

Модуль "Утилиты". Вопросы.

1. Сборка

а. Раздельная компиляция

Что такое файл исходного кода и исполняемый файл? Этап сборки программы: препроцессинг, ассемблирование, компиляция и линковка. Что такое заголовочные файлы (header-файлы)? Что делает директива препроцессора `#include`? Что такое единица трансляции? Компиляция программы с помощью `g++`. Опции компиляции `-E`, `-S` и `-c`. Что такое раздельная компиляция и в чём её преимущества?

б. Библиотеки

Что такое библиотека? Виды библиотек: header-only библиотеки, open-source библиотеки, статические библиотеки, динамические библиотеки. В чём различия между этими видами библиотек? В чём преимущества и недостатки каждого из видов библиотек? Как подключить библиотеки к своему проекту?

в. Статические библиотеки

Как создать статическую библиотеку? Как подключить статическую библиотеку? Опции компилятора `-I`, `-L` и `-l`. Характерные расширения файлов статических библиотек на Linux и Windows.

г. Динамические библиотеки

В чём главная разница между статическими и динамическими библиотеками? Как создать динамическую библиотеку? Как подключить динамическую библиотеку? Характерные расширения файлов динамических библиотек на Linux и Windows.

д. Опции компилятора `g++`

- Опции для указания стандарта языка, например `-std=c++20`
- Опции для включения/отключения предупреждений: `-Wall`, `-Wextra`, `-Werror`.
- Опция для указания директорий заголовочных файлов, необходимых для компиляции `-I`
- Опция для указания директорий библиотек, необходимых для компиляции `-L`
- Опция для указания названий библиотек, необходимых для компиляции `-l`
- Опция для включения возможности проведения дебага: `-g`
- Опции для включения оптимизаций: `-O0`, `-O1`, `-O2`, `-O3`, `-Os`
- Опция `-DNDEBUG`
- Опция `-D` для задания `#define`-макросов. Как её использовать? Пример использования данной опции.

2. CMake как система сборки

а. Основы CMake

Что такое Cmake и для чего он нужен? Основы работы с CMake. Структура CMake-проекта. Файл `CMakeListis.txt`. Как скомпилировать проект с помощью CMake? Что делают следующие команды CMake:

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

Как собрать проект с использованием CMake? Генерация файлов проекта для данной среды. Выбор генератора. Опции программы `cmake`: `-S`, `-B`, `-G`, `--build`.

б. Таргеты

Что такое таргет (target)? Что делают следующие команды CMake:

- `add_executable`
- `add_library` и её опции `STATIC` и `SHARED`
- `target_link_libraries` (если аргумент является таргетом)

в. Свойства таргетов

Что делают следующие команды CMake:

- `target_include_directories`
- `target_link_directories`
- `target_link_libraries` (если аргумент не является таргетом)

- target_compile_features
- target_compile_definitions
- target_compile_options

d. **Типы зависимостей между таргетами**

Типы связей между двумя таргетами PRIVATE, PUBLIC и INTERFACE. Типы связей между таргетом и его свойством PRIVATE, PUBLIC и INTERFACE. В чём отличия между этими типами зависимостей? Зачем нужно указывать тип для каждой связи? Примеры ситуаций когда нужно использовать ту или иную связь.

e. **Простые переменные CMake**

Простые переменные CMake. Какие бывают типы у переменных языка CMake? Как создать простую переменную в CMake? Команда **set**. Как напечатать значение переменной на экран? Основные стандартные переменные:

- | | |
|----------------------|------------------------------------|
| • CXX_STANDARD | • PROJECT_BUILD_DIR |
| • CMAKE_CXX_COMPILER | • <имя проекта>_SOURCE_DIR |
| • CMAKE_SOURCE_DIR | • <имя проекта>_BUILD_DIR |
| • CMAKE_BUILD_DIR | • BUILD_SHARED_LIBS |
| • PROJECT_SOURCE_DIR | • WIN32, LINUX, APPLE, MSVC, MINGW |

f. **Поддиректории**

Как добавить новую поддиректорию в CMake проект? Команда **add_subdirectory**. Что происходит при выполнении этой команды? Область видимости переменных. Видны ли переменные, созданные в родительской Cmake-директории, в поддиректории? Видны ли переменные, созданные в поддиректории, в родительской Cmake-директории? Опция **PARENT_SCOPE** команды **set**. Переменные:

- CMAKE_CURRENT_SOURCE_DIR
- CMAKE_CURRENT_BUILD_DIR

3. CMake как язык программирования

a. **Переменные**

Какие бывают типы у переменных языка CMake? Как создать простую переменную в CMake? Команда **set**. Как получить значение переменной по её названию?

