;
auto lambda = [=]() {
    return a + b; // Копии a и b
};
// a и b остаются неизменными
```

Смешанный захват

```
int x = 1, y = 2, z = 3;
auto lambda = [x, &y]() {
    // x - по значению, y - по ссылке
    y = x + y; // Изменяет y, но не x
    return y;
};
```

Захват по ссылке `[&]`

```
int a = 5, b = 10;
auto lambda = [&]() {
    a = 15; // Изменяет оригинальные переменные
    return a + b;
};
// a теперь равно 15
```

Списки захвата в лямбда-выражениях — это механизм, который позволяет лямбде получать доступ к переменным из окружающей области видимости.

Решим проблему

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


int main() {
    std::vector<int> numbers = {15, 3, 22, 8, 17, 9, 31, 6, 12};
    int min_value = 10;
    std::cout << "Исходный массив: ";
    for (int n : numbers) std::cout << n << " ";
    std::cout << std::endl;

    std::sort(numbers.begin(), numbers.end(), [min_value](int a, int b) {
        // Сначала числа >= min_value, потом остальные
        if (a < min_value && b >= min_value) return false;
        if (a >= min_value && b < min_value) return true;
        return a < b; // Внутри групп сортируем обычно
    });

    std::cout << "После сортировки (числа >= " << min_value << " сначала): ";
    for (int n : numbers) std::cout << n << " ";
    std::cout << std::endl;
}
```

Еще о лямбдах

```
int main()
{
    [] {
        std::cout << "Hello";
    }();

        [](){}();
        []{}();

};
```

Так можно

Так тоже :)