

# CMake

Михаил Смирнов  
Разработчик C++



# Проверка связи



Поставьте “+”, если меня видно и слышно



## Если у вас нет звука:

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

# Михаил Смирнов

О спикере:

- В C++ разработке с В C++ разработке с 2010 года
- С 2002 года работаю в Муромском Институте Владимирского Государственного Университета
- Цифровая обработка сигналов в радиолокации и гидролокации
- Траекторная обработка для радиолокаторов ближней зоны
- Создание автоматизированного рабочего места для управления гидролокатором



# Вспоминаем прошрое занятие

**Вопрос:** что такое перегрузка операторов?



# Вспоминаем прошрое занятие

**Вопрос:** что такое перегрузка операторов?

**Ответ:** это предоставление своей реализации для обычных операторов типа  $+$ ,  $>$  и др.



# Вспоминаем прошрое занятие

**Вопрос:** какие операторы бывают?



# Вспоминаем прошрое занятие

**Вопрос:** какие операторы бывают?

**Ответ:** перегружаемые - бинарные, унарные и остальные - и неперегружаемые



# Вспоминаем прошрое занятие

**Вопрос:** для чего можно перегрузить операторы?





# Вспоминаем прошрое занятие

**Вопрос:** для чего можно перегрузить операторы?

**Ответ:** для класса или структуры



# Вспоминаем прошрое занятие

**Вопрос:** где и как можно перегрузить операторы?



# Вспоминаем прошрое занятие

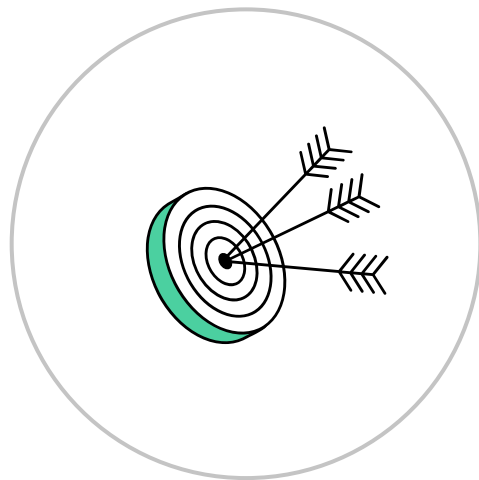
**Вопрос:** для чего можно перегрузить операторы?

**Ответ:** все операторы можно перегрузить внутри класса в виде членов класса. Большинство операторов можно перегрузить вне класса в виде глобальных функций



# Цели занятия

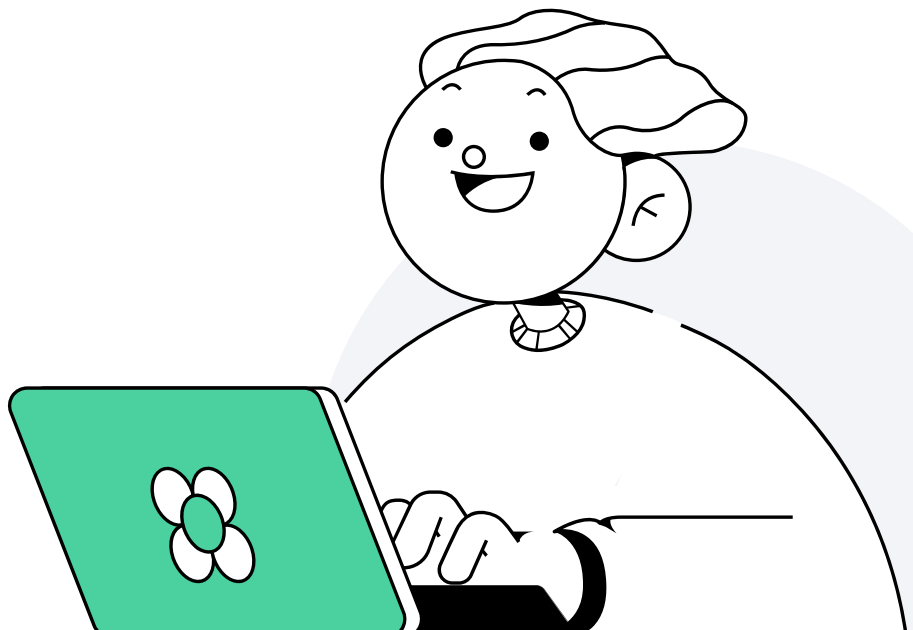
- Разберёмся, зачем понадобились инструменты типа CMake
- Познакомимся с инструментом CMake
- Узнаем, как создавать CMakeLists.txt
- Выясним, как работать с проектом CMake



# План занятия

- 1 [CMake](#)
- 2 [CMake в Visual Studio](#)
- 3 [CMakeLists.txt](#)
- 4 Итоги
- 5 Домашнее задание

\*Нажми на нужный раздел для перехода



# CMake

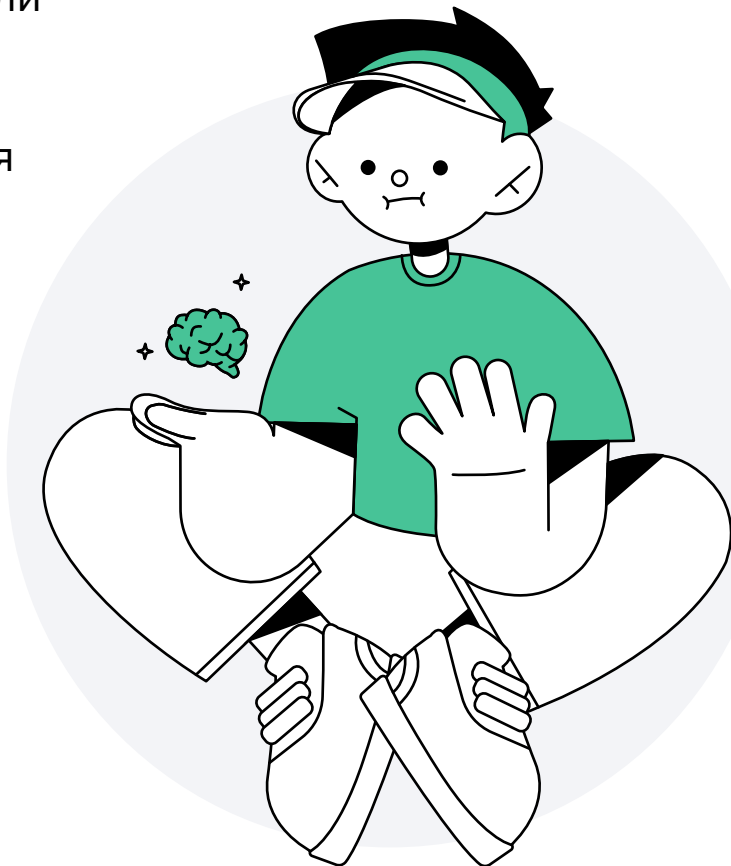


1

# Проблема

C++ в отличие от таких языков, как, например, C# или Java, не является кроссплатформенным

Это значит, что приложение, скомпилированное для конкретной архитектуры процессора и ОС не запустится на других - чтобы запустилось, надо компилировать заново, изменяя параметры компиляции



# Решение

Один из вариантов решения - это сторонний инструмент, который позволяет абстрагировать процесс сборки от архитектуры процессора и ОС

По сути, вы описываете свой проект с помощью языка этого инструмента, а он уже за вас производит компиляцию теми средствами, которые нужны в текущей ситуации



# Инструменты

Существует несколько таких инструментов. Вот некоторые из них:

- GNU Make
- Autotools
- SCons
- Premake
- Ninja
- Meson
- CMake

В этой лекции мы рассмотрим CMake

# Что такое CMake

CMake - это кроссплатформенный инструмент для автоматической сборки программы из исходного кода

При этом сам CMake не занимается именно сборкой - он “рассказывает” разным компиляторам и сборщикам, как им нужно собрать проект

