Санкт-Петербургский государственный университет

# План на II семестр

1. Сборка многопроектного проекта (cmake). Модульное тестирование
2. Шаблоны функций/классов. Практика классов (длинные числа, вектор, список, дерево)
3. Наследование. Полиморфизм
4. Обработка исключений. Работа с файлами
5. JSON. Создание многоязычного интерфейса

Санкт-Петербургский
государственный
университет

# Многопроектный проект, CMake, GoogleTest

Санкт-Петербургский государственный университет

# Современное программирование — это

1. Хорошие математические навыки
2. Язык программирования
3. Инструменты разработки (сборка, тестирование, автоматизация, СКВ и т.д.)
4. Общий кругозор программирования (парадигмы п-я, устройство памяти, сети и т.д.)
5. Умение писать код (стили написания кода, рефакторинг и т.д.)

Санкт-Петербургский государственный университет

# Cmake

**Cmake** — кроссплатформенная автоматизированная система сборки проектов.

https://cmake.org

Разработана Kitware по заказу национальной библиотеки медицины США.

Первый выпуск 2000 г.

Cmake является свободным и открытым ПО. https://github.com/Kitware/CMake

**CMake не занимается непосредственно сборкой**, а лишь генерирует файлы сборки из предварительно написанного файла сценария CmakeLists.txt. Cmake считается альтернативой распространённой в сообществе GNU системе Autotools. CMake может создавать файлы проектов для многих сред разработки.

Сборка программы или библиотеки с помощью CMake представляет собой двухэтапный процесс. На первом этапе стандартные файлы сборки генерируются из файлов конфигурации CMakeLists.txt, которые написаны на языке CMake. Затем задействуются системные инструменты сборки, такие как make, ninja, используемые для непосредственной компиляции программ.

После сборки каждый проект в подкаталогах содержит CMakeCache.txt и каталог make-файлов, что помогает избежать или ускорить этап «перегенерации» в случае повторного запуска сборки.

Санкт-Петербургский государственный университет

# Сборка проекта

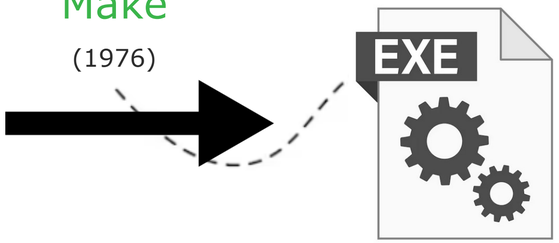
Раньше

+

Makefile

Make

(1976)

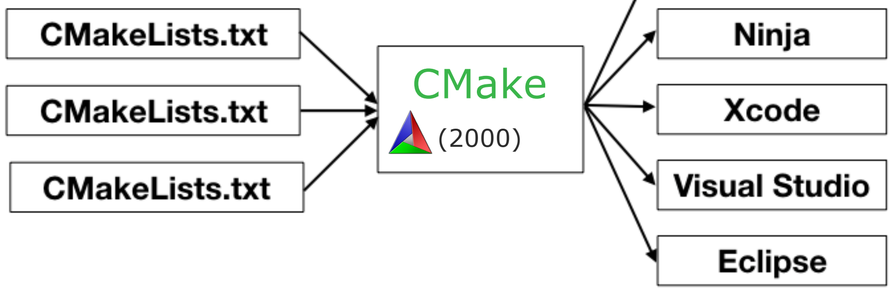







Сейчас

+





Ninja

(2012)

Санкт-Петербургский государственный университет

# Ninja

**Ninja** — кроссплатформенная консольная утилита, представляющая из себя систему сборки программного обеспечения из исходного кода.

https://ninja-build.org/

Ninja разработана Эваном Мартином, сотрудником компании Google.

Первый выпуск 8 мая 2012 г.

Cmake является свободным и открытым ПО. https://github.com/ninja-build/ninja

**Ninja представляет собой улучшенную и доработанную версию утилиты Make.**

Команда запуска cmake -G Ninja ..

# Batch

**Batch — пакет**

**Пакетные файлы** (**.bat**), или батники, — текстовый файл в MS-DOS, OS/2 или Windows, содержащий последовательность команд, предназначенных для исполнения командным интерпретатором (**cmd.exe**).

https://www.tutorialspoint.com/batch_script/index.htm

Пакетный файл — аналог скриптовых файлов командной строки (shell script) в Unix-подобных операционных системах. Основная задача bat-файлов — автоматизация  рутинных операций: копирование, перемещение, переименование, удаление файлов; работа с папками; архивация и т. п. Для написания сценариев используется скриптовый язык **batch script**.

# Cmake

**CMakeLists.txt** — скрипт (рецептом, сценарием) сборки проекта. С точки зрения IDE, таких как CLion или Visual Studio, файл CMakeLists.txt также служит проектом.

**cmake_minimum_required**(VERSION 3.2...<current version>)

**project**(<name>) — это набор компонентов, по смыслу похожий на solution и project в Visual Studio. Устанавливает имя в переменную ${PROