Модуль 4. Вопросы.

1. **Компиляция и линковка**

   a. **Этапы компиляции**
   Что такое файл исходного кода и исполняемый файл? Этапы компиляции: препроцессинг, компиляция, ассемблирование, линковка. Что происходит на каждом из этапе компиляции? Как выполнить только часть этапов компиляции?

   b. **Раздельная компиляция**
   В чем преимущества разделения программы на несколько файлов? В чём преимущества и недостатки раздельной компиляции? Как разделить программу на файлы так, чтобы происходила раздельная компиляция? Как программа компилируется при раздельной компиляции?

   c. **Заголовочные файлы**
   Что такое заголовочные файлы (header-файлы)? Что обычно хранится в заголовочных файлах? Что делает директива препроцессора `#include`? Проблема множественного включения. Стражи включения и директива `#pragma once`.

   d. **Имена**
   Что такое единица трансляции? Понятия объявления (англ. *declaration*) и определения (англ. *definition*) сущностей. Объявление функций. Определение функций. Объявление глобальный переменных. Определение глобальных переменных. Объявление и определение структур. Понятие *имя* в контексте линковки на языке C. Понятие *символ* в контексте линковки на языке C. Внутреннее связывание, внешнее связывания и отсутствие связывания. Ключевые слова `static` и `extern`. Ключевое слово `inline`. Использования утилит `nm` и `objdump` для просмотра имён в скомпилированных файлах.

   e. **Опции компиляции**
   - Опция для задания названия выходного файла.
   - Опции для выполнения только первых этапов компиляции
   - Опции для выбора стандарта языка
   - Опции для работы с предупреждениями (`-Wall`, `-Wextra`, `-Werror`)
   - Опции для оптимизации кода
   - Опция для включения информации для дебаггера
   - Опция для определения макросов (определений компиляции)
   - Опция `-DNDEBUG`
   - Опции для подключения библиотек

   f. **Библиотеки**
   Что такое библиотека? Виды библиотек: header-only библиотеки, статические библиотеки, динамические библиотеки. В чём различия между этими видами библиотек? В чём преимущества и недостатки каждого из видов библиотек?

   g. **Статические библиотеки**
   Как создать статическую библиотеку? Утилита `ar`. Как подключить статическую библиотеку, с помощью компилятора `gcc`? Опции компилятора `-I`, `-L` и `-l`. Характерные расширения файлов статических библиотек на Linux (gcc) и Windows (MinGW и MSVC).

   h. **Динамические библиотеки**
   В чём главная разница между статическими и динамическими библиотеками? Как создать динамическую библиотеку? Что делает опция `-fPIC`? Характерные расширения файлов динамических библиотек на Linux (gcc) и Windows (MinGW и MSVC). Подключение динамической библиотеки при запуске программы. Опции компилятора `-I`, `-L` и `-l`. Подключение динамической библиотеки во время выполнения программы. Библиотека `dlfcn.h`. Функции `dlopen`, `dlsym` и `dlclose`.

   i. **Особенности компиляции и линковки на языке C++ по сравнению с языком C**
   - Раздельная компиляция при наличии в коде классов и шаблонов.
   - Манглирования имён. `extern "C"`.
   - ODR. Использования ключевого слова `inline` для обхода ODR.
   - Инициализация статических полей класса.

2. **Основы CMake**

   a. **Системы сборки**
   Задачи, которые решают системы сборки. Примеры систем сборки. Генераторы систем сборки. Преимущества генераторов систем сборки по сравнению с обычными системами сборки. Примеры генераторов систем сборки. Как задать генератор в CMake? Опция `-G` программы `cmake`.

   b. **Простейшие проекты на CMake**
   Файл `CMakeLists.txt`. Команды:

   - `cmake_minimum_required`
   - `project`
   - `add_executable`

   Создание и подключение библиотек в CMake. Команды:

   - `add_library`
   - `target_link_libraries`

   c. **Таргеты**
   Понятие *таргет* в CMake. Подключение файлов к таргетам с помощью команды:

   - `target_sources`

   Подключение опций к таргетам с помощью команд:

   - `target_include_directories`
   - `target_link_directories`
   - `target_link_libraries` (не путать с другой командой с таким же названием)
   - `target_compile_definitions`
   - `target_compile_features`
   - `target_compile_options`
   - `target_link_options`

   Использование ключевых слов `PUBLIC`, `PRIVATE` и `INTERFACE` при подключении опций к таргетам и при подключении таргетов друг к другу.

   d. **Переменные CMake**
   Язык CMake. Создание переменных в CMake. Команда `set`. Какого типа может быть переменная в CMake? Использование переменных. Переменные-строки. Команда `string` и её опции:

   - `FIND`
   - `REPLACE`
   - `APPEND`
   - `JOIN`
   - `TOLOWER`
   - `TOUPPER`
   - `LENGTH`
   - `SUBSTRING`
   - `COMPARE`

   Переменные-списки. Команда `list` и её опции:

   - `LENGTH`
   - `GET`
   - `JOIN`
   - `FIND`
   - `APPEND`
   - `INSERT`
   - `REMOVE_ITEM`
   - `REMOVE_AT`
   - `SORT`

   e. **Команда `if` и циклы**
   Как работает команда `if`. Какие строки считаются истинными, а какие – ложными? Цикл `while`. Цикл `foreach`. Разновидности цикла `foreach`: `RANGE`, `IN ITEMS` и `IN LISTS`.

   f. **Функции CMake**
   Сколько аргументов могут принимать функции в CMake? Команда `function` для создания функций. Передача аргументов в функции. Переменные `ARGC`, `ARGV` и `ARGN`. Возврат из функций. `PARENT_SCOPE`.

   g. **Работа с файлами в CMake. Команда `file` и её опции.**

   - `READ`
   - `STRINGS`
   - `WRITE`
   - `APPEND`
   - `GLOB`
   - `MAKE_DIRECTORY`
   - `COPY_FILE`
   - `SIZE`
   - `DOWNLOAD`

3. **CMake: структура проекта, кэшированные переменные, конфиги, генераторные выражения**

   a. **Разделение кода CMake на несколько директорий**
   Поддиректории в CMake. Команда `add_subdirectory`. Область видимости переменных, объявленных в поддиректории и вне поддиректории. Опция `PARENT_SCOPE` команды `set`. Переменные:

   - `CMAKE_SOURCE_DIR`.
   - `CMAKE_BINARY_DIR`.
   - `CMAKE_CURRENT_SOURCE_DIR`
   - `CMAKE_CURRENT_BINARY_DIR`

   b. **Модули**
   Модули CMake. Скриптовый режим. `cmake -P`. Подключение модуля. Команда `include`. Переменные:

   - `CMAKE_MODULE_PATH`.
   - `CMAKE_CURRENT_LIST_DIR`
   - `CMAKE_CURRENT_LIST_FILE`

   Область видимости переменных, объявленных в модуле и вне модуля.

   c. **Кэшированные переменные**
   Что такое кэшированная переменная CMake и чем она отличается от обычной? Зачем нужны кэшированные переменные? Создание кэшированных переменных с помощью команды `set`. Создание кэшированных переменных с помощью команды `option`. Как изменить уже созданную кэшированную переменную? Опция `FORCE` команды `set`. Файл `CMakeCache.txt`. Опция `-D` программы `cmake`. Программы `ccmake` и `cmake-gui`. "Тип" кэшированной переменной и как он используется? Совпаденение имени обычной и кэшированной переменной.

   d. **Предопределённые переменные**
   Переменные, хранящие информацию о проекте:

   - `PROJECT_NAME`
   - `PROJECT_SOURCE_DIR`
   - `PROJECT_BINARY_DIR`
   - `PROJECT_IS_TOP_LEVEL`

   Переменные, задающие глобальные опции компиляции:

   - `CMAKE_C_FLAGS`
   - `CMAKE_CXX_FLAGS`
   - `CMAKE_EXE_LINKER_FLAGS`

   Переменные, предоставляющие информацию о платформе:

   - `CMAKE_C_COMPILER`
   - `CMAKE_CXX_COMPILER`
   - `CMAKE_GENERATOR`
   - `CMAKE_SYSTEM_NAME`
   - `UNIX`
   - `APPLE`
   - `WIN32`
   - `MSVC`

   Информация о самой программе `cmake`:

   - `CMAKE_ROOT`
   - `CMAKE_VERSION`

   e. **Свойства**
   Что такое свойства в CMake? Чем свойства отличаются от переменных? Команды `get_property` и `set_property`. Глобальные свойства. Свойства директорий:

   - `VARIABLES`
   - `CACHE_VARIABLES`
   - `SUBDIRECTORIES`
   - `BUILDSYSTEM_TARGETS`
   - `IMPORTED_TARGETS`
   - `TESTS`
   - `PARENT_DIRECTORY`

Свойства таргетов:

- INCLUDE_DIRECTORIES
- COMPILE_DEFINITIONS
- COMPILE_FEATURES
- COMPILE_OPTIONS
- LINK_DIRECTORIES
- LINK_LIBRARIES
- LINK_OPTIONS
- INTERFACE_INCLUDE_DIRECTORIES
- INTERFACE_COMPILE_DEFINITIONS
- INTERFACE_COMPILE_FEATURES
- INTERFACE_COMPILE_OPTIONS
- INTERFACE_LINK_DIRECTORIES
- INTERFACE_LINK_LIBRARIES
- INTERFACE_LINK_OPTIONS
- SOURCES
- TYPE
- IMPORTED
- OUTPUT_NAME
- OUTPUT_NAME_<CONFIG>
- ARCHIVE_OUTPUT_DIRECTORY
- ARCHIVE_OUTPUT_DIRECTORY_<CONFIG>
- LIBRARY_OUTPUT_DIRECTORY
- LIBRARY_OUTPUT_DIRECTORY_<CONFIG>
- RUNTIME_OUTPUT_DIRECTORY
- RUNTIME_OUTPUT_DIRECTORY_<CONFIG>
- C_STANDARD
- CXX_STANDARD

Свойства файлов исходного кода. Свойства тестов.

f. **Типы сборки**

Основные типы сборки:

- (Пустой тип сборки)
- Release
- Debug
- RelWithDebInfo
- MinSizeRel

Как тип сборки влияет на опции компиляции? Одноконфигурационные и мультиконфигурационные генераторы. Как задать тип сборки в случае одноконфигурационного генератора и в случае мультиконфигурационного генератора? Переменные:

- CMAKE_BUILD_TYPE
- CMAKE_CONFIGURATION_TYPES

Глобальное свойство:

- GENERATOR_IS_MULTI_CONFIG

Как написать код, который бы работал для одноконфигурационных и мультиконфигурационных генераторов. Создание пользовательского типа сборки. Кэшированные переменные:

- CMAKE_CXX_FLAGS_<CONFIG>
- CMAKE_C_FLAGS_<CONFIG>
- CMAKE_EXE_LINKER_FLAGS_<CONFIG>

g. **Генераторные выражения**

Этапы сборки CMake-проекта:

i. Этап конфигурации
ii. Этап генерации
iii. Этап сборки

Генераторные выражения. Привести пример применения генераторного выражения. Какие команды CMake поддерживают генераторные выражения? Основные виды генераторных выражений:

- $<условие:значение>
- $<BOOL:значение>
- Логические операции OR, AND, NOT.
- $<CONFIG:тип_сборки>
- $<PLATFORM_ID:имя_платформы>
- $<TARGET_FILE:таргет>

4. **CMake: подключение сторонних библиотек**

   a. **Ручное подключение сторонней библиотеки**
   Как подключить библиотеку, если есть только файлы самой библиотеки без скриптов сборки? Преимущества и недостатки этого метода подключения сторонних библиотек.

   b. **Подключение с помощью** `add_subdirectory`
   Как подключить другой CMake-проект с помощью `add_subdirectory`? Преимущества и недостатки этого метода подключения сторонних библиотек.

   c. `find_package`
   Команда `find_file`. Алгоритм поиска файла в системе при использовании `find_file`. Переменные:

   - `<ИмяПакета>_ROOT`
   - `CMAKE_PREFIX_PATH`
   - `CMAKE_INCLUDE_PATH`
   - `CMAKE_FRAMEWORK_PATH`
   - `CMAKE_SYSTEM_PREFIX_PATH`

   Опции `HINTS` и `PATHS`. Команды `find_library` и `find_program`.
   Команда `find_package` и её опции:

   - `REQUIRED`
   - `COMPONENTS`

   Два режима работы команды `find_package`:

   i. Режим поиска модуля:
   Какой файл ищет CMake в этом режиме? Алгоритм поиска файла. Переменная `CMAKE_MODULE_PATH`.