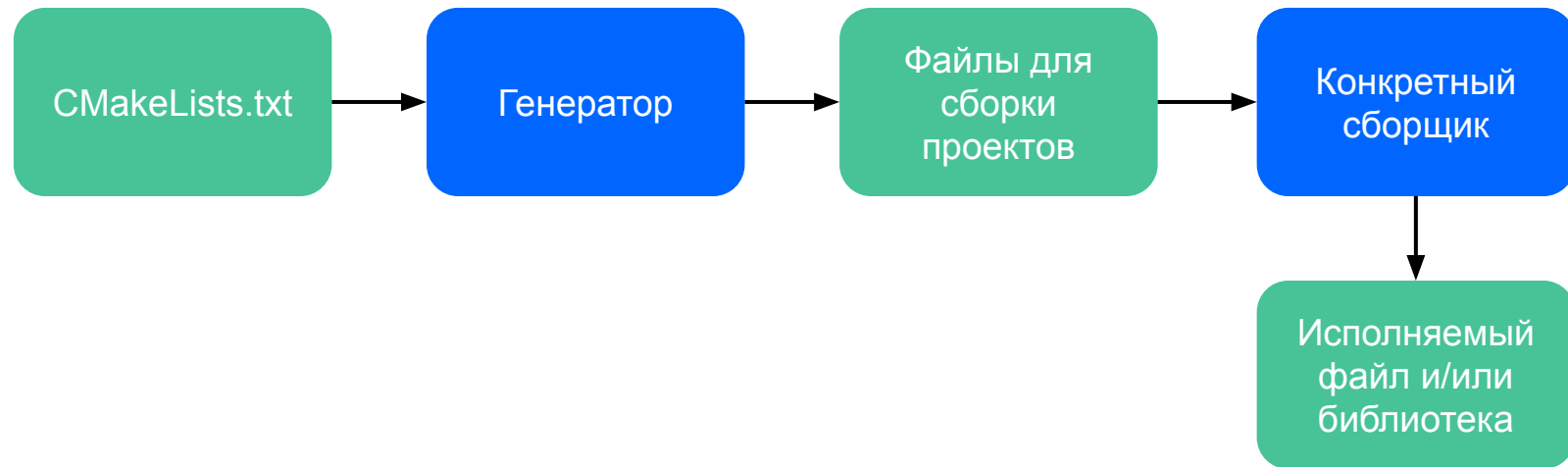
# Артефакты CMake

Прежде, чем пользоваться CMake, нужно установить его или убедиться в том, что он установлен. Этот пункт мы рассмотрим позже

CMake использует информацию из специального файла, который нужно создать программисту - **CMakeLists.txt**

Обычно CMakeLists.txt размещают в корне проекта. Если в вашем решении несколько проектов (а именно так и происходит в сложных программах), то файл CMakeLists.txt размещается в корне каждого проекта, и ещё один в корне решения

# Как работает CMake



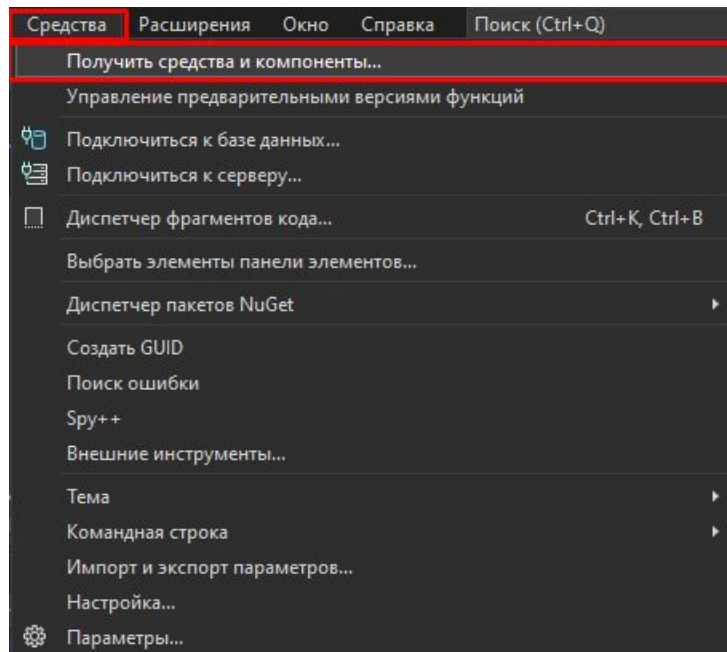
# CMake в Visual Studio



2

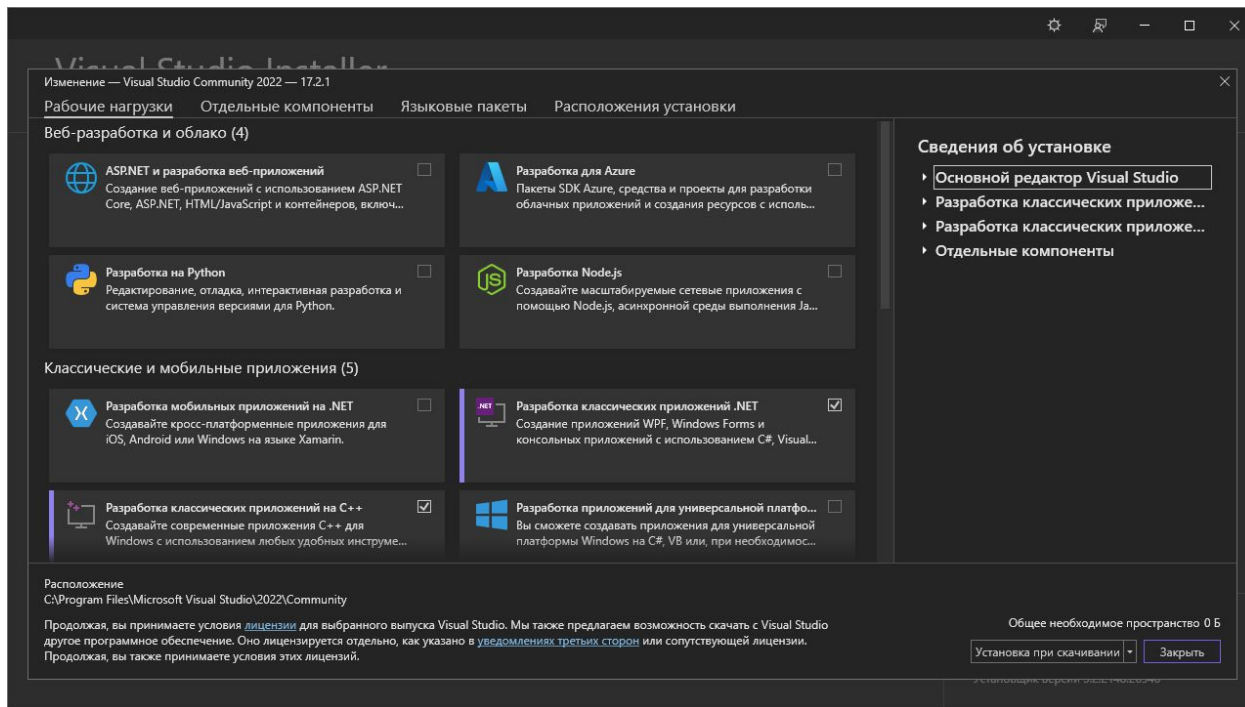
# Удостоверимся, что CMake установлен

Для того, чтобы удостовериться, что CMake установлен на вашем компьютере вместе с Visual Studio, в запущенной Visual Studio откройте в верхнем меню **“Средства” -> “Получить средства и компоненты”**



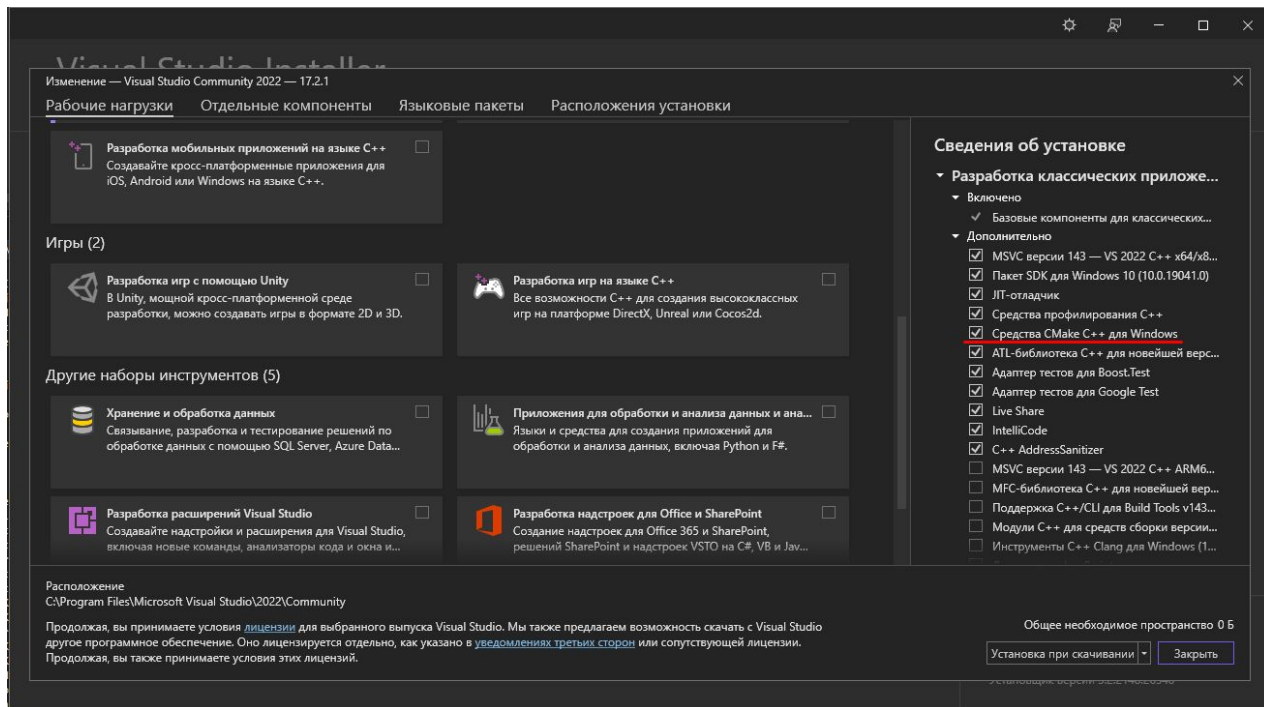
# Удостоверимся, что CMake установлен

У вас запустится установщик Visual Studio. В нём вам нужно посмотреть сведения об установке - они находятся в правой секции во вкладке **“Рабочие нагрузки”**



# Удостоверимся, что CMake установлен

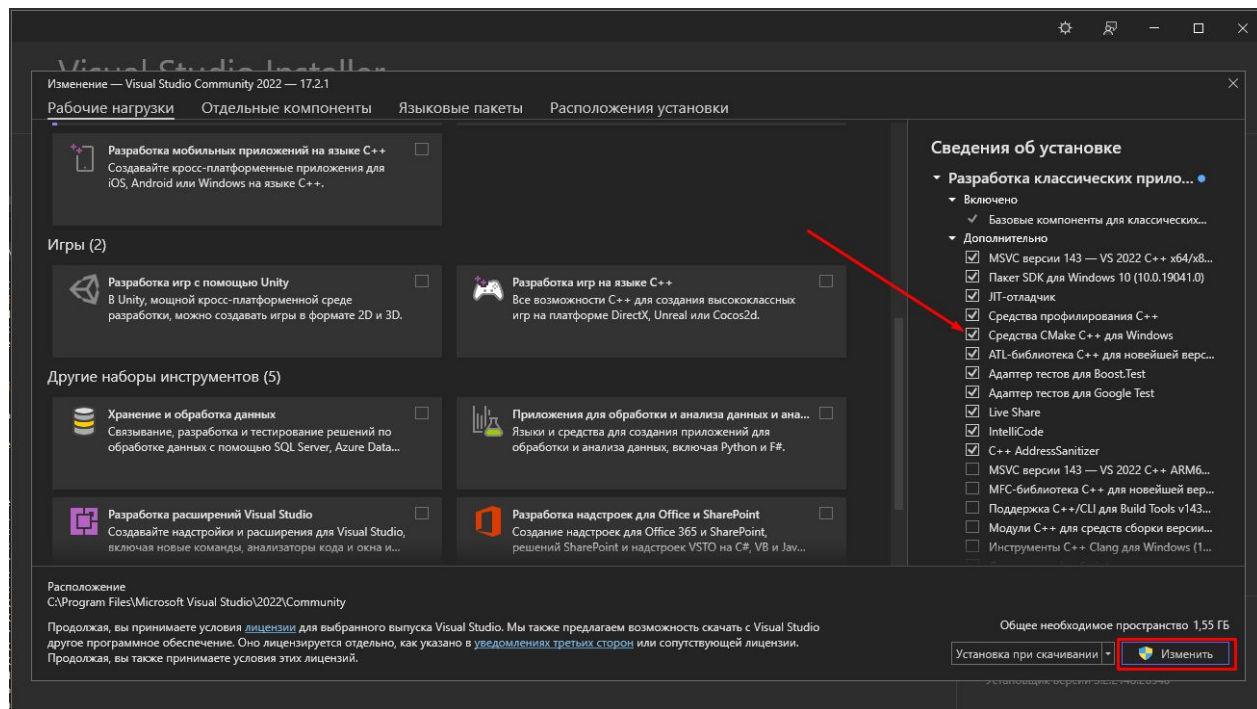
Здесь вам нужно раскрыть пункт **“Разработка классических приложений на C++”** и найти там пункт **“Средства CMake C++ для Windows”**





# Удостоверимся, что CMake установлен

Пункт должен быть отмечен галочкой. Если нет - отметьте его и нажмите появившуюся кнопку **“Изменить”** - установщик добавит CMake в Visual Studio

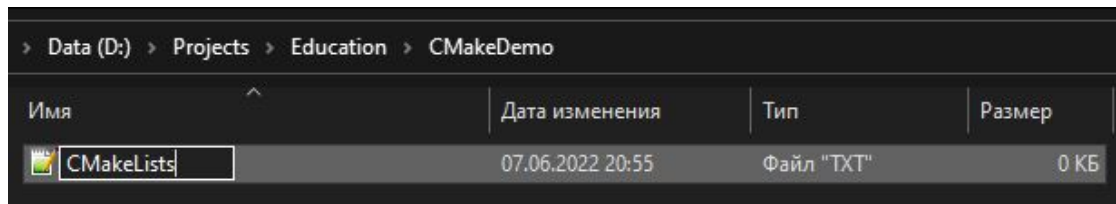
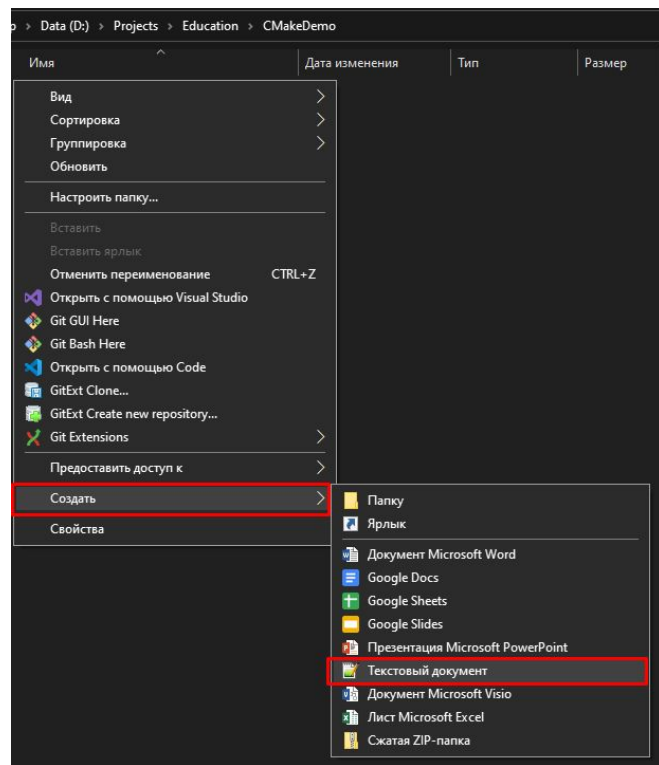


# Создаём файл CMakeLists

Для того, чтобы воспользоваться возможностями Visual Studio по работе с CMake, нам нужно создать файл CMakeLists.txt – именно по его наличию Visual Studio определит, что имеет дело с CMake

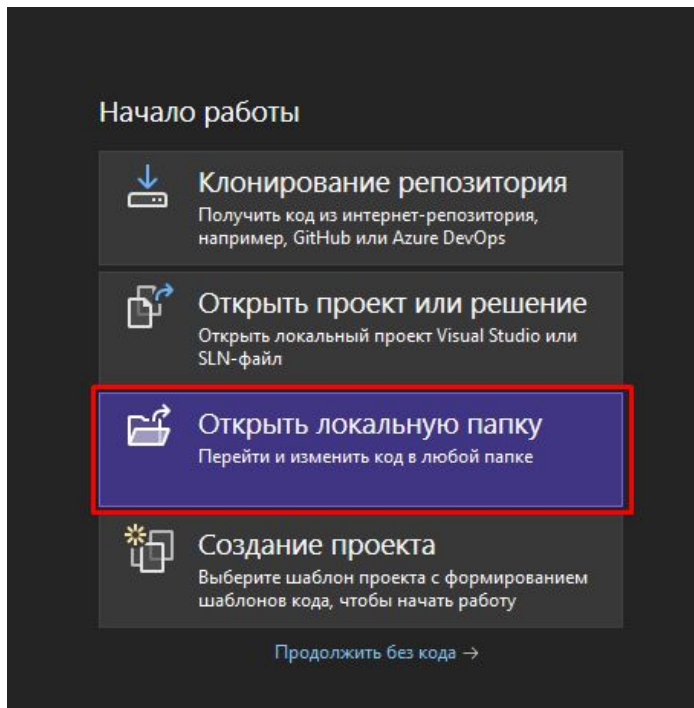
Создайте где-нибудь на компьютере пустую директорию (назовём её CMakeDemo), внутри неё создайте пустой текстовый файл CMakeLists с расширением txt

# Создаём файл CMakeLists



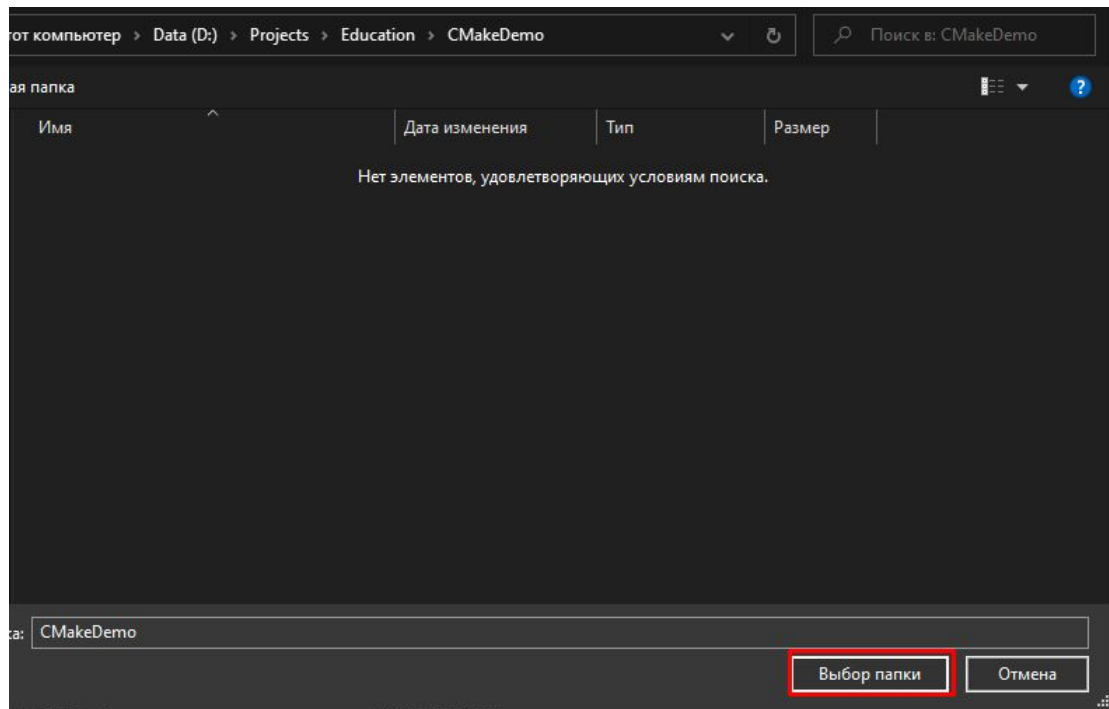
# Открываем Visual Studio

После этого нам нужно открыть нашу папку с помощью Visual Studio. Для этого запустите её и в стартовом окне выберите **“Открыть локальную папку”**



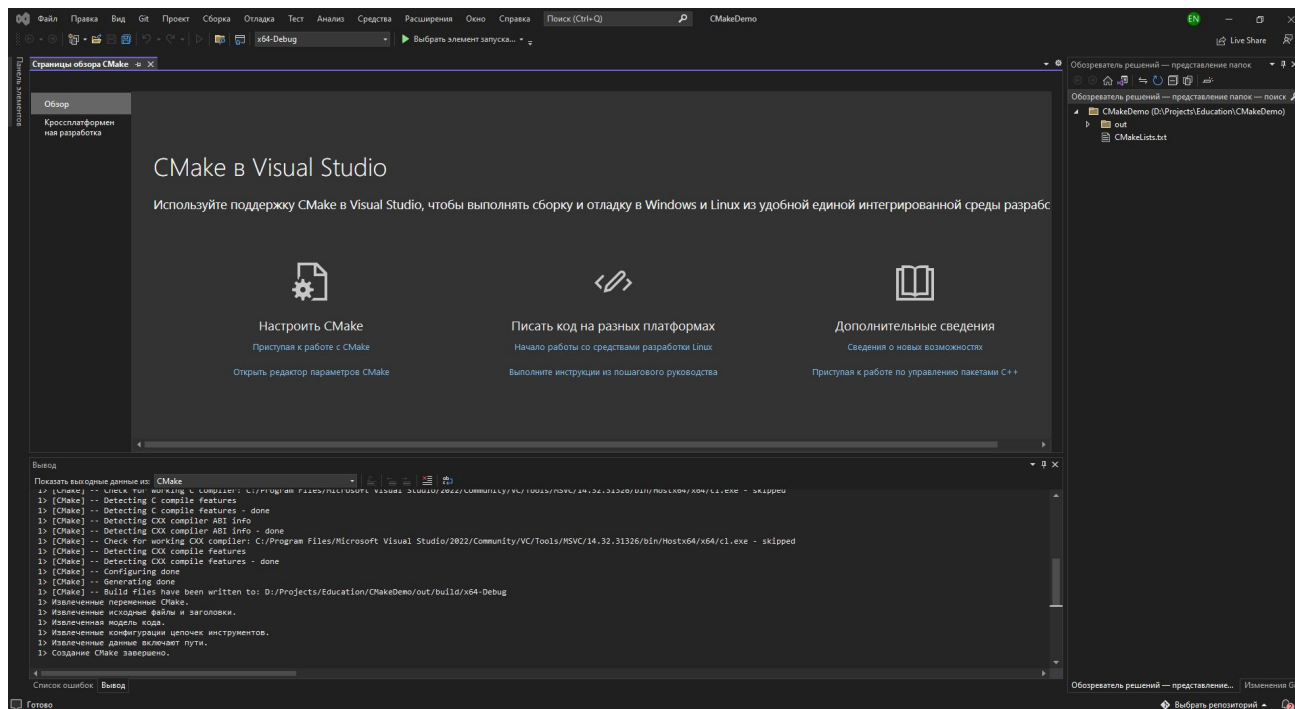
# Открываем Visual Studio

В проводнике выберите **папку**, в которой вы создали пустой файл CMakeLists.txt.  
Щёлкните по кнопке “Выбор папки”



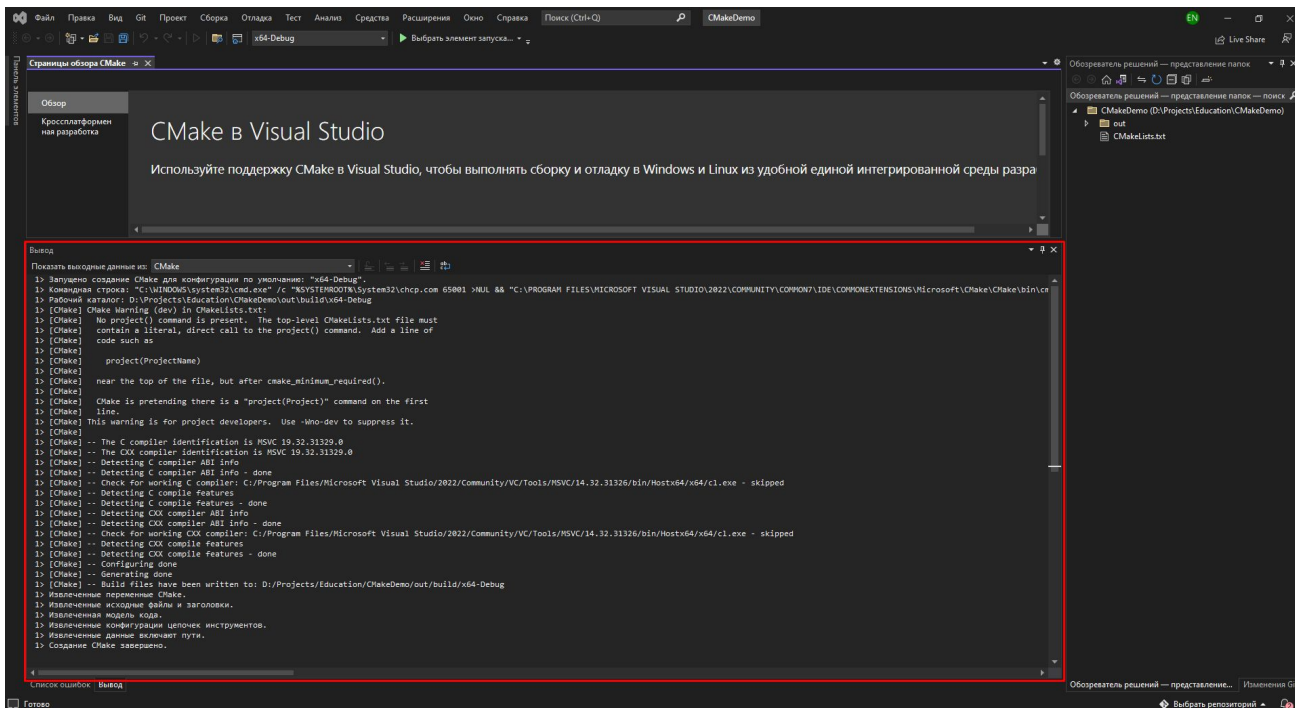
# Открываем Visual Studio

После этого откроется Visual Studio, и будет выглядеть она по-другому, не так, как мы привыкли



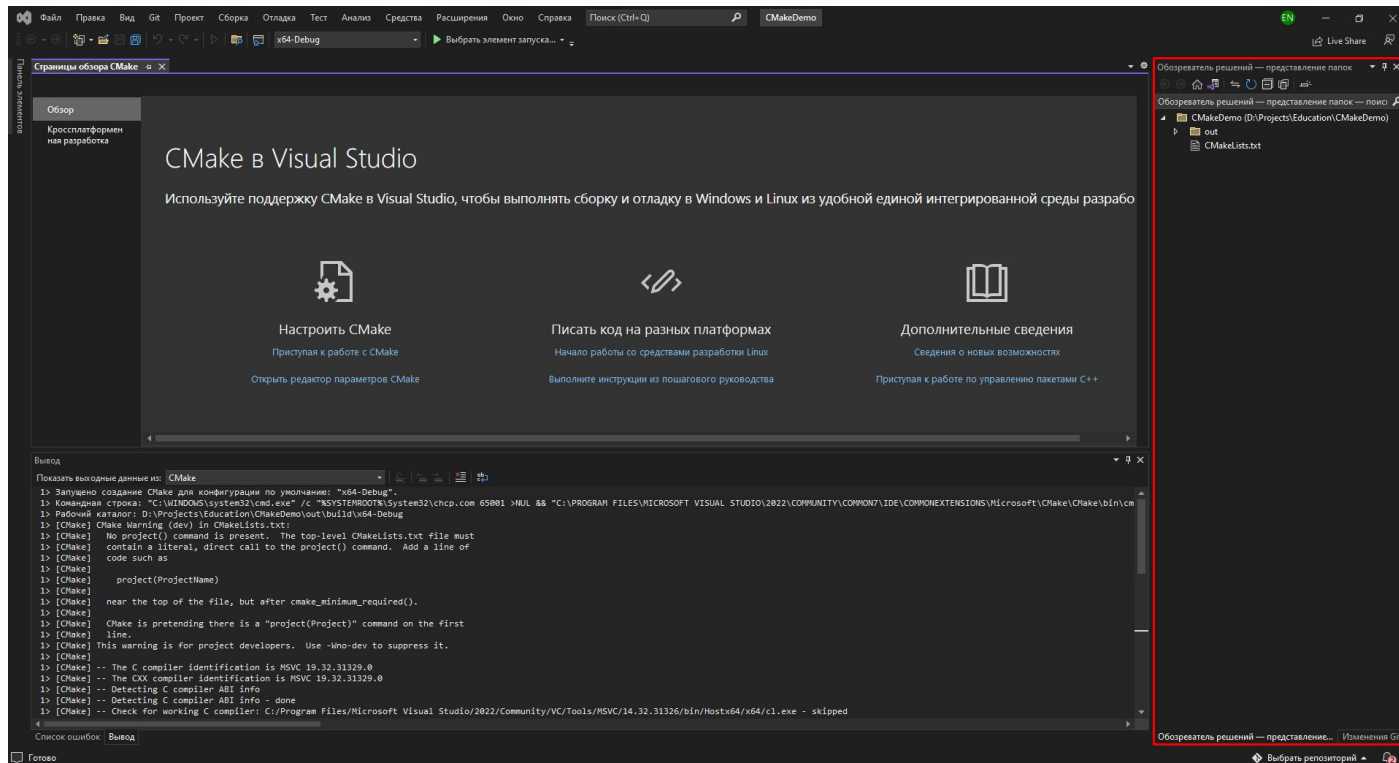
# Открываем Visual Studio

Внизу мы видим результаты обработки установленным в Visual Studio CMake вашего пустого файла CMakeLists.txt



# Открываем Visual Studio

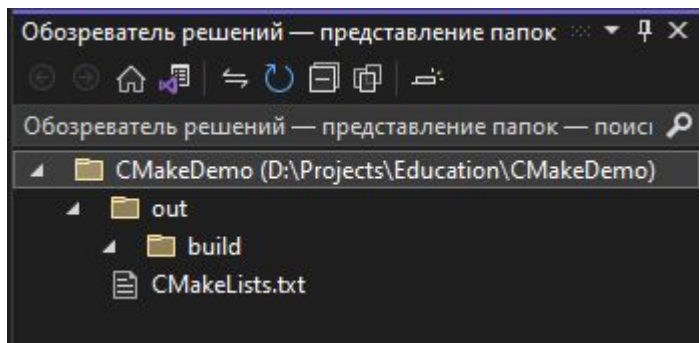
Справа мы увидим Обозреватель решений - представление папок





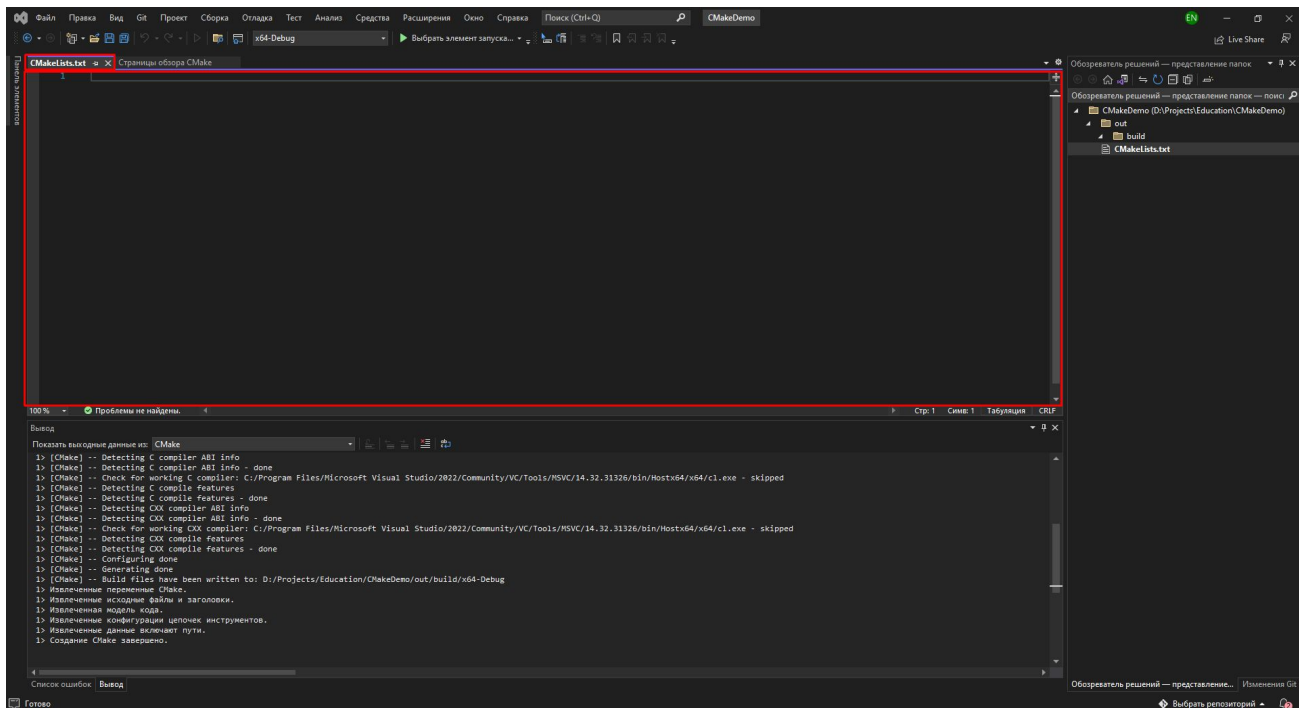
# Открываем Visual Studio

Тут можно увидеть наш файл CMakeLists.txt и папку out, внутри которой есть пустая папка build. Эти папки Visual Studio создал для того, чтобы помещать туда результаты сборки. Откроем файл CMakeLists.txt



# Открываем Visual Studio

Откроем файл CMakeLists.txt и увидим появившийся редактор в центральной области. Здесь мы и будем наполнять файл CMakeLists.txt



# CMakeLists.txt



3

# CMakeLists

Файл CMakeLists.txt содержит информацию о том, из каких исходных файлов состоит проект, как его надо собрать (как исполняемый файл или библиотеку), информацию о самом проекте и т.д.

Эта информация пишется на специальном языке – не C++ – который мы сейчас с вами рассмотрим

# Синтаксис CMakeLists

Общий синтаксис команд CMakeLists выглядит следующим образом:

**<команда>(<аргументы>)**

Синтаксис выглядит похоже на функции в C++, но проще. Разделителем в CMakeLists является пробел, поэтому будьте внимательны при его использовании

**<аргументы>** разделяются пробелом

Рассмотрим некоторые команды, с помощью которых мы построим простейший проект

# Команда версии

Часто файлы CMakeLists начинаются с команды, которая устанавливает **минимальную** версию CMake. Если файл CMakeLists будет обрабатываться CMake версии ниже указанной, то обработка остановится на этой команде с ошибкой

`cmake_minimum_required(VERSION <версия>)`

Visual Studio 2022 поддерживает версию 3.22.2 - это значит, что вам нужно указывать версию **не выше** этой

Добавим в наш файл команду

```
cmake_minimum_required(VERSION 3.22.0)
```

# Команда проекта

После версии обычно указывают название проекта, для которого пишется этот CMakeLists. Команда выглядит вот так:

**project(<название проекта> [<другие аргументы>])**

<другие аргументы> могут содержать версию нашего проекта и информацию о языке, но нам сейчас это не надо

Назовём наш проект cmake\_demo и добавим в файл команду:

```
project(cmake_demo)
```

# Команда создания исполняемого файла

Для нашего простого примера нам не хватает одной команды – собственно указания, какие файлы надо собрать и что должно из них получиться

Мы всё это время создавали запускаемые программы (скоро научимся делать библиотеки), поэтому посмотрим на команду для создания исполняемой программы:

**add\_executable(<название> [<другие аргументы>] <исходные файлы>)**

**<название>** должно быть уникальным в рамках проекта - на его основе будет назван исполняемый файл нашей программы

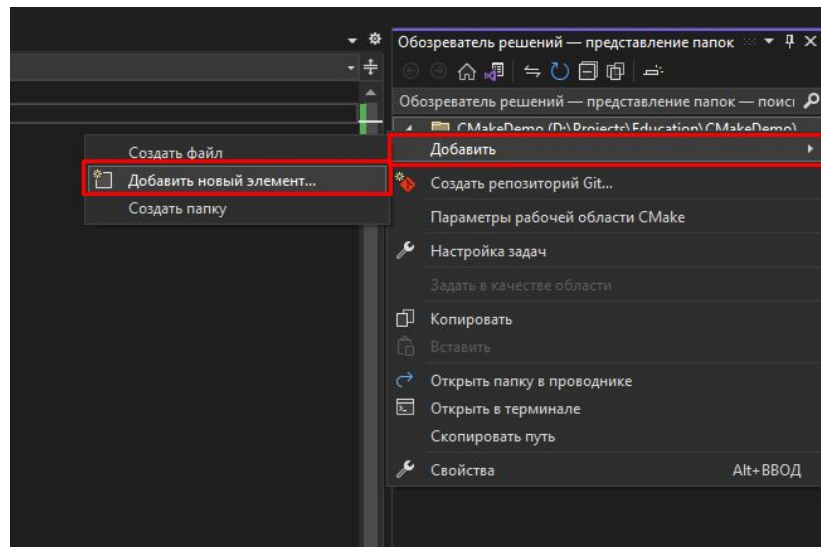
**<исходные файлы>** – это пути (включая названия) исходных файлов, из которых будет скомпилирована программа. Сейчас у нас их нет, поэтому создадим один



# Добавляем файл исходного кода

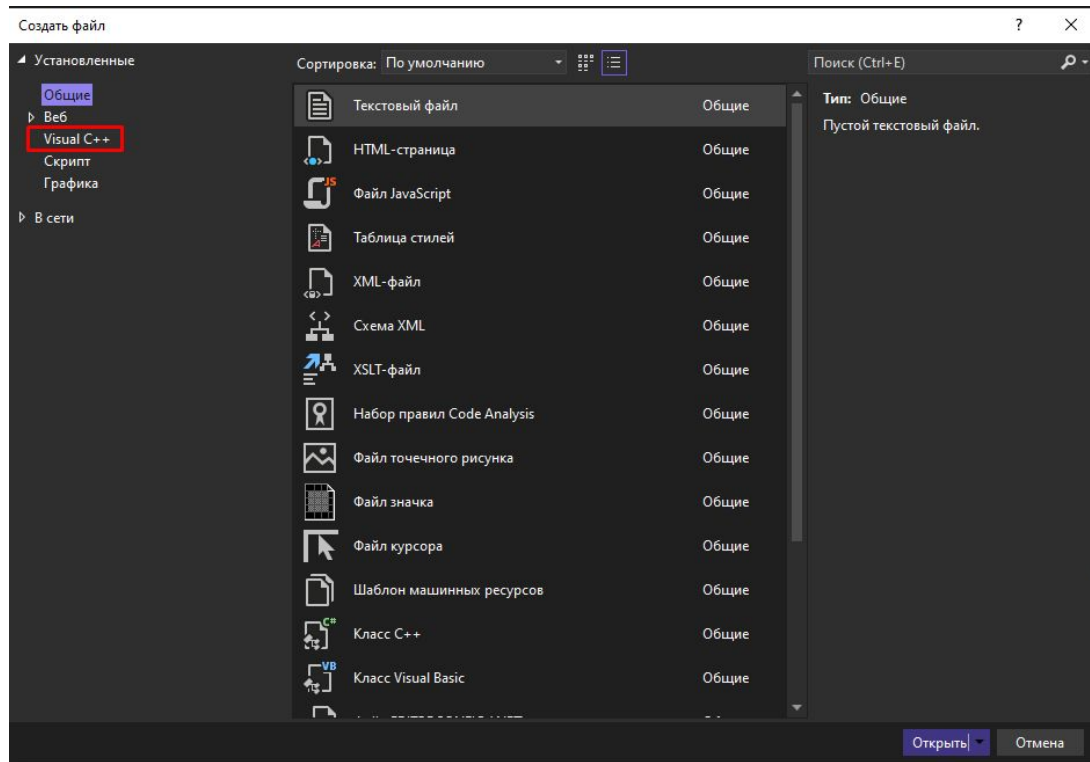
Наш проект простой и будет содержать один файл **source.cpp**, в котором мы выведем на консоль Hello world!

Создадим этот файл - для этого щёлкнем правой кнопкой по папке CMakeDemo в обозревателе решений, выберем пункт **“Добавить”** -> **“Добавить новый элемент”**



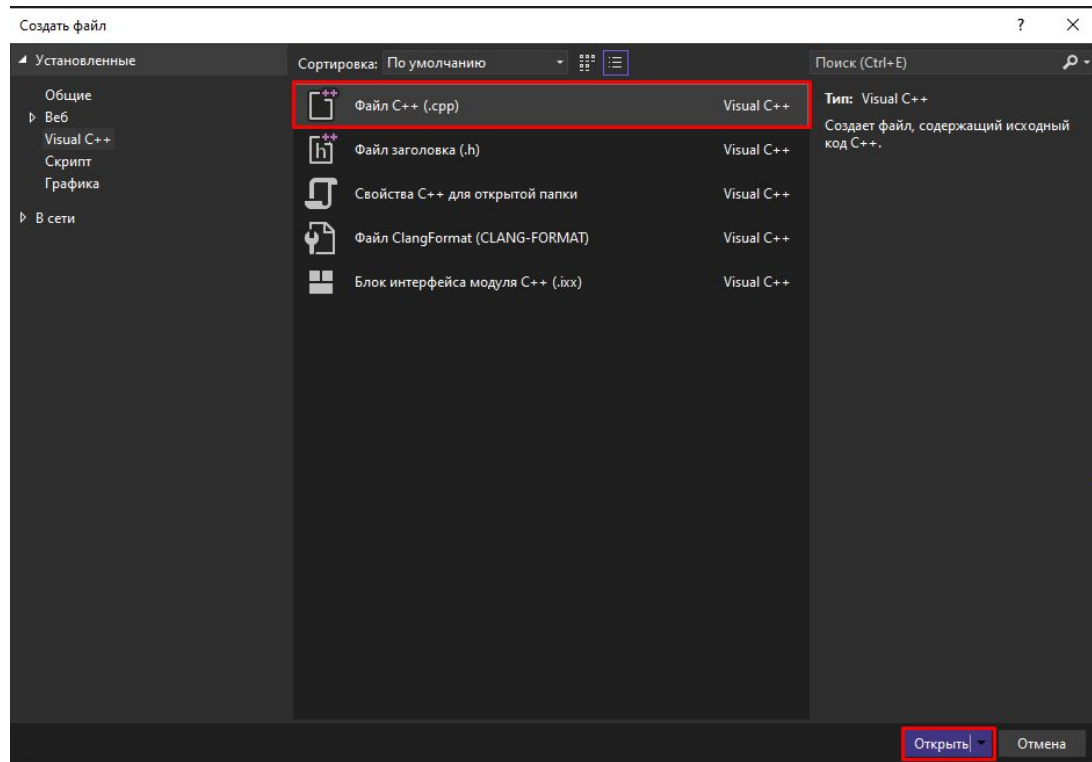
# Добавляем файл исходного кода

В появившемся окне выберите категорию **Visual C++**



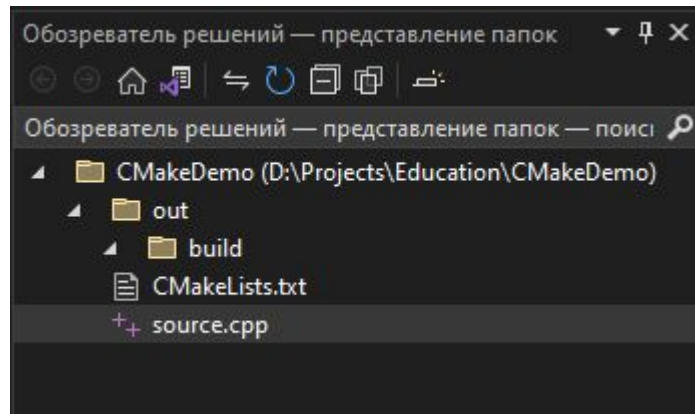
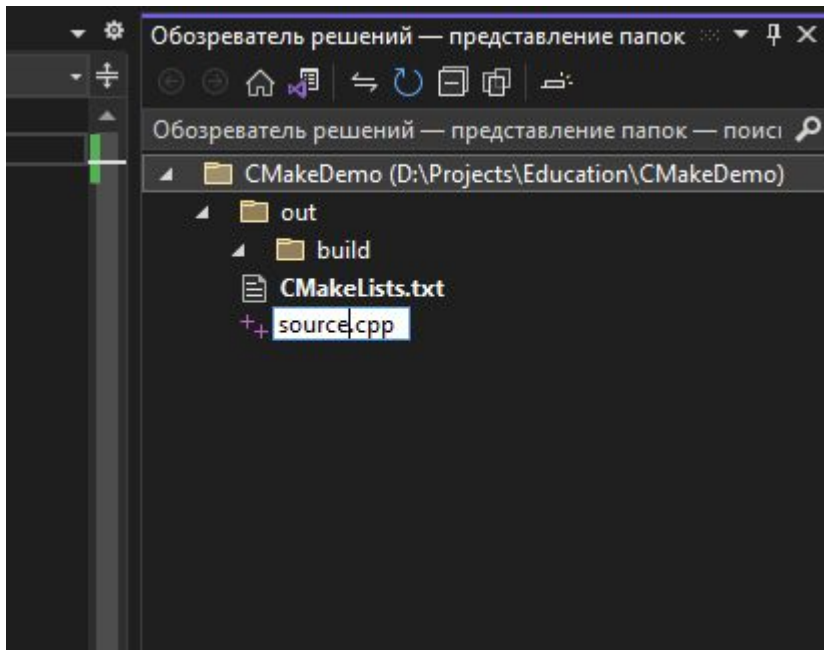
# Добавляем файл исходного кода

В категории Visual C++ выберите **“Файл C++ (.cpp)”** и нажмите **“Открыть”**



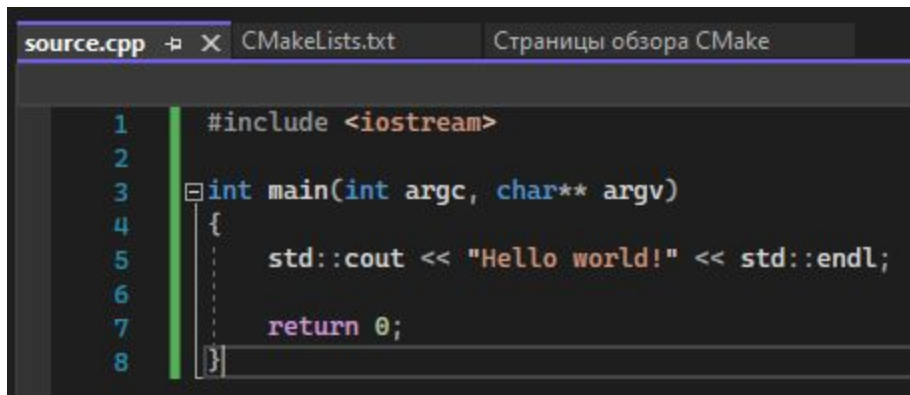
# Добавляем файл исходного кода

В Обозревателе решений появится файл исходного кода - назовите его **source.cpp**



# Добавляем код

Откройте файл source.cpp и напишите код для вывода фразы “Hello world!” на экран



```
source.cpp  X  CMakeLists.txt  Страницы обзора CMake

1  #include <iostream>
2
3  int main(int argc, char** argv)
4  {
5      std::cout << "Hello world!" << std::endl;
6
7      return 0;
8  }
```

# Команда создания исполняемого файла

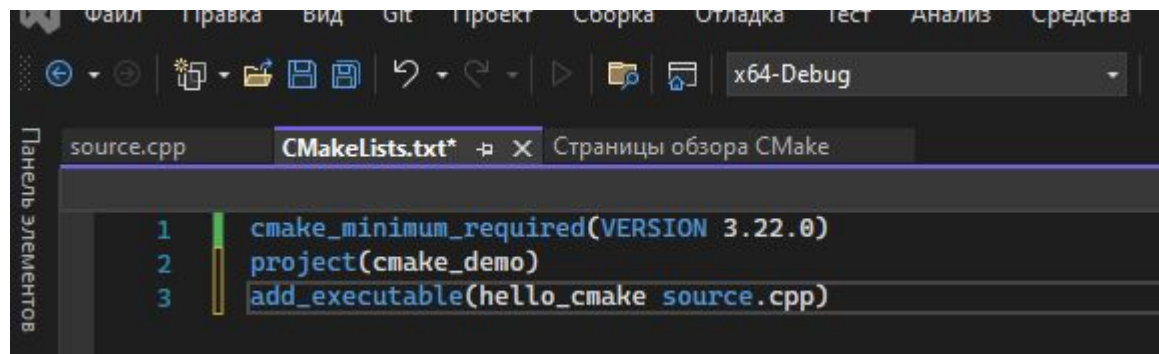
Теперь мы можем добавить команду для создания исполняемой программы в наш CMakeLists.txt

Назовём её hello\_cmake и добавим в CMakeLists:

```
add_executable(hello_cmake source.cpp)
```

# Получившиеся команды

В итоге наш CMakeLists должен выглядеть вот так.



The screenshot shows an IDE window with a menu bar (Файл, Правка, Вид, Git, Проект, Сборка, Отладка, Тест, Анализ, Средства) and a toolbar. The active file is `CMakeLists.txt*`. The code in the editor is as follows:

```
1 cmake_minimum_required(VERSION 3.22.0)
2 project(cmake_demo)
3 add_executable(hello_cmake source.cpp)
```

# Сохраняем

Сохраните его, нажав **иконку дискеты** в панели инструментов или сочетание клавиш **Ctrl+S** - после этого запустится обработка файла и создание промежуточных файлов

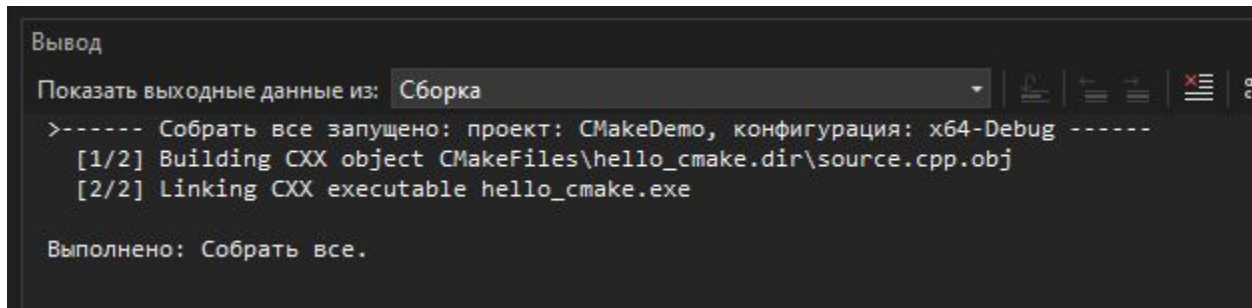
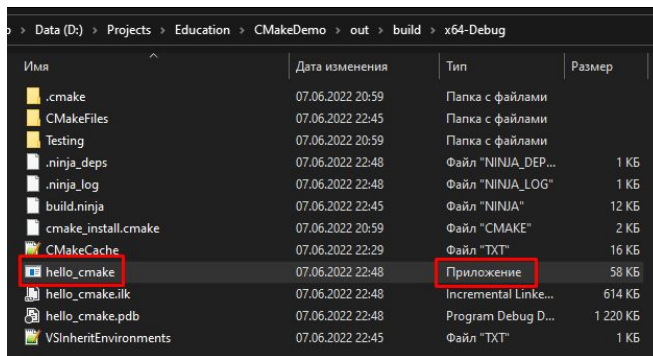
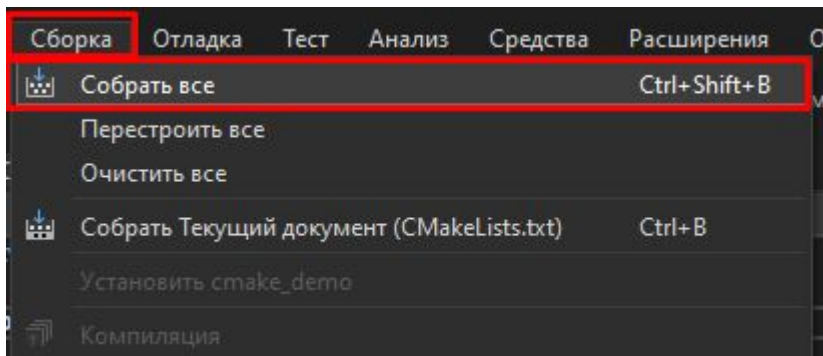
Показать выходные данные из: CMake

```
1> Запущено создание CMake для конфигурации по умолчанию: "x64-Debug".
1> Командная строка: "C:\WINDOWS\system32\cmd.exe" /c "%SYSTEMROOT%\System32\chcp.com 65001 >NUL && "C:
1> Рабочий каталог: D:\Projects\Education\CMakeDemo\out\build\x64-Debug
1> [CMake] -- Configuring done
1> [CMake] -- Generating done
1> [CMake] -- Build files have been written to: D:/Projects/Education/CMakeDemo/out/build/x64-Debug
1> Извлеченные переменные CMake.
1> Извлеченные исходные файлы и заголовки.
1> Извлеченная модель кода.
1> Извлеченные конфигурации цепочек инструментов.
1> Извлеченные данные включают пути.
1> Создание CMake завершено.
```



