

# Под капотом: компиляция и линковка

Евгений Белоусов  
Ведущий программист в компании IQTech



# Проверка связи





## Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти



## Поставьте в чат:

-  если меня видно и слышно
-  если нет

# Евгений Белоусов

О спикере:

- Ведущий программист в компании IQTech
- Работает в IT с 2011
- Опыт разработки на C++ более 12 лет



# Вспоминаем прошрое занятие

**Вопрос:** зачем нужен спецификатор final?



# Вспоминаем прошрое занятие

**Вопрос:** зачем нужен спецификатор `final`?

**Ответ:** для того, чтобы запретить  
переопределение виртуальной функции



# Вспоминаем прошрое занятие

**Вопрос:** зачем использовать умные  
указатели?



# Вспоминаем прошрое занятие

**Вопрос:** зачем использовать умные  
указатели?

**Ответ:** для того, чтобы избежать утечек  
памяти



# Вспоминаем прошрое занятие

**Вопрос:** что такое лямбда-функция?





# Вспоминаем прошрое занятие

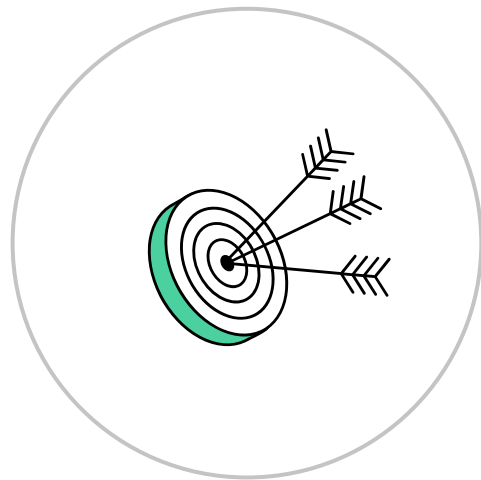
**Вопрос:** что такое лямбда-функция?

**Ответ:** анонимная функция, которую можно вызывать тут же или передавать в качестве параметра



# Цели занятия

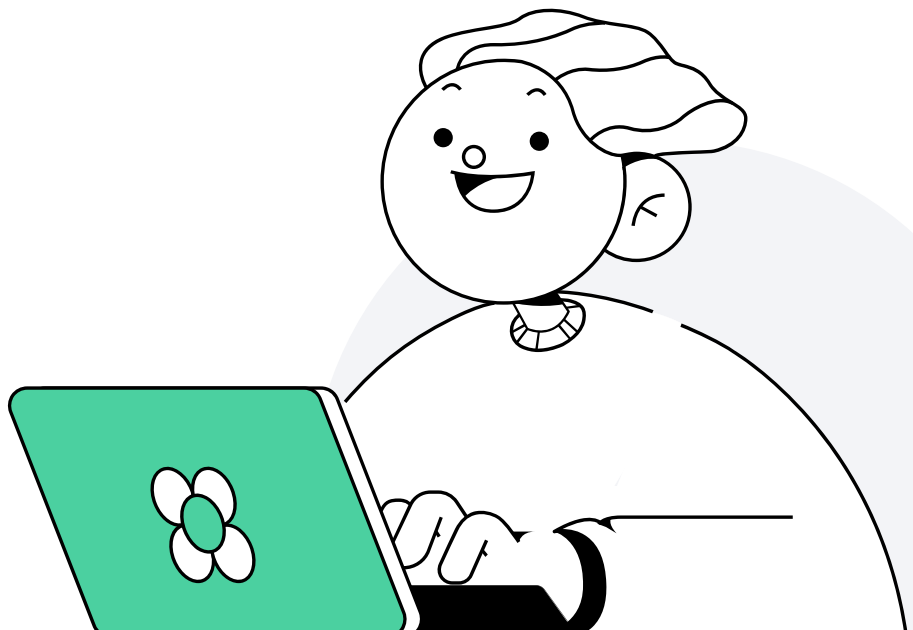
- Узнаем зачем нужно компилировать исходные файлы
- Узнаем из чего состоит процесс компиляции программ на C++
- Узнаем какие бывают ошибки и как их исправлять



# План занятия

- 1 Препроцессинг
- 2 Компиляция
- 3 Ассемблирование
- 4 Линковка
- 5 Итоги
- 6 Домашнее задание