b. **Условная команда if**

Как пользоваться командой **if** и сопутствующими командами в языке CMake? Какие строки команды **if** воспринимает как истинные, а какие как ложные? Использование переменных как аргументы команды **if**. Логические операторы AND, OR, NOT. Сравнение чисел: EQUAL, LESS, GREATER. Сравнение строк на равенство: STREQUAL. Проверка на то, что существует файл: EXISTS. Проверка, существует ли переменная и данным именем: DEFINED.

c. **Списки**

Что представляет собой список в языке CMake? Как создать список? Как работать со списком? Передача списка в функцию. Команда **list**. Опции этой команды: LENGTH, GET, FIND, APPEND, SORT.

d. **Циклы**

Команда **while**. Команда **foreach**. Опции команды **foreach**: RANGE и IN LISTS. Итерирование по списку с помощью команды **foreach**.

e. **Функции**

Функции в языке CMake. Как создать функцию с помощью команды **function**? Как передавать в функцию? Переменные ARGV, ARGV, ARGV. Как возвращать из функции. Опция **PARENT_SCOPE** команды **set**. Команда **return** с опцией **PROPAGATE**. Области видимости функций. Команда **cmake_parse_arguments**.

f. **Манипуляции со строками**

Команда **string** и её опции:

- | | | |
|-----------|-----------|-------------|
| • FIND | • JOIN | • LENGTH |
| • REPLACE | • TOLOWER | • SUBSTRING |
| • APPEND | • TOUPPER | • COMPARE |

g. Файлы

Команда `file` и её опции:

- | | | |
|------------------|----------|-------------|
| • READ | • REMOVE | • CHMOD |
| • STRINGS | • RENAME | • REAL_PATH |
| • WRITE | • COPY | • DOWNLOAD |
| • MAKE_DIRECTORY | • SIZE | • GLOB |

Является ли хорошей идеей использование команды `file` с опцией `GLOB`, чтобы найти названия всех файлов исходного кода некоторого таргета?

h. Модули

Что представляет собой модуль в языке CMake. Подключение модулей. Команда `include`. В каких папках ищутся модули? Переменная `CMAKE_MODULE_PATH`. Область видимости переменных. Переменные `CMAKE_CURRENT_LIST_DIR` и `CMAKE_CURRENT_LIST_FILE`. Чем команда `include` отличается от команды `add_subdirectory`? Команда `include_guard`.

4. CMake - дополнительные возможности

a. Переменные среды

Что такое переменные среды? Как пользоваться переменными среды из CMake?

b. Кэшированные переменные CMake

Чем кэшированные переменные отличаются от обычных переменных? Создание кэшированных переменных с помощью команды `set`. Поле `type` при создании кэшированной переменной. Как изменить уже созданную ранее кэшированную переменную? Опция `FORCE` команды `set`. Задание кэшированных переменных в командной строке (опция `-D`). Файл `CMakeCache.txt`.

c. Свойства

Что такое свойства в языке CMake? В чём отличие свойств от переменных? Какие объекты могут иметь свойства? Команды `get_property` и `set_property` для получения и изменения свойств различных объектов. Свойства директорий:

- VARIABLES
- CACHE_VARIABLES
- SUBDIRECTORIES
- PARENT_DIRECTORY
- BUILDSYSTEM_TARGETS

Свойства таргетов:

- | | |
|---------------------------------|------------------------------|
| • TYPE | • INTERFACE_COMPILE_OPTIONS |
| • OUTPUT_NAME | • LINK_DIRECTORIES |
| • SOURCES | • LINK_LIBRARIES |
| • INCLUDE_DIRECTORIES | • LINK_OPTIONS |
| • COMPILE_DEFINITIONS | • INTERFACE_LINK_DIRECTORIES |
| • COMPILE_OPTIONS | • INTERFACE_LINK_LIBRARIES |
| • INTERFACE_INCLUDE_DIRECTORIES | • INTERFACE_LINK_OPTIONS |
| • INTERFACE_COMPILE_DEFINITIONS | |

d. Тип сборки

Что такое тип сборки (также известный как *тип конфигурации* или просто *конфиг*)?

Типы сборки по умолчанию:

- Release
- Debug
- RelWithDebInfo
- MinSizeRel

Чем отличаются эти типы сборки? Какие опции компилятора использует каждый из этих типов сборки? Одноконфигурационные (*single-config*) и мультиконфигурационные (*multi-config*) генераторы. Как установить тип сборки при использовании одноконфигурационного генератора? Переменная

СMAKE_BUILD_TYPE. Как установить тип сборки при использовании мультikonфигурационного генератора? Переменные CMAKE_CONFIGURATION_TYPES и GENERATOR_IS_MULTI_CONFIG. Как узнать тип сборки при использовании мультikonфигурационного генератора?

e. **Генераторные выражения**

Стадия конфигурации и стадия генерации. Что такое генераторные выражения (generator expressions)? Когда их нужно использовать? Приведите пример команд, которые поддерживают генераторные выражения и команд, которые их не поддерживают. Синтаксис генераторных выражений. Что делают следующие генераторные выражения:

0, 1, BOOL, AND, OR, NOT, IF, STREQUAL, CONFIG, TARGET_PROPERTY, TARGET_FILE

5. **Git - локальный репозиторий**

a. **Основы**

Что такое система контроля версий? Централизованные и распределённые системы контроля версий. Снимок состояния. Репозиторий. Локальный и удалённый репозитории.

b. **Настройка git**

Команда `git config` и её использование для настройки git. Что делают следующие команды:

```
git config user.name "Ivan Ivanov"
git config user.email ivan.ivanov@mail.ru
git config core.editor vim
git config core.autocrlf true
```

Опции команды `git config`: `--local`, `--global` и `--list`.

c. **Создание нового репозитория**

Создание нового пустого репозитория с помощью команды `git init`. Клонирование существующего репозитория с помощью команды `git clone`.

d. **Добавление новых коммитов**

Области хранения файлов в системе git:

- рабочая папка
- индекс
- локальный репозиторий
- удалённый репозиторий

Типы файлов в системе git:

- неотслеживаемые
- игнорируемые
- индексированные
- изменённые
- зафиксированные (закомиченные)

Как создать неотслеживаемый файл? Добавление неотслеживаемых файлов в индекс с помощью команды `git add`. Как изменить/удалить файлы, находящиеся в индексе? Добавление индексированных файлов в локальный репозиторий с помощью команды `git commit`. Что такое коммит? Опции команды `git commit`: `-m` и `-a`. Игнорируемые файлы. Файл `.gitignore`.

e. **Просмотр информации**

Просмотр состояния рабочей папки и индекса с помощью команды `git status`. Команда `git diff` для просмотра разницы в файлах рабочей папки и файлах индекса. Команда `git diff --staged` для просмотра разницы в файлах индекса и файлах текущего коммита репозитория. Просмотр истории коммитов с помощью команды `git log`. Опции команды `git log`: `--oneline`, `--graph`, `--since`, `-S`. Хеш коммита. Сокращённый хеш коммита. Команда `git diff` для просмотра разницы в файлах двух коммитов.

f. **Очистка и отмена**

Удаление всех неотслеживаемых файлов с помощью `git clean -fd`. Удаление всех игнорируемых файлов с помощью команды `git clean -fdX`. Отмена изменений в рабочей директории с помощью `git restore`. Отмена изменений в индексе с помощью `git restore --staged`. Правка последнего коммита с помощью команды `git commit --amend`. Отмена зафиксированных изменений, с помощью `git reset`.

Чем отличаются следующие команды:

- `git reset --hard`
- `git reset --mixed`
- `git reset --soft`

Можно ли восстановить данные, если вы случайно сделали `git reset --hard`? Как полностью удалить коммит из репозитория?

g. **Ветки**

Что такое ветки в системе git? Создание веток с помощью команды `git branch`. Удаление веток с помощью команды `git branch -d`. Переименование веток командой `git branch -m`. Перенос ветки на другой коммит с помощью команды `git branch -f`. Переход на другую ветку с помощью команды `git switch`. Создание новой ветки с переходом на неё с помощью `git switch -c`. Указатель HEAD. Переход на произвольный коммит с помощью `git switch`. Состояние "отделённой головы" (detached HEAD). Как выйти из состояния отделённой головы? Использование символов `~` и `^` для навигации по коммитам репозитория.

h. **Слияние веток**

Слияние веток с помощью команды `git merge`. Слияние веток с помощью "перемотки" (fast-forwarding). Конфликты слияния. Разрешение конфликтов слияния и команда `git merge --continue`. Отмена конфликтного слияния, команда `git merge --abort`. Как отменить уже сделанное слияние? Опции `--no-commit` и `--no-ff` команды `git merge`.

i. **Копирование коммитов**

Копирование коммитов с помощью `git cherry-pick`. Конфликты при копировании коммитов. Разрешение конфликтов при копировании и команда `git cherry-pick --continue`. Отмена конфликтного копирования, команда `git cherry-pick --abort`.

j. **Перебазирование веток**

Перебазирование веток с помощью `git rebase`. Отличие слияния от перебазирования. Конфликты при перебазировании. Разрешение конфликтов при перебазировании и команда `git rebase --continue`. Отмена конфликтного перебазирования, команда `git rebase --abort`. Интерактивное перебазирование. Действия при интерактивном перебазировании: `pick`, `reword`, `edit`, `squash`, `drop`.

6. **Git - удалённые репозитории**

a. **Удалённый репозиторий**

b. **GitHub**

7. **CMake - подключение сторонних библиотек**

a. **Поиски**

b. **FetchContent**

c. **git submodule**

8. **Тестирование**

a. **Юнит-тестирование**

b. **GoogleTest**

c. **CTest**