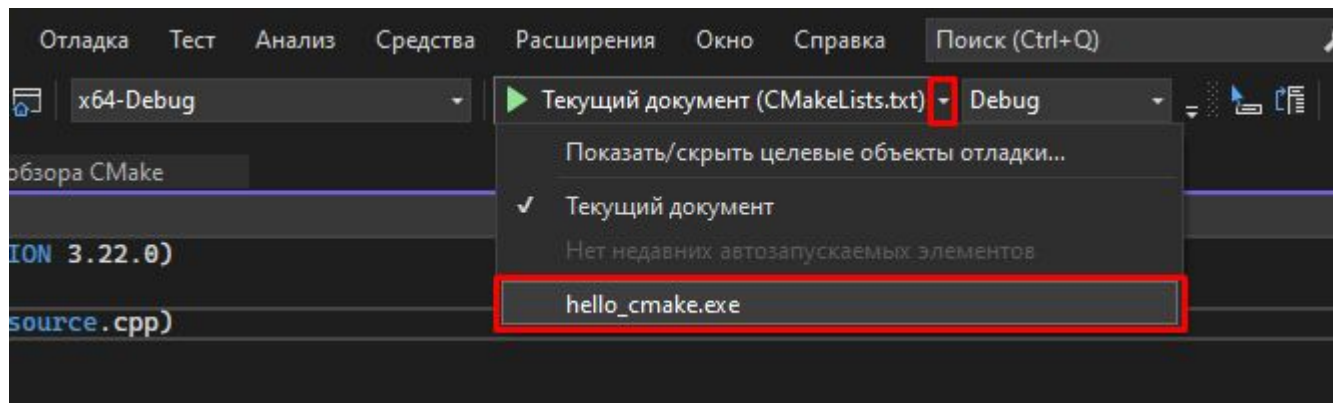
# Собираем

Теперь мы можем собрать решение - выбираем в верхнем меню **“Сборка”** -> **“Собрать всё”**. Произойдёт сборка и в папке out\build\x64-Debug появится наша программа hello\_cmake



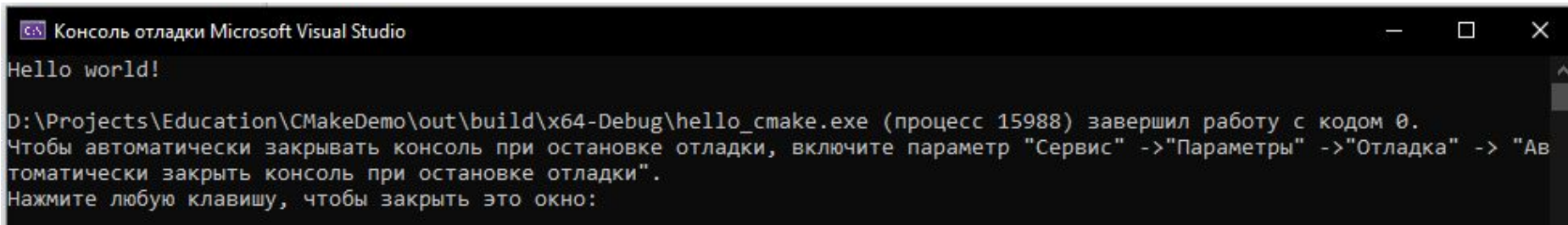
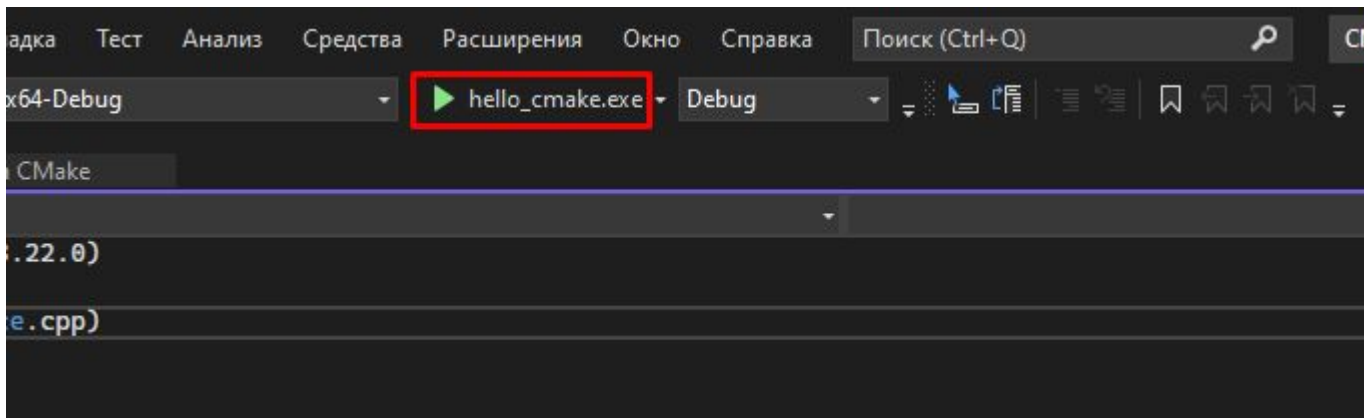
# Собираем и запускаем

Можно сделать так же, как и в случае с обычными проектами - запустить прямо из Visual Studio. Для этого щёлкните по **стрелочке справа от кнопки запуска** и выберите нужную программу - она у нас одна, это **hello\_cmake.exe**



# Собираем и запускаем

После этого щёлкните по **кнопке запуска** (или “Отладка” -> “Начать отладку”)



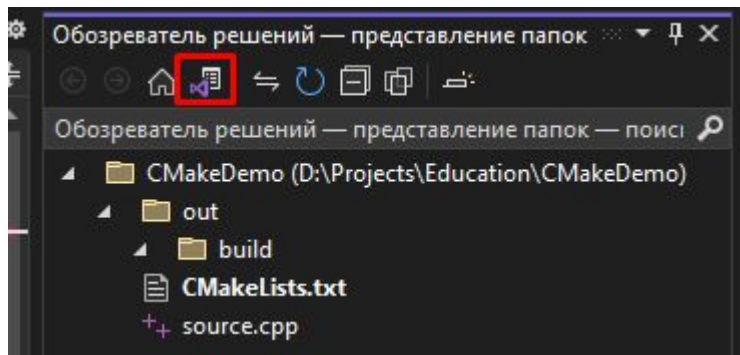
# Отладка

Отладка выполняется точно так же, как для обычных проектов C++ – с помощью точек останова

# Обозреватель решений

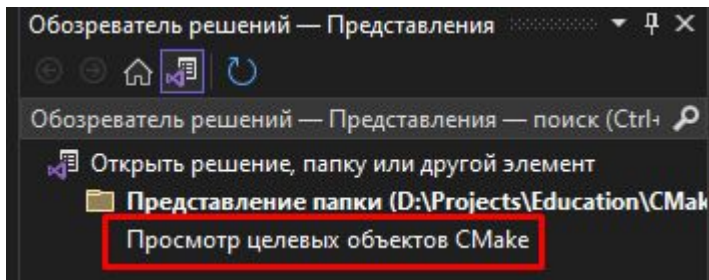
Стоит отметить, что в обозревателе решений можно переключиться с представления папок на просмотр целевых объектов - с помощью него будет видно структуру вашего проекта из CMakeLists

Для этого в панели инструментов Обозревателя решений щёлкните по **иконке со значком документа и Visual Studio**



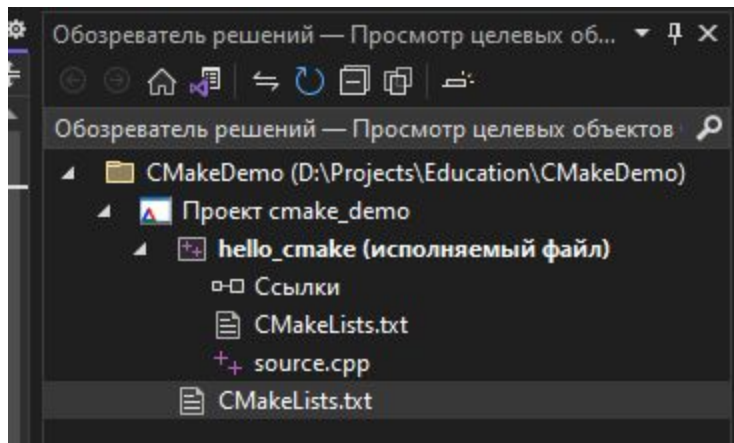
# Обозреватель решений

У вас появится список доступных представлений. Выбранное представление выделено жирным шрифтом. Выберите **“Просмотр целевых объектов CMake”** - щёлкните два раза левой кнопкой мыши



# Просмотр целевых объектов CMake

У вас появится структура вашего проекта из CMakeLists - здесь тоже можно добавлять файлы кода, целевые объекты и в целом удобнее управлять решением



# Итоги





# Итоги занятия

Сегодня мы

- 1 Разобрались, зачем понадобились инструменты типа CMake
- 2 Познакомились с инструментом CMake
- 3 Узнали, как создавать CMakeLists.txt
- 4 Выяснили, как работать с проектом CMake



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



# Дополнительные материалы

- [Для чего нужен CMake](#)



**Задавайте вопросы  
и пишите отзыв о лекции**