   ii. Режим поиска конфигурационного файла:
   Какой файл ищет CMake в этом режиме? Алгоритм поиска файла. Переменные:

   - `<ИмяПакета>_ROOT`
   - `CMAKE_PREFIX_PATH`
   - `CMAKE_FRAMEWORK_PATH`
   - `CMAKE_APPBUNDLE_PATH`

   Переменные, создаваемые командой `find_package`:

   - `<ИмяПакета>_FOUND`
   - `<ИмяПакета>_LIBRARIES`

   Импортированные таргеты, создаваемые командой `find_package`.
   Преимущества и недостатки `find_package` для подключения сторонних библиотек.

   d. `ExternalProject`
   Команда `ExternalProject_Add` и её опции:

   - `GIT_REPOSITORY`
   - `GIT_TAG`
   - `DOWNLOAD_COMMAND`
   - `BUILD_COMMAND`
   - `INSTALL_COMMAND`

   Преимущества и недостатки этого метода подключения сторонних библиотек.

   e. `FetchContent`
   Команда `FetchContent_Declare` и её опции:

   - `GIT_REPOSITORY`
   - `GIT_TAG`

   Команда `FetchContent_MakeAvailable`. Чем `FetchContent` отличается от `ExternalProject`? Преимущества и недостатки этого метода подключения сторонних библиотек.

   f. **Пакетный менеджер** `conan`
   Пакетные менеджеры для библиотеки на языках C и C++. Использование пакетного менеджера `conan` совместно с `find_package`. Преимущества и недостатки этого метода подключения сторонних библиотек.

5. **Тестирование**

   a. **Основы тестирования**
   Что такое тесты? Написание тестов без использования сторонних библиотек. Юнит-тестирование.

   b. **CTest**
   Команда `enable_testing`. Команда `add_test` и её опции `NAME`, `COMMAND` и `WORKING_DIRECTORY`. Свойства тестов:

   - `TIMEOUT`
   - `WILL_FAIL`
   - `FAIL_REGULAR_EXPRESSION`
   - `PASS_REGULAR_EXPRESSION`
   - `LABELS`
   - `COST`

   Программа `ctest`. Опции команды `ctest`:

   - Опция `-N`
   - Опции `-R` и `-E`
   - Опция `-j`
   - Опция `-repeat-until-fail`
   - Опция `-timeout`

   c. **Подключение GoogleTest**
   Подключение библиотеки `Google Test` с помощью CMake. Таргеты `gtest` и `gtest_main`.

   d. **GoogleTest**
   Создание простых тестов. Макрос `TEST`. Наборы тестов. `EXPECT_` и `ASSERT_` проверки. В чём отличие между ними? Проверки:

   - `EXPECT_TRUE`, `EXPECT_FALSE`
   - `EXPECT_EQ`, `EXPECT_LE` и т. д.
   - `EXPECT_STREQ`
   - `EXPECT_FLOAT_EQ`, `EXPECT_NEAR`
   - `EXPECT_THROW`, `EXPECT_ANY_THROW`, `EXPECT_NO_THROW`
   - `EXPECT_DEATH`, `EXPECT_EXIT`

   Фикстуры (англ. *fixtures*). Макрос `TEST_F`. Отключение тестов с помощью `DISABLED_`.

   e. **GoogleMock**
   Мок-объекты. Создание мок-классов. Макрос `MOCK_METHOD`. Задание ожиданий. Макрос `EXPECT_CALL`. Методы `Times`, `WillOnce` и `WillRepeatedly`. Кардинальности:

   - `testing::AnyNumber`
   - `testing::AtLeast`
   - `testing::AtMost`
   - `testing::Between`
   - `testing::Exactly`

   Матчеры:

   - `testing::_`
   - `testing::Any<T>`
   - `testing::Eq`, `testing::Gt`, и т. д.
   - `testing::Contains`
   - `testing::Truly`
   - `testing::AllOf`
   - `testing::AnyOf`
   - `testing::ElementsAre`

   Действия:

   - `testing::Return`
   - `testing::ReturnNew`
   - `testing::SetArgReferee`
   - `testing::Throw`
   - `testing::Invoke`
   - `testing::DoAll`