\*Нажми на нужный раздел для перехода

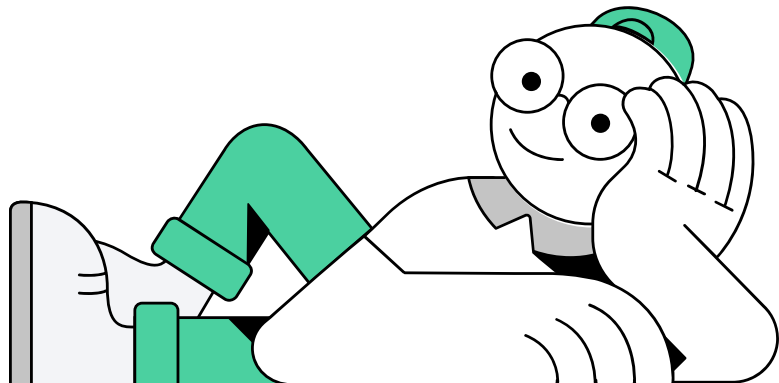


# Зачем нужно компилировать исходные файлы?

Исходный C++ файл - это всего лишь текстовый файл. Его не представляется возможным запустить просто так или использовать как-то еще.

Поэтому каждый исходный текстовый файл нужно скомпилировать:

- в исполняемый файл
- в динамическую библиотеку
- в статическую библиотеку



# Машинный код

Для того чтобы процессор “понял” то, что мы просим его выполнить, надо нашу программу перевести на его язык - машинный код.

ooo

## Языки программирования высокого уровня

**Пример:** сложить содержимое двух разных ячеек и результат поместить в третью.

(пусть адреса ячеек равны 51, F2 и 93)

**Машинный код:**

```
0001 0001 0101 0001
0001 0010 1111 0010
0101 0011 0001 0010
0011 0011 1001 0011
```

**Мнемокод:**

```
LD    R1, 51
LD    R2, F2
ADD   R3, R1, R2
ST    R3, 93
```

**Язык программирования в/у:**  $c=a+b$

# Препроцессинг



1



**Препроцессинг - обработка исходных файлов программы специальной утилитой (текстовым препроцессором)**

# Преппроцессинг

На данном этапе исходные тексты программ изменяются следующим образом:

- Добавление хэдеров в код (`#include`)
- Макроподстановки (`#define`)
- Обработка директив условной компиляции (`#if`, `#ifdef` ...)
- Замена комментариев пустыми строками

На выходе получается огромный файл, состоящий исключительно из выражений языка C++. Расширение такого файла обычно `.i` или `.ii`



# Пример

Рассмотрим простой пример.

```
#include <iostream>

int main() {

    std::cout << "hello world" << std::endl;
    return 0;
}
```

Во что превратится программа с таким простым исходным кодом?

# Препроцессинг

Содержимое файла Source.i будет выглядеть таким образом

```
# 1 "Source.cpp"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 366 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "Source.cpp" 2
# 1 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\iostream" 1 3

# 1 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\yvals_core.h" 1 3
# 370 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\yvals_core.h" 3
# 1 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\vcruntime.h" 1 3
# 57 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\vcruntime.h" 3
# 1 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\sal.h" 1 3
# 2361 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\sal.h" 3
extern "C" {
# 2971 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\sal.h" 3
}

# 1 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\concurrency\\sal.h" 1 3
# 22 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\concurrency\\sal.h" 3
extern "C" {
# 391 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\concurrency\\sal.h" 3
}
# 2975 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\sal.h" 2 3
# 58 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\MSVC\\14.30.30705\\include\\vcruntime.h" 2 3
# 1 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\LLVM\\lib\\clang\\12.0.0\\include\\vdefs.h" 1 3
# 18 "C:\\Program Files\\Microsoft Visual Studio\\2022\\Community\\VC\\Tools\\LLVM\\lib\\clang\\12.0.0\\include\\vdefs.h" 3
```

# Компиляция



2



**Компиляция - преобразование полученного кода после препроцессинга в код на машинном языке (бинарный) или на языке, близком к машинному (доступному для понимания человеком, чаще всего это ассемблерный код)**

# Компиляция

Это промежуточный шаг между между высокоуровневым языком (кодом на C++) и машинным кодом

## Этапы компиляции:

- 1 Лексический анализ
- 2 Синтаксический анализ
- 3 Оптимизация
- 4 Генерация кода

# Лексический анализ

Последовательность символов исходного кода преобразуется в последовательность лексем

Например, есть следующий код:

```
int val = (value1 - value2);
```

Он преобразуется примерно в следующий поток лексем:

```
ИМЯ "val"  
ПРИСВАИВАНИЕ  
ОТКРЫВАЮЩАЯ_СКОБКА  
ИМЯ "value1"  
МИНУС  
ИМЯ "value2"  
ЗАКРЫВАЮЩАЯ_СКОБКА  
ТОЧКА_С_ЗАПЯТОЙ
```

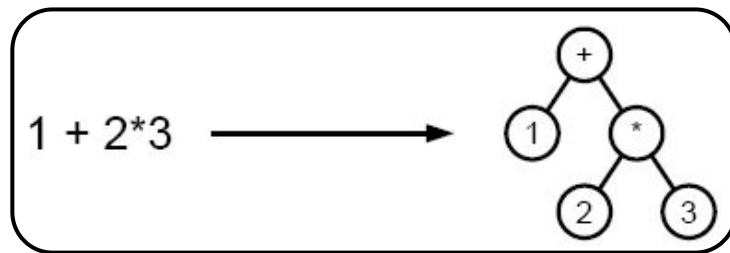
# Синтаксический анализ

Последовательность лексем преобразуется в дерево разбора

Например, есть следующее выражение:

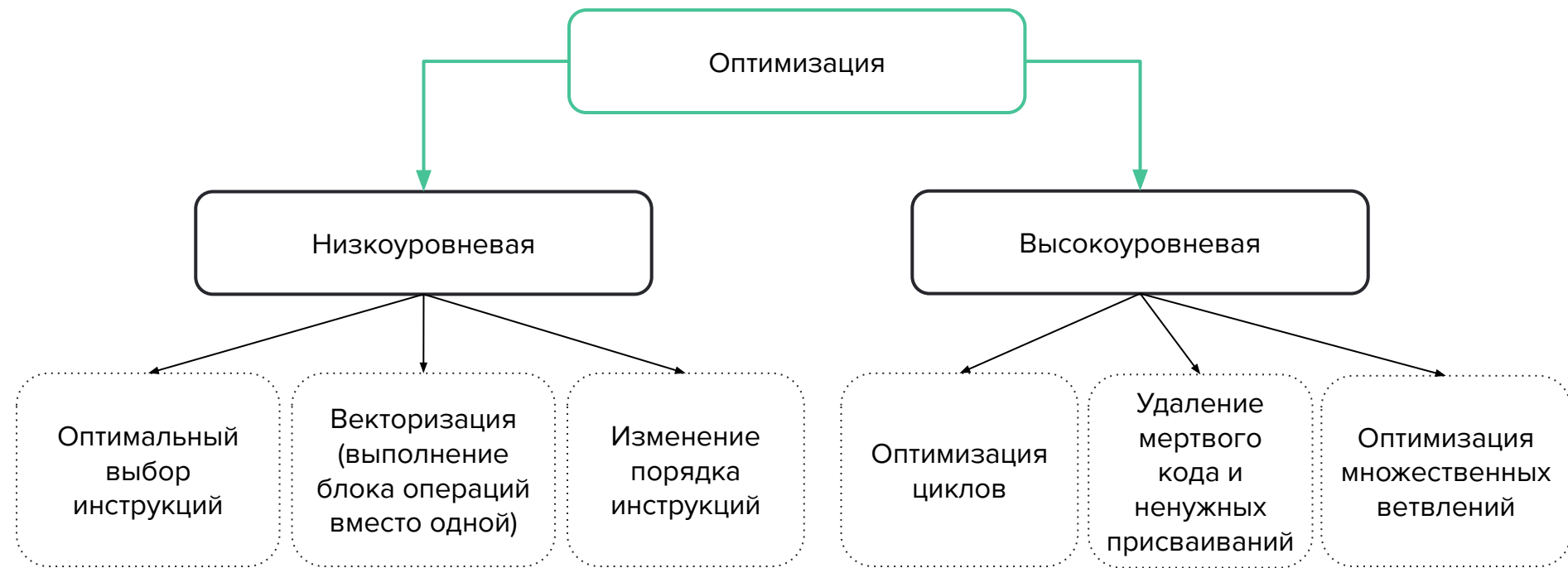
1 + 2 \* 3

Оно преобразуется примерно в следующий поток лексем:



# Оптимизация

Выполняется удаление излишних конструкций и упрощение кода





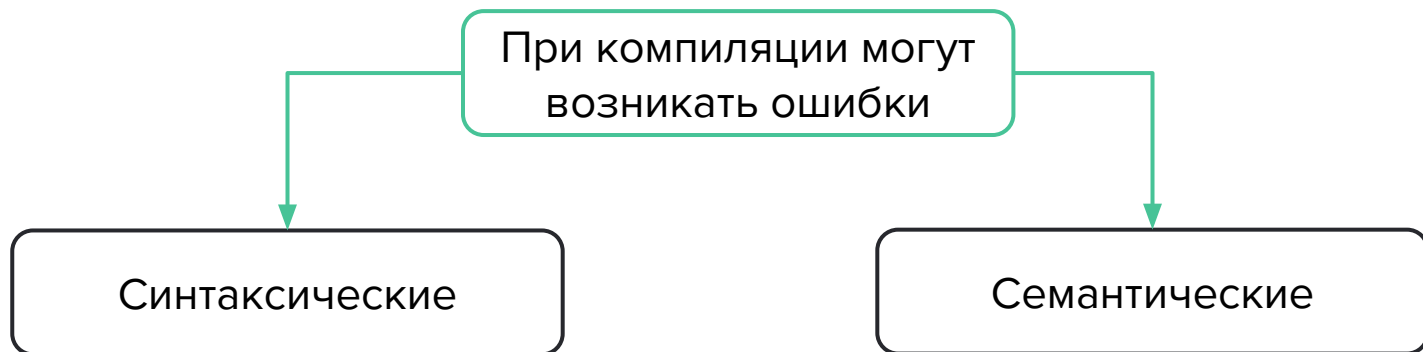
# Генерация кода

Из промежуточного представления создается код на языке, близком к машинному

В различных компиляторах эти этапы могут выполняться по-разному: отдельно или быть совмещены



# Ошибки при компиляции





## **Синтаксические ошибки - это ошибки, возникающие при нарушении базового синтаксиса языка C++**

Самые частые из них:

- пропущена нужная скобка
- пропущена точка с запятой
- обращение к переменной, хотя ее не объявили

Такие ошибки легко исправлять, достаточно просто прочесть сообщения компилятора.

# Синтаксические ошибки

Какая здесь допущена ошибка?

```
#include <iostream>
using namespace std;

int main()
{
    int x = 10;
    std::cout << " " << x
    return 0;
}
```



# Синтаксические ошибки

Какая здесь допущена ошибка?

```
#include <iostream>
using namespace std;

int main()
{
    int x = 10;
    std::cout << " " << x
    return 0;
}
```

Пропущена точка с запятой!



# Синтаксические ошибки

Какая здесь допущена ошибка?

```
#include <iostream>
using namespace std;

int main()
{
    int x = 10;
    while(!){
        x++;
    }
    std::cout << " " << x;
    return 0;
}
```



# Синтаксические ошибки

Какая здесь допущена ошибка?

```
#include <iostream>
using namespace std;

int main()
{
    int x = 10;
    while(!){
        x++;
    }
    std::cout << " " << x;
    return 0;
}
```



Некорректное использование оператора while



**Семантические ошибки возникают, когда код может быть синтаксически верный, но выполняет не то, что нужно.**

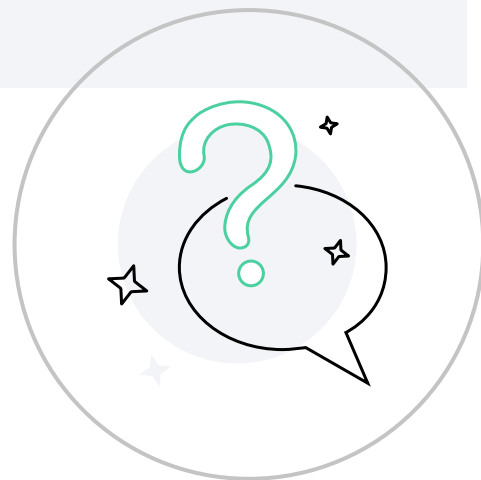
Такие ошибки сложнее отловить, так как нарушена именно логика программы.



