C++

# Урок 20

# Разбор ДЗ

# Лямбда ф-ии

# Что означает лямбда?

# Лямбда ф-ии

В C++11 и более поздних версиях **лямбда-выражение** — это удобный способ определения анонимного объекта функции (закрытия) в расположении, где он вызывается или передается в качестве аргумента функции.

Как правило, лямбда-выражения **используются** для **инкапсуляции** нескольких строк **кода**, передаваемых алгоритмам или асинхронным функциям.

**Лямбда-выражения** — это краткая форма записи анонимных функторов.

# Лямбда ф-ии

Почему будем использовать:

- удобство функционального стиля;
- написание логики в теле STL ф-ии.

# Список

## Синтаксис

```
auto lambda_name(
    []()
    {
        // тело лямбды
    });
```

[ ] – список захвата;
( ) - аргументы, которые передаются в лямбду функцию.

# Лямбда ф-ии

```
#include <iostream>

auto show_Val([](int val)
              { std::cout << val << std::endl; });

int main()
{
    int a = 42;
    show_Val(a);
}
```

Лямбда выражение вывода

# Задача

**Задача:** напишите простую лямбда-функцию, которая возвращает сумму 2 целочисленных параметров в main().

# Практическое применение

# Где чаще всего применяются лямбда ф-ии?

# Лямбда ф-ии

Конечно же в STL

# Лямбда ф-ии

```cpp
#include <algorithm>
#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 0;
}
```

Использование функтора

# Лямбда ф-ии

```
#include <algorithm>
#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 0;
}
```

Использование лямбда-выражения

# На какой код потратим меньше времени?

# Лямбда ф-ии

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

using namespace std;

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

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

    cout << result << endl;

    return 0;
}
```

Использование лямбда-выражения 2

# Задача

**Задача:** введите/выведите данные в вектор через for_each и лямбда выражение.

# Задача

**Задача:** удалите из вектора все элементы, которые не превосходят введенное с клавиатуры значение.

# Лямбда ф-ии

В примере 2 лямбда играет роль унарного **предиката**, то есть тип возвращаемого значения **bool**, хотя мы нигде этого не указывали.

При наличии **одного return** в лямбда-выражении, компилятор вычисляет тип возвращаемого значения **самостоятельно**.

Если же в лямбда-выражении присутствует if или switch (или другие сложные конструкции), после которых может стоять return, то нужно указывать **самому** то, что вернет лямбда.

**Функции-предикаты** (или функции-вопросы) отвечают на какой-то вопрос и всегда (без исключений!) возвращают либо true, либо false. Пример: isDigit, isAlpha.

# Лямбда ф-ии

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

using namespace std;

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

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

    ostream_iterator<double> outIt(cout, " ");
    copy(destVec.begin(), destVec.end(), outIt);
    cout << endl;

    return 0;
}
```

```
20_16.cpp:25:35: error: inconsistent types 'double' and 'int' deduced for lambda return type
   25 |                         return _n * _n; });
      |                                ~~^~~
```

Использование лямбда-выражения 3

# Лямбда ф-ии

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

using namespace std;

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

    vector<double> destVec;
    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.;
        else
            return _n * _n; });

    for (const auto &data : destVec)
    {
        cout << data << " ";
    }

    return 0;
}
```

Использование лямбда-выражения 4

# Список захвата

# Лямбда ф-ии

Рассмотренные примеры - **анонимные функции**, потому что не хранили никакого промежуточного состояния.

**Лямбда-выражения** в C++ — это анонимные функторы, а значит состояние они хранить могут!

# Лямбда ф-ии

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

using namespace std;

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

    int lowerBound = 0, upperBound = 0;
    cout << "Enter the value range: ";
    cin >> lowerBound >> upperBound;

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

    return 0;
}
```

Листинг список захвата 1

# Лямбда ф-ии

**Лямбда** в процессе компиляции становится **функтором** (объектом), внутри тела которого мы не можем напрямую **использовать переменные**, объявленные в main(), так как это непересекающиеся области видимости.

Внутри тела функтора (происходит тот самый «захват»): конструктор их инициализирует, а внутри operator()() они используются.

# Задача

**Задача:** пройдитесь по всем элементам контейнера и запишите сумму всех значений в переменную sum, используя список захвата.