

# Программирование на современном C++

Вводная лекция. Этапы сборки программ на языке C++.



образование

# О преподавателях



## Александр Бадьин

Руководитель команды предобработки запросов  
департамента технологий ИИ и прикладных  
исследований

- Занимаюсь обработкой запросов в поисковых системах VK
- Люблю C++

Telegram: [@a\\_badin](https://www.telegram.me/a_badin)

# О преподавателях



## Иван Возвахов

Руководитель команды разработки в VK Devices

- Занимается разработкой колонки с Марусей: как частью клиента, так и драйвера в u-boot, ядро линукса.
- Приходилось работать со звуком и голосом
- Взломал станцию мини за два часа
- Видел много китайского кода, поэтому не придирается к мелочам

Telegram: [@vozvivan](https://t.me/vozvivan)

# О преподавателях



## Илья Крамичев

Старший программист команды предобработки запросов департамента технологий ИИ и прикладных исследований

- Занимаюсь графом знаний и семантическим вебом в поисковых системах VK
- Имею хороший опыт в области IP-телефонии

Telegram: [@IKramichev](https://www.instagram.com/ikramichev)

# Менторская программа

## Кто такие менторы?

- Наставник
- Менеджер
- Приятель
- Психолог
- Справочное бюро
- Проверятель первой итерации домашних
- С ними можно на “ты”

# Менторская программа

## Менторы курса:

- Арсений Евдокимов (Telegram: @Arugraf)
- Георгий Седойкин (Telegram: @Ge0rgiyX)
- Виктор Кочур (Telegram: @PivasBulba)
- Павел Чеклин (Telegram: @paulnopaul)
- Артем Ветошкин (Telegram: @TheCompilerA)
- Андрей Коленков (Telegram: @Adefe)
- Сергей Ванданов (Telegram: @rapid76)
- Евгений Рудых (Telegram: @RudykhEugeniY)

## Старший ментор:

Пешков Дмитрий (Telegram: @DmPesh) – по любым вопросам взаимодействия с менторами

# Отметьтесь на портале!

- Посещение необязательное, но тем, кто пришёл, следует отмечаться на портале в начале каждого занятия
- Это позволяет нам анализировать, какие занятия были более или менее интересны студентам, и менять курс в лучшую сторону
- Также это даст возможность вам оставить обратную связь по занятию после его завершения


пн, 1 ноября	вт, 2 ноября	ср, 3 ноября	чт, 4 ноября	пт, 5 ноября	сб, 6 ноября
Нет занятий	<div>18:00 Углубленный C/C++ (w... <span>п</span></div> <div>Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование</div> <div>А. Халайджи</div> <div>18:00 Углубленный C/C++ (ML) <span>п</span></div> <div>Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование</div> <div>А. Халайджи</div>	Нет занятий	Нет занятий	Нет занятий	Нет занятий

Ссылка на Zoom РК сегодня в 18:00

Безопасность интернет-приложений (третий семестр)

Подключиться к конференции Zoom  
<https://mailru.zoom.us/j/98586081818?pwd=eWxwSDdCbHhQNnNwQU5iblFvU2dTZz09>

Идентификатор конференции: 985 8608 1818  
Код доступа: 785885

 Алексей Набережный 55 минут назад

★ 0 💬 0 ↓ 0 ↑

Запись прошлой лекции (+ что почитать перед РК)

Безопасность интернет-приложений (третий семестр)

### Углублённый C/C++

Лекция 5

📍 Онлайн - ML

Отметьтесь, что вы пришли на занятие. Так вы улучшите свою посещаемость и вас увидит преподаватель в своём "Журнале посещений".

Отметиться

Оставьте отзыв о занятии и мы сможем улучшить учебный процесс.

Оставить отзыв

# Цель и структура курса

**Цель курса** – сформировать практические навыки и умения, необходимые специалистам по разработке программного обеспечения (ПО) UNIX-подобных операционных систем для участия в проектах промышленной разработки среднего уровня сложности на C++, в том числе для замещения стажерских должностей разработчиков высоконагруженных приложений.

## **Структура курса:**

- 13 лекций с практическими семинарами
- 3 рубежных контроля

В конце курса – публичная защита командных проектов.



# Чему научимся?

- Разрабатывать код на языке C++ с использованием возможностей современных стандартов;
- Создавать качественный код промышленного уровня;
- Проектировать программные системы, в том числе при помощи UML-диаграмм;
- Применять популярные C++ библиотеки;
- Обрабатывать ошибки и диагностировать программы;
- Разрабатывать базовые сетевые и многопоточные приложения;
- Тестировать код;
- Проходить ревью и выполнять рефакторинг программного кода;
- Работать в команде;
- Презентовать и защищать свои разработки перед аудиторией.

# Модульно-рейтинговая система

- Модуль № 1. Введение в современный C++. ООП.

Выдаётся 2 практических задания:

- ДЗ № 1 – Работа с файлами и строками (индивидуально) – до 10 баллов;
- ДЗ № 2 – ООП и умные указатели (индивидуально) – до 15 баллов;

Завершается РК № 1 – Дедлайн по ДЗ № 1-2, обсуждение проектов с командами

# Модульно-рейтинговая система

- Модуль № 2. Прикладной C++. Тестирование, проектирование программных систем

Выдаётся 2 практических задания:

- ДЗ № 3 – Разработка через тестирование (индивидуально) – до 10 баллов;
- ДЗ № 4 – Архитектура программной системы, UML (в командах) – до 10 баллов;

Завершается РК № 2 – Дедлайн по ДЗ № 3-4, обсуждение промежуточных результатов работы над проектами.

# Модульно-рейтинговая система

- Модуль № 3. Многопоточное и системное программирование. Многопоточное программирование, POSIX.

Модуль нацелен на реализацию MVP (Minimum Viable Product) проектов.

Выдаётся 2 практических задания:

- ДЗ № 5 – Разработка прикладных библиотек (индивидуально) – до 20 баллов;
- ДЗ № 6 – MVP (в командах) – до 10 баллов;

Завершается РК № 3 – Дедлайн по ДЗ № 5-6, контроль готовности основных функций MVP командных проектов.

# Модульно-рейтинговая система

- Модуль № 4. Подготовка проекта к защите.

Модуль нацелен на доведение проектов до ума, наведение «красоты», добавления новых возможностей и подготовки выступления.

Завершается РК № 4 – предзащита, на которой выдаётся доступ к защите.

- Открытая защита командных проектов. На ней можно получить до 25 баллов за готовность продукта, технологичность проекта и качество выступления
- Пересдача (для желающих повысить свой балл) Имеет формат экзамена с пристрастием, можно получить до 20 баллов

# Проверка домашних заданий

Все ДЗ отправляются через портал!

При сдаче необходимо добавить на проверку ментора, продублировать описание задачи и ссылку на выполненное ДЗ в чат, чтобы у ментора появился к нему доступ на портале. Не забудьте выдать доступы к репозиторию!

Ментор выбирается командами при разбиении на команды. Не откладывайте это!!!

Действует система **жестких дедлайнов**!

Проверяются только те задания, которые отправлены на проверку **СТРОГО ДО ДЕДЛАЙНА**.

Если задание не отправлено до дедлайна – ставится **0 баллов**.

Индивидуальные задания проверяются ментором по готовности до дедлайна, после дедлайна их проверяет в порядке общей очереди преподаватель. Баллы в ДЗ отражают ситуацию на момент дедлайна.

До дедлайна стоит **активно консультироваться** с ментором (возможны доп. итерации ревью) и преподавателем.

# Требования к домашним заданиям

- В репозитории находятся все файлы необходимые для сборки программы.
- В репозитории отсутствуют файлы, которые не нужны для сборки программы (настройки IDE, сгенерированные файлы сборки).
- Программа собирается стандартным окружением с включенным стандартом C++17.
- В репозитории есть пример данных, на которых можно запустить программу и убедиться в ее работоспособности.
- В программе используются понятные, развернутые имена (переменные, функции, классы итд.)  
Либо развернутые комментарии с описанием, что делают методы, параметры или класс.

# Требования к домашним заданиям

- Длина методов ограничена 50 строками.
- Длина выражения ограничена 80 символами.
- Максимальная вложенность блоков без учета пространств имен 3.
- Код оформлен в любом стиле какой вам нравится, но он должен быть един на весь компонент.
- В коде **отсутствуют методы и типы и конструкции из языка C** (заголовки вида <c\*\*\*>) при наличии эквивалентных методов, типов и конструкции из языка C++.
- В программе нет утечек ресурсов.



# Система оценок

Итоговая оценка выставляется в соответствии со следующей шкалой перевода:

- от 0 до 49 – «неудовлетворительно»
- от 50 до 69 – «удовлетворительно»
- от 70 до 89 – «хорошо»
- от 90 и более – «отлично»

И ещё:

За списывание чего-либо в ДЗ **обнуляются** баллы за полное ДЗ у **всех уличенных**.

Пересдача позволяет получить до 20 баллов => повысить оценку на 1 балл.

Но пересдача – это **«сложно»**, поэтому рекомендуется работать в течение семестра!

Студенты, не набравшие 30 баллов в течение семестра, **не допускаются к пересдаче!**

# Организационные моменты

Блог дисциплины размещён по адресу:

<https://park.mail.ru/blog/view/18/>

Крайне рекомендуется:

- подписаться на него;
- ознакомиться с более ранними записями;
- задавать вопросы;
- участвовать в опросах и обсуждениях.

Все материалы, касающиеся курса, будут своевременно публиковаться в нём.

# Организационные моменты

- длительность занятия – 3 часа с 1 перерывом продолжительностью до 10 минут;
- занятия состоят из лекции 1.5 – 2 часа и семинара;
- место проведения занятия следует проверять заранее на портале;
- в начале каждого занятия необходимо отмечаться на портале в электронном журнале;
- вопросы во время лекции стоит задавать по мере их поступления или индивидуально на перерыве или после занятия;
- вопросы вне аудиторных занятий лучше задавать в блоге, чате или через личные сообщения на портале;
- после каждого занятия следует уделять пару минут для того, чтобы оставить отзыв по занятию на портале

# Организационные моменты

Про взаимодействие в чате настоятельно рекомендуется:

- относиться нужно ко всем уважительно, не переходя на личности. Стараться помогать друг другу;
- общаться стоит конструктивно, для флуда и холиваров на отвлеченные темы есть другие чаты;
- в чате не стоит давать готовые решения. Лучше – объяснить суть проблемы, привести ссылку, где проблема описана более подробно, обсудить возможные подходы к решению;
- перед тем, как задать вопрос, проверьте, что этот же вопрос не задавался ранее;
- перед обращением за помощью постарайтесь сначала самостоятельно разобраться в проблеме, чтобы диалог начинался не с «как», а с «есть проблема, я вижу такие-то решения, но...».
- не стоит рассчитывать, что преподаватели/менторы будут охотно отвечать по ночам или в последние часы перед дедлайном/сразу в течение нескольких часов.

# Организационные моменты

Чаще коммуницируйте с вашим ментором и преподавателем, задавайте больше вопросов.

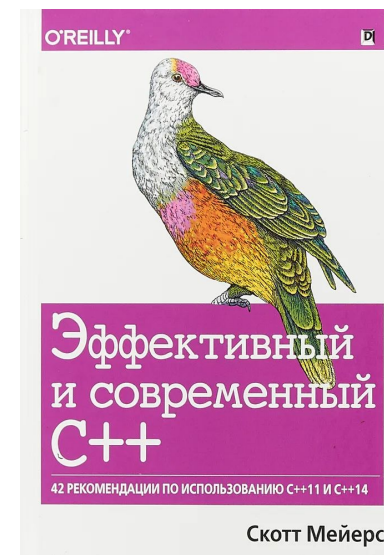
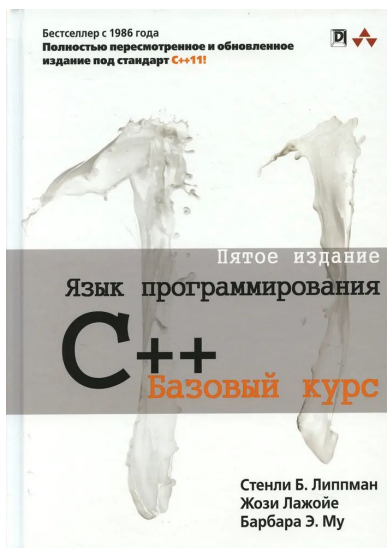
От этого **напрямую** зависит результат прохождения курса!

# Web-ресурсы

- **Справка по C++** — <http://en.cppreference.com/w/>
- **Standard C++** — <http://isocpp.org/>
- **Google C++ Code Style** — <https://google.github.io/styleguide/cppguide.html>
- **Code Style Guidelines | WebKit** — <https://webkit.org/code-style-guidelines/>
- **LLVM Coding Standards** — LLVM5 documentation: <http://llvm.org/docs/CodingStandards.html>
- **C++ Core Guidelines (CG)** – <https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md>

# Книги

- Эффективный и современный C++:  
42 рекомендации по использованию C++11 и C++14 | Мейерс Скотт
- Язык программирования C++. 4-е изд | Страуструп Бьярне
- Язык программирования C++. Базовый курс. 5-е изд | Му Барбара Э., Лажойе Жози



# Этапы сборки программ на языке C++



# План лекции

- Язык C++ сегодня
- Препроцессор
- Компилятор
- Компоновщик
- Перерыв
- Make
- CMake

# Язык C++ сегодня

2022 — активное применение языка в практике программирования задач общего и специального назначения: ядра ОС;

- игровые/научные движки, высокопроизводительные распределенные вычисления;
- системы управления БД и Web-серверы и пр.;
- проекты VK/Яндекс/Google/...: Почта, Поиск, YouTube и пр.

Язык активно развивается: C++98, C++03, C++11, C++14, C++17, C++20, C++23 (в разработке)

Язык поддерживает большое количество разных парадигм — после него проще переходить на другие языки программирования.

Чаще всего используется тогда, когда нужно писать высокопроизводительный код, имея возможность использовать высокоуровневые языковые абстракции и парадигмы.

# Язык C++ сегодня

TIOBE Programming Community Index - индекс популярности языков программирования.

Базируется на оценке числа документов найденных по запросу, содержащему название языка программирования, в популярных поисковых системах.

В феврале 2023 г. TIOBE Programming Community Index языка C++ составляет 3-е место, конкурируя, главным образом, с другими Си-подобными языками (Си, Java, C#) и Python.

Получил награду Programming Language of the Year 2022, как язык с самым высоким ростом популярности за год.

# Трансляторы

Трансляция программы — преобразование программы, представленной на одном из языков программирования, в программу, написанную на другом языке

1. Компиляторы (динамические и статические)
2. Интерпретаторы

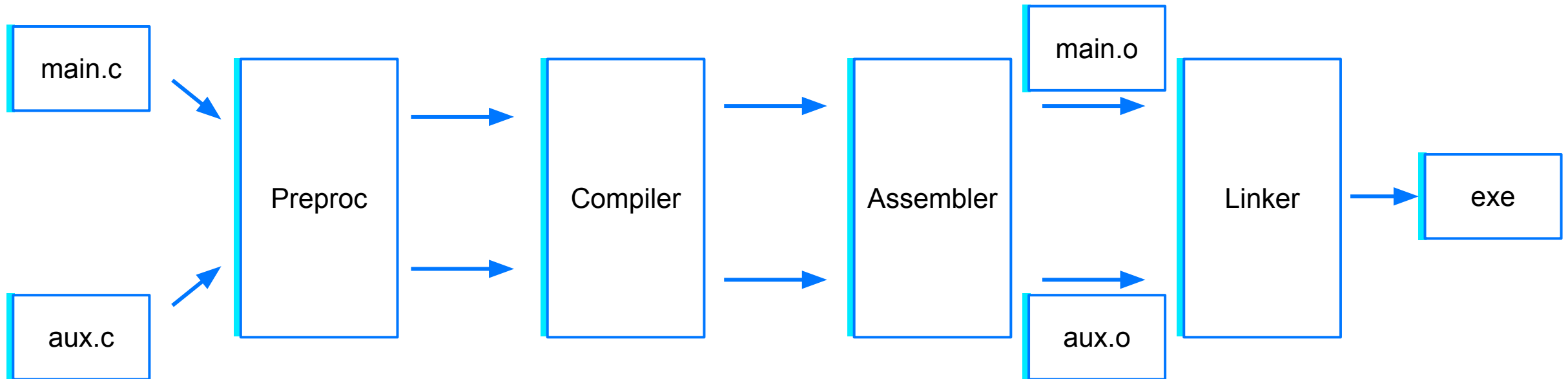
# Препроцессор, компилятор, компоновщик

Как из исходного кода получается программа на C++?

# Этапы компиляции

1. **Препроцессор.** Обработка исходного кода (preprocessing);
2. **Компиляция.** Перевод подготовленного исходного кода в ассемблерный код (compiling);
3. **Ассемблирование.** Перевод ассемблерного кода в объектных файл (assembly);
4. **Компоновка.** Сборка одного или нескольких объектных файлов в один исполняемый файл (linking).

# Этапы компиляции



# Препроцессор

Делаются макроподстановки:

- определения (`#define`, `#undef`);
- условные включения (`#ifdef`, `#ifndef`, `#if`, `#endif`, `#else` и `#elif`);
- директива `#line`;
- директива `#error`;
- директива `#pragma`;

Результат обработки препроцессором исходного файла называется единицей трансляции.



# Препроцессор

Выполняются директивы `#include`

- `#include <name>` — целиком вставляет файл с именем “name”, вставляемый файл также обрабатывается препроцессором. Поиск файла происходит сначала в директориях перечисленных через ключ `-I`, затем в системных директориях. Для заголовков системных программ
- `#include “name”` — аналогично предыдущей директиве, но поиск производится сначала в директориях с файлом, из которого происходит включение, затем как для `#include <name>`. Для заголовков своих программ

# Препроцессор. Пример

```
$ mkdir mylib; echo "#include <mylib.h>" > example.c; gcc -v example.c -Imylib; rm example.c mylib
```

...

#include "... " search starts here:

#include <...> search starts here:

mylib

/usr/lib/gcc/aarch64-linux-gnu/11/include

/usr/local/include

/usr/include/aarch64-linux-gnu

/usr/include

End of search list.

...

-v - Print (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper.

# Препроцессор. Двойное включение

- Чтобы защититься от двойного включения одного и того же заголовочного файла, и не словить ошибку компиляции, используется страж включения (include guard, или предохранитель включения).

```
#ifndef HEADER_NAME_HPP
```

```
#define HEADER_NAME_HPP
```

```
// Содержимое заголовочного файла
```

```
#endif
```

- Большинство компиляторов поддерживают отдельную директиву

```
#pragma once
```

```
// Содержимое заголовочного файла
```

# Препроцессор. Пример

```
$ g++ -E example.cpp -o example.ii
```

```
1. #include <iostream>
```

```
2. #define NAME(world) #world
```

задаем строковую константу

```
3. int main(int argc, char **argv)
```

```
4. {
```

```
5. #line 100
```

```
6. std::cout << "Hello, " << NAME(world) << " from " << __FILE__ << " and line #" << __LINE__ << std::endl;
```

```
7. }
```

The expansions of both `__FILE__` and `__LINE__` are altered if a `#line` directive is used. It causes the compiler to view the line number of the next source line as the specified number.

// Опция -E запускает только препроцессор.

# Компиляция/Ассемблирование

Файлы .cpp/.c — один файл с исходным кодом — один объектный файл.  
Это называется единица трансляции.

# Компиляция и ассемблирование: создать объектный файл

# example.o

\$ g++ -c example.cpp

# или

# Только компиляция: создать ассемблерный код example.s, ...

\$ g++ -S example.cpp

# ... а затем ассемблирование: создание объектного файла.

\$ as example.s -o example.o

ИЛИ: СРАЗУ ИЗ C/CPP В ИСПОЛНЯЕМЫЙ ФАЙЛ БЕЗ ФЛАГОВ -C  
-O

ОБЪЕКТНЫЙ ФАЙЛ - ФАЙЛ С БИНАРНЫМ КОДОМ, МОЖНО  
ПОСМОТРЕТЬ ДАМПЫ С ПОМОЩЬЮ LINUX УТИЛИТЫ XXD

ЛИБО ИСПОЛЬЗОВАТЬ OBJDUMP/NM/READELF - ДЛЯ  
ДИЗАССЕМБЛИРОВАНИЯ, ПРОСМОТРА СЕКЦИЙ  
ОБЪЕКТНЫХ ФАЙЛОВ И ТД

МОЖНО ДАМПИТЬ(СМОТРЕТЬ БИНАРНЫЙ КОД) ОТДЕЛЬНЫХ  
СЕКЦИЙ(НАПРИМЕР, С ПОМОЩЬЮ ФЛАГА -X УТИЛИТЫ  
READELF)

command1 && command2 etc - для выполнения нескольких  
команд

# Компиляция/Ассемблирование

- `objdump/nm/readelf`    ЭТО ЧАСТИ BINUTILS
- Сравните C и C++

# Компоновка

Компоновщик собирает из одного и более объектных файлов исполняемый файл.

```
$ g++ -o my_prog main.o square.o
```

```
$ ./my_prog
```

# Компоновка

- Организация программы как набор файлов с исходным кодом, а не один монолитный файл.
- Организовывать библиотеки функций, являющихся общими для разных программ;
- Раздельная компиляция:
  - Меняем код в одном файле, компилируем только его, повторяем компоновку;
  - Нет необходимости повторять компиляцию остальных файлов с исходным кодом.
- Исполняемые файлы и образ программы в памяти содержит только те функции, которые действительно используются.



# GNU Compiler Collection

- C (gcc)
- C++ (g++)
- Objective C
- Fortran
- Ada (GNAT)
- Go
- D



# Компиляция executable с GCC

- Компиляция в объектный файл

```
gcc -c -I lib/foo/include main.cpp -o main.o
```

-I - для поиска всех хедеров внутри директории

В языке программирования C код библиотек представляет собой функции, размещенные в файлах, которые скомпилированы в объектные файлы, а те, в свою очередь, объединены в библиотеки. В одной библиотеке объединяются функции, решающие определенный тип задач. Например, существует библиотека математических функций.

У каждой библиотеки должен быть свой заголовочный файл, в котором должны быть описаны прототипы (объявления) всех функций, содержащихся в этой библиотеке. С помощью заголовочных файлов вы "сообщаете" вашему программному коду, какие библиотечные функции есть и как их использовать.

При компиляции программы библиотеки подключаются линковщиком, который вызывается gcc. Если программе требуются только стандартные библиотеки, то дополнительных параметров линковщику передавать не надо (есть исключения). Он "знает", где стандартные библиотеки находятся, и подключит их автоматически. Во всех остальных случаях при компиляции программы требуется указать имя библиотеки и ее местоположение

# Компиляция executable с GCC

- Компиляция в объектный файл

```
gcc -c -I lib/foo/include main.cpp -o main.cpp.o
```

УКАЗЫВАЕМ ДЛЯ -I ГДЕ ИСКАТЬ НУЖНЫЕ  
ХЕДЕРЫ

- Компоновка с shared библиотеками

```
gcc main.cpp.o -L/opt/gtest/lib -lgtest -o example
```

ДОБАВЛЕНИЕ ССЫЛКИ НА SO В  
EXECUTABLE

команда для поиска файлов: `find dir -name "name"`

# Компиляция executable с GCC

- Компиляция в объектный файл

```
gcc -c -I lib/foo/include main.cpp -o main.o
```

- Компоновка с shared библиотеками

```
gcc main.cpp.o -L/opt/gtest/lib -lgtest -o example
```

ДОБАВЛЕНИЕ ССЫЛКИ НА SO В EXECUTABLE

- Компоновка со static библиотеками

```
gcc main.cpp.o /opt/gtest/lib/liblog.a -o example
```

ДОБАВЛЕНИЕ АРХИВА STATIC ЛИБЫ В EXECUTABLE

# Компиляция библиотек с GCC

- Компиляция shared библиотеки

```
gcc -I include -fPIC -c log.cpp -o log.cpp.o
```

-fPIC - ПОЗИЦИОННО-НЕЗАВИСИМЫЙ КОД

# Компиляция библиотек с GCC

- Компиляция shared библиотеки (ПОЛУЧЕНИЕ ОБЪЕКТНИКОВ )

```
gcc -I include -fPIC -c log.cpp -o log.cpp.o
```

- Компоновка shared библиотеки ПОЛУЧЕНИЕ ФАЙЛА ЛИБЫ

```
gcc -shared -o liblog.so log.cpp.o
```

# Компиляция библиотек с GCC

- Компиляция shared библиотеки

```
gcc -I include -fPIC -c log.cpp -o log.cpp.o
```

- Компоновка shared библиотеки

```
gcc -shared -o liblog.so log.cpp.o
```

- Компиляция static библиотеки

```
gcc -I include -c log.cpp -o log.cpp.o
```

# Компиляция библиотек с GCC

- Компиляция shared библиотеки

```
gcc -I include -fPIC -c log.cpp -o log.cpp.o
```

- Компоновка shared библиотеки

```
gcc -shared -o liblog.so log.cpp.o
```

- Компиляция static библиотеки ОБЪЕКТНЫЕ ФАЙЛЫ ДЛЯ STATIC  
ЛИБЫ

```
gcc -I include -c log.cpp -o log.cpp.o
```

- Архивация static библиотеки ПОЛУЧЕНИЕ STATIC  
ЛИБЫ

```
ar qc liblog.a log.cpp.o
```



# Полезные флаги

## Оптимизации:

- O0 — отключение оптимизации (по умолчанию).
- O1 — пытается уменьшить размер кода и ускорить работу программы за счет увеличения времени компиляции.
- O2 — все поддерживаемые оптимизации, которые не включают уменьшение времени исполнения за счет увеличения длины кода.
- O3 — оптимизирует ещё немного за счет увеличения длины кода.

# Полезные флаги

## **Оптимизации:**

- O0 — отключение оптимизации (по умолчанию).
- O1 — пытается уменьшить размер кода и ускорить работу программы за счет увеличения времени компиляции.
- O2 — все поддерживаемые оптимизации, которые не включают уменьшение времени исполнения за счет увеличения длины кода.
- O3 — оптимизирует ещё немного за счет увеличения длины кода.

## **Предупреждения:**

- Wall — вывод сообщений о всех предупреждениях или ошибках, возникающих во время компиляции программы.
- Wextra — "агрегатор" дополнительных предупреждений.
- Werror — делает все предупреждения ошибками.

# Полезные флаги

## Оптимизации:

- O0 — отключение оптимизации (по умолчанию).
- O1 — пытается уменьшить размер кода и ускорить работу программы за счет увеличения времени компиляции.
- O2 — все поддерживаемые оптимизации, которые не включают уменьшение времени исполнения за счет увеличения длины кода.
- O3 — оптимизирует ещё немного за счет увеличения длины кода.

## Предупреждения:

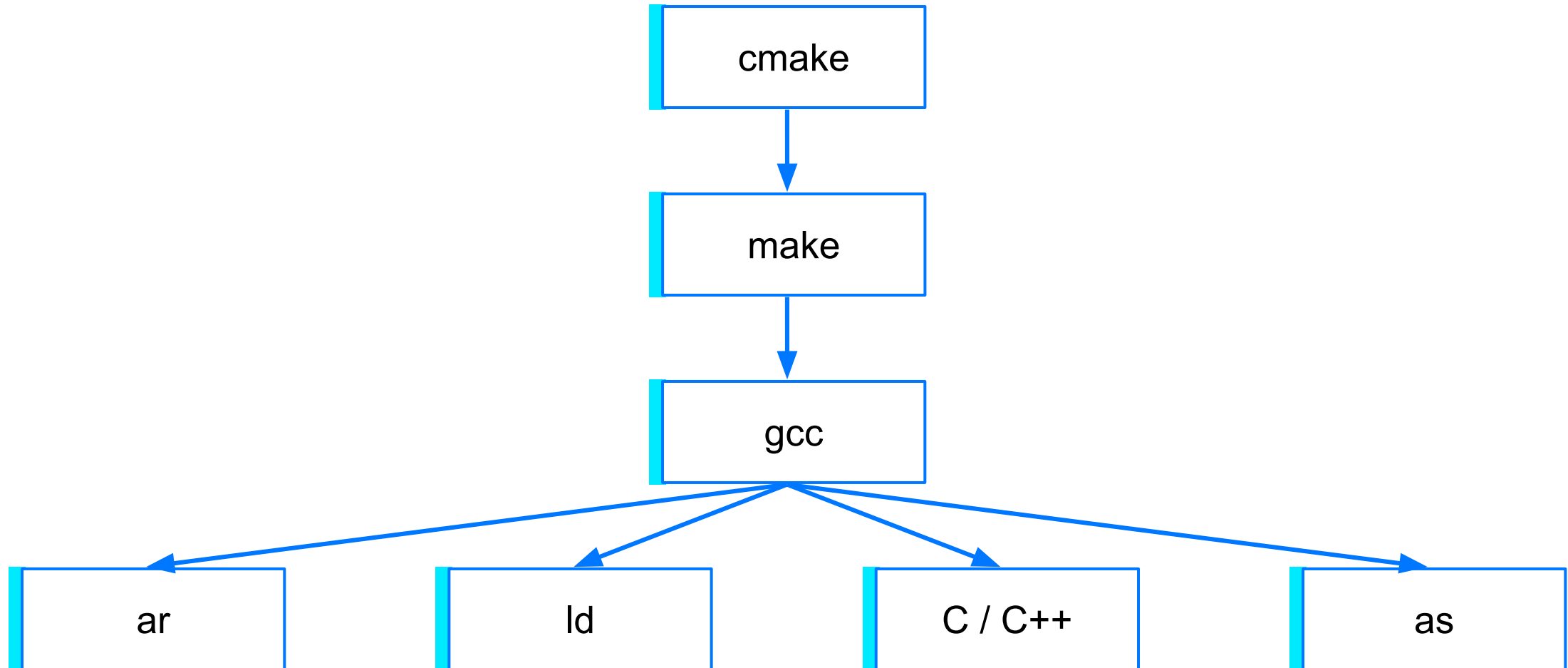
- Wall — вывод сообщений о всех предупреждениях или ошибках, возникающих во время компиляции программы.
- Wextra — "агрегатор" дополнительных предупреждений.
- Werror — делает все предупреждения ошибками.
- Wpedantic - соответствие стандарту

## Отладка:

- g - генерация и сохранение отладочной информации.
- fsanitize=address — умеет ловить использование освобожденной памяти, переполнения и утечки.

# Перерыв!

# Иерархия процесса сборки



# GNU Make

Программа автоматизации  
преобразования файлов.

Описывается в файлах Makefile в  
формате правил.



# Формат Makefile

```
target: [dependencies]  
    command  
    ...
```

# Формат Makefile

command исполнится, если цели нет, или если зависимости новее цели.  
(mod. time в файловой системе)

```
target: [dependencies]  
        command  
        . . .
```

ИЩЕМ ЦЕЛЬ --> ЕСЛИ ЗАВИСИМОСТЬ ЕСТЬ, ТО КОМАНДА  
ИСПОЛНИТСЯ



# Пример

```
main: main.o  
      gcc main.o -o main
```

```
main.o: main.cpp  
      gcc -c main.cpp -o main.o
```

# Make, пример Makefile

Обычно в начале располагают блок переменных

```
1 TARGET = hello
2 CC = g++
3
4 INCLUDE = \
5     -I./include
6
7 C_DEFS = \
8     -DUSE_ZLIB
9
10 LIBS = -lzlib
11 LDFLAGS = -L./lib $(LIBS)
12
13 CFLAGS = $(C_DEFS) $(INCLUDE)
```

# Make, пример Makefile

Затем блок правил

```
1 all: $(TARGET) # FIRST IS DEFAULT
2
3 .PHONY: all clean install # NOT A FILE/DIR
4
5 main.o: src/main.cpp
6     $(CC) $(CFLAGS) -c src/main.cpp -o main.o
7
8 $(TARGET): main.o
9     $(CC) $^ -o $@
10
11 clean:
12     rm -rf $(TARGET) *.o
```

# Make, аргументы вызова

**make <target>** — собрать <target>

**make VAR1=1 VAR2=2** — установить переменные

**make -j <N>** — собрать используя несколько процессов

**make -C <dir>** — запустить из рабочей директории

**make -n** — dry run, вывод команд без выполнения

**make -B** — принудительно пересобрать цели

**make -d** — отладка

# Make, минусы

- Сложно конфигурировать
- Нельзя переиспользовать код
- Много boilerplate-кода
- Конфигурация является частью исходников

# CMake

Кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода.

- Сборка
- Кросс-компиляция
- Тестирование
- Сборка пакетов



# CMake

Кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода.

- Сборка
- Кросс-компиляция
- Тестирование
- Сборка пакетов

CMake предоставляет **переносимость** процесса сборки программ



# CMake, введение

Правила сборки описываются в файлах CMakeLists.txt.

После вызова cmake генерирует необходимые для сборки файлы в **рабочей** директории.

- `mkdir build`
- `cd build`
- `cmake ../`
- `make`



# CMake, введение

Правила сборки описываются в файлах CMakeLists.txt.

После вызова cmake генерирует необходимые для сборки файлы в рабочей директории.

- `mkdir build`
- `cd build`
- `cmake ../`
- `make`


**Out-of-source build**



# CMake, введение

Правила сборки описываются в файлах CMakeLists.txt.

После вызова cmake генерирует необходимые для сборки файлы в рабочей директории.

- |                            |   |                               |
|----------------------------|---|-------------------------------|
| ● <code>mkdir build</code> |   | ● <code>cmake -B build</code> |
| ● <code>cd build</code>    |   | ● <code>make -C build</code>  |
| ● <code>cmake ../</code>   |  |                               |
| ● <code>make</code>        |   |                               |

# CMake, CMakeCache.txt

В сгенерированных файлах сборки всегда присутствует кэш конфигурации CMakeCache.txt с параметрами.

# CMake, CMakeCache.txt

В сгенерированных файлах сборки всегда присутствует кэш конфигурации CMakeCache.txt с параметрами.

Кэш CMakeCache.txt:

- Ускоряет повторную конфигурацию

# CMake, CMakeCache.txt

В сгенерированных файлах сборки всегда присутствует кэш конфигурации CMakeCache.txt с параметрами.

Кэш CMakeCache.txt:

- Ускоряет повторную конфигурацию
- Позволяет пользователю переопределять параметры сборки.

# CMake, CMakeCache.txt

Примеры кэшируемых параметров:

## Базовые

CMAKE\_BUILD\_TYPE # Тип сборки Debug, Release

CMAKE\_INSTALL\_PREFIX # Префикс пути установки

# CMake, CMakeCache.txt

Примеры кэшируемых параметров:

## Базовые

CMAKE\_BUILD\_TYPE # Тип сборки Debug, Release

CMAKE\_INSTALL\_PREFIX # Префикс пути установки

## Расширенные

CMAKE\_CXX\_FLAGS # Флаги компилятора C++

CMAKE\_CXX\_COMPILER # Компилятор C++

CMAKE\_VERBOSE\_MAKEFILE # Вывод команд make

...

ZLIB\_LIBRARY # Путь до библиотеки

ZLIB\_INCLUDE\_DIR # Путь до заголовочных файлов

# CMake, синтаксис CMake файлов

CMake файлы состоят **только** из команд формата

команда(параметр1 параметр2...)



# CMake, синтаксис CMake файлов

CMake файлы состоят из команд формата

`команда(параметр1 параметр2...)`

```
1 cmake_minimum_required(VERSION 3.20)
2
3 project>Hello)
4 add_executable(hello hello.cpp)
```

# CMake, переменные

```
1  set(VAR Foo)
2  message(${VAR}) # Foo
3
4  set(VAR_SPACED_STR "Hello World!")
5  message(${VAR_SPACED_STR}) # Hello World!
6
7  set(VAR_LIST Hello World!)
8  message(${VAR_LIST}) # HelloWorld!
9  message("${VAR_LIST}") # Hello;World!
10
11 set(VAR_SEMICOLON Hello;World!)
12
13 set(VAR_NEWLINE Hello
14      World!)
```

# CMake, полезные команды

- Добавление под-директории с CMakeLists.txt

```
add_subdirectory(lib/utills)
```

- Стандарт C/C++

```
set(CMAKE_CXX_STANDARD 17)  
set(CMAKE_C_STANDARD 99)
```

# CMake, ветвления

```
1  if(...)
2      message(TRUE)
3  endif()
```

Попадет в условие, если аргумент:

1 или ON или YES или TRUE или Y или число больше 1

# CMake, ветвления

```
1 if(...)
2     message(TRUE)
3 endif()
```

Попадет в условие, если аргумент:

1 или ON или YES или TRUE или Y или число больше 1

Не попадет в условие, если аргумент:

0 или OFF или NO или FALSE или N  
IGNORE или NOTFOUND или пустая строка

# CMake, ветвления

В остальных случаях считает, что строка - переменная и автоматически разыменовывает её.

```
1 set(VAR_1 1)
2 if(VAR_1)
3     message("Shown")
4 endif()
```

# CMake, ветвления

В остальных случаях считает, что строка - переменная и автоматически разыменовывает её.

```
1 set(VAR_1 1)
2 if(VAR_1)
3     message("Shown")
4 endif()
```

```
1 set(VAR_2 False)
2 if(VAR_2)
3     message("Not shown")
4 endif()
```

# CMake, ветвления

В if есть различные операторы, например, STREQUAL

```
1 if(CMAKE_BUILD_TYPE STREQUAL Debug)
2   ...
3 elseif(CMAKE_BUILD_TYPE STREQUAL MinRelInfo)
4   ...
5 else(CMAKE_BUILD_TYPE STREQUAL Debug) # ()
6   ...
7 endif(CMAKE_BUILD_TYPE STREQUAL Debug)
```



# CMake, работа с целями

Добавить исполняемый файл:

```
add_executable(my_exe src/main.cpp)
```

# CMake, работа с целями

Добавить исполняемый файл:

```
add_executable(my_exe src/main.cpp)
```

Добавить библиотеку:

```
add_library(my_lib [STATIC|SHARED] my_lib/my_lib.cpp)
```

# CMake, работа с целями

Добавить исполняемый файл:

```
add_executable(my_exe src/main.cpp)
```

Добавить библиотеку:

```
add_library(my_lib [STATIC|SHARED] my_lib/my_lib.cpp)
```

Связать цель и библиотеку:

```
target_link_libraries(my_exe my_lib  
                      /home/user/3rd/libjson.so  
                      -lpthread  
                      z)
```

# CMake, добавление зависимостей

Добавить библиотеку:

```
add_library(my_lib my_lib/my_lib.cpp)
```

Найти библиотеку в системе:

```
find_package(GLUT)
```

Связать библиотеку с другой библиотекой:

```
target_link_libraries(my_lib PUBLIC GLUT::GLUT)
```

# CMake, добавление опций

Добавляются только для цели и зависимостей:

Добавить флаги компиляции:

```
target_compile_options(my_lib PRIVATE --coverage)
```

Добавить флаги компоновщика:

```
target_link_options(my_lib PRIVATE --coverage)
```

# CMake, добавление опций

Добавляются только для цели и зависимостей:

Добавить определения препроцессора:

```
target_compile_definitions(my_lib PRIVATE -DUSE_LIBZ)
```

Добавить заголовочные файлы:

```
target_include_directories(my_lib PUBLIC include)
```

# CMake, PRIVATE PUBLIC INTERFACE

**PUBLIC** - распространяется на саму цель и все зависимые от нее

**PRIVATE** - распространяется только на саму цель

**INTERFACE** - распространяется только на зависимые цели

Работает с `target_link_libraries`, если линкуем с библиотекой

# CMake, добавление опций

Для текущей и **всех** дочерних директорий:

Добавить заголовочные файлы:

```
include_directories(include)
```

Добавить флаги компиляции:

```
add_compile_options(-Wall)
```

Добавить определения препроцессора:

```
add_definitions(USE_LIBZ)
```



# CMake, структура проекта

```
project/  
├── cmake/ # Модули CMake  
├── docs/  # Документация  
├── examples/ # Примеры использования  
├── include/ # Заголовочные файлы библиотеки  
├── lib/    # Исходные файлы библиотек  
├── src/    # Исходные файлы библиотеки  
└── tests/  # Тесты
```

# CMake, структура проекта

```
lib
├── common
│   ├── CMakeLists.txt
│   ├── include # Заголовочные файлы библиотеки
│   ├── src
│   └── tests
├── parser
│   ├── CMakeLists.txt
│   ├── include
│   ├── src # Исходные файлы библиотеки
│   └── tests
└── utils
    ├── CMakeLists.txt
    ├── include
    ├── src
    └── tests # Тесты
```

# CMake, что почитать

- Документация:  
<https://cmake.org/cmake/help/latest/>
- Mastering CMake. Ken Martin
- Тьюториал <https://cmake.org/cmake/help/latest/guide/tutorial/index.html>

# Практика

# Оставьте обратную связь!

- Обратная связь позволяет нам менять курс сразу, не дожидаясь его завершения.
- Умение оставлять конструктивную обратную связь является одним из важнейших при работе в команде, поэтому воспользуйтесь такой возможностью с целью дополнительного обучения
- Это занимает не более 5 минут и абсолютно анонимно

The image shows a course schedule and a feedback form. The schedule is a horizontal row of six cards for the week of November 1st to 6th. The second card, for Tuesday, November 2nd, is expanded to show two 18:00 sessions of 'Углубленный C/C++' (Advanced C/C++) by A. Khalidzhi. The first session is for 'w...' and the second for '(ML)'. Both sessions have a red 'П' icon. Below the schedule, there is a section for a Zoom link for today at 18:00, titled 'Безопасность интернет-приложений (третий семестр)'. It includes a Zoom link, the conference ID (985 8608 1818), and the access code (785885). Below this, a comment from Алексей Набережный is shown, dated 55 minutes ago. To the right of the Zoom section is a feedback form titled 'Углублённый C/C++'. It indicates 'Лекция 5' (Lecture 5) and 'Онлайн - ML'. A green checkmark and the word 'Посещено' (Attended) are shown. Below this, a message asks for feedback to improve the learning process, and there is a button labeled 'Оставить отзыв' (Leave feedback). A blue arrow points from the 'Оставить отзыв' button in the feedback form to the 'Оставить отзыв' button in the Zoom section.

пн, 1 ноября

вт, 2 ноября

18:00 Углубленный C/C++ (w... П

Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование

А. Халайджи

18:00 Углубленный C/C++ (ML) П

Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование

А. Халайджи

ср, 3 ноября

Нет занятий

чт, 4 ноября

Нет занятий

пт, 5 ноября

Нет занятий

сб, 6 ноября

Нет занятий

Ссылка на Zoom РК сегодня в 18:00

Безопасность интернет-приложений (третий семестр)

Подключиться к конференции Zoom

<https://mailru.zoom.us/j/98586081818?pwd=eWxwSDdCbHhQNnNwQU5lbiFvU2dTZz09>

Идентификатор конференции: 985 8608 1818

Код доступа: 785885

Алексей Набережный 55 минут назад

★ 0 0 ↓ 0 ↑

Углублённый C/C++

Лекция 5

Онлайн - ML

✓ Посещено

Оставьте отзыв о занятии и мы сможем улучшить учебный процесс.

Оставить отзыв

# Спасибо за внимание!