# Семантические ошибки

Какая здесь допущена ошибка?

```
int main()
{
    int a, b, c;
    a = 4; b = 1; c = 4;
    a + b = c;
}
```

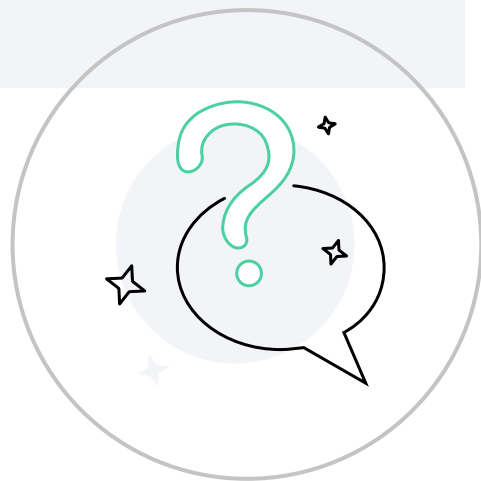


# Семантические ошибки

Какая здесь допущена ошибка?

```
int main()
{
    int a, b, c;
    a = 4; b = 1; c = 4;
    a + b = c;
}
```

Некорректное присваивание в последней строке.



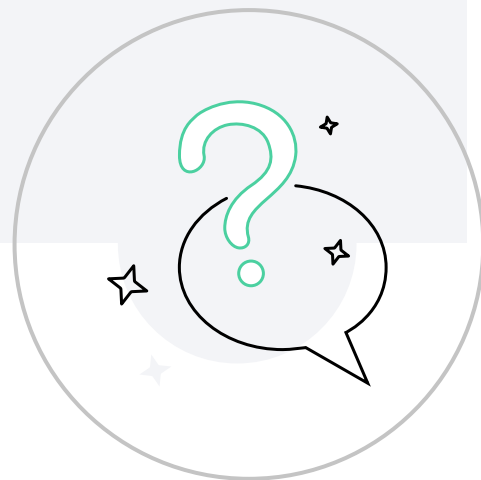
# Семантические ошибки

Какая здесь допущена ошибка?

```
#include <iostream>

int sum(int x, int y) {
    return x - y;
}

int main()
{
    std::cout << sum(5, 3);
    return 0;
}
```



# Семантические ошибки

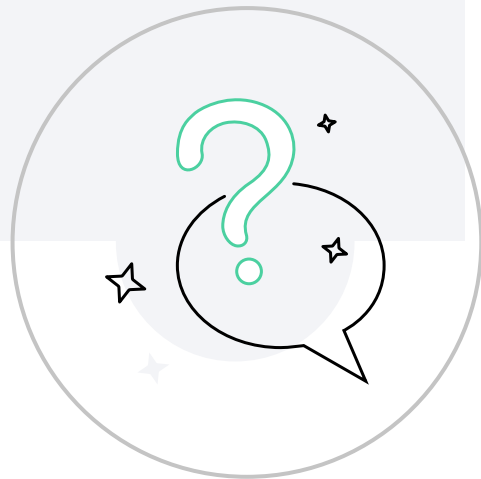
Какая здесь допущена ошибка?

```
#include <iostream>

int sum(int x, int y) {
    return x - y;
}

int main()
{
    std::cout << sum(5, 3);
    return 0;
}
```

Вместо суммирования мы вычисляем сумму двух чисел



# Семантические ошибки

Какая здесь допущена ошибка?

```
#include <iostream>

int sum(int x, int y) {
    return x - y;
}

int main()
{
    return 0;
    std::cout << sum(5, 3);
}
```



# Семантические ошибки

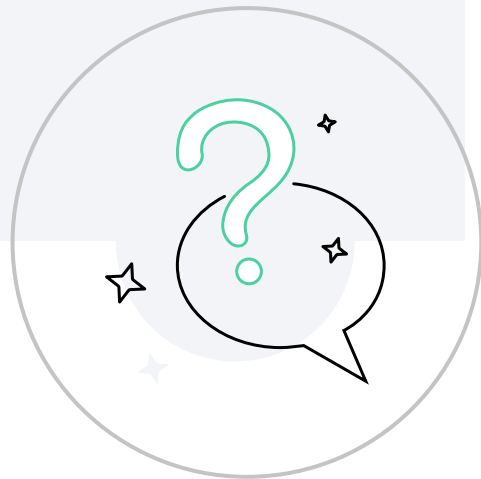
Какая здесь допущена ошибка?

```
#include <iostream>

int sum(int x, int y) {
    return x - y;
}

int main()
{
    return 0;
    std::cout << sum(5, 3);
}
```

Ничего не будет выведено на консоль, так выход из функции почти сразу же.



# Ассемблирование



3

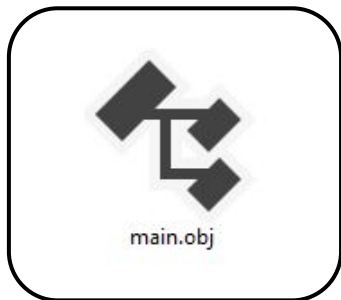


**Так как процессоры исполняют код на машинном коде, нужно перевести код с ассемблера в машинный. Этот процесс и называется ассемблированием.**



# Ассемблирование

Ассемблер преобразовывает ассемблерный код в машинный, сохраняя его в объектном файле (расширение .o, .obj)



Объектный файл - бинарный файл, состоящий из набора функций

**Но это еще не завершающий этап:** объектных файлов может быть много и нужно скомпоновать их в единый исполняемый файл

# Линковка



4



**Линковка - объединение всех объектных файлов и статических библиотек в единый исполняемый файл**

# Зачем нужна линковка

Основная задача линковки - задача по подстановке адресов вызова внешних объектов, которые были образованы в объектных файлах проекта

Это осуществляется с помощью **таблицы символов**



## **Таблица символов - структура, созданная компилятором**

Хранит в себе имена переменных, функций, классов, где каждому идентификатору (символу) соотносится его тип, область видимости, а также адреса ссылок на данные и функции в других объектных файлах.

# Ошибки линковки

Эти ошибки обнаруживаются уже после компиляции на этапе линковки.

Например, компилятор не может найти функцию `main` в вашей программе, функция или объект определены дважды, либо вообще не определены.

# Ошибки линковки

Где здесь ошибка?

```
#include <iostream>
using namespace std;

void func(std::string str);

int main()
{
    func("tmp");
    int a = 5;
    return 0;
}
```

# Ошибки линковки

Где здесь ошибка?

```
#include <iostream>
using namespace std;

void func(std::string str);

int main()
{
    func("tmp");
    int a = 5;
    return 0;
}
```

Нет реализации func. Есть только определение.



# Ошибки линковки

Где здесь ошибка?

```
#include <iostream>
using namespace std;

void func(std::string str){};

int ain()
{
    func("tmp");
    int a = 5;
    return 0;
}
```

# Ошибки линковки

Где здесь ошибка?

```
#include <iostream>
using namespace std;

void func(std::string str){};

int ain()
{
    func("tmp");
    int a = 5;
    return 0;
}
```

Опечатка в функции main. Компилятор не сможет найти функцию main

# Итоги



# Итоги занятия

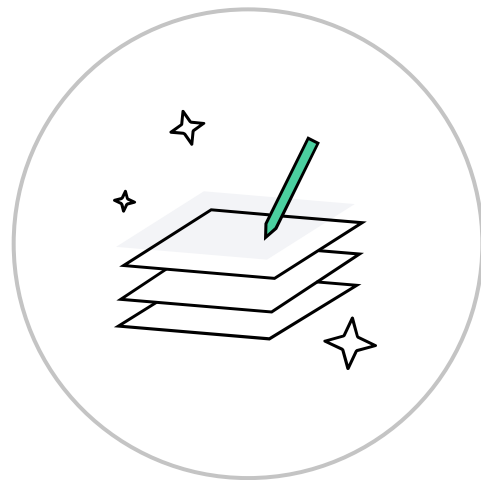
- 1 Узнали, зачем нужно компилировать программы на C++
- 2 Познакомились с основными этапами компиляции
- 3 Узнали какие бывают ошибки компиляции и линковки



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 *\*Обратите внимание, что это самостоятельное задание, без проверки экспертом. После его сдачи в личном кабинете, для самопроверки вы получите авто ответ с правильным решением.\**



# Дополнительные материалы

- [Статья про компиляцию программ на языке C++](#)



**Задавайте вопросы  
и пишите отзыв о лекции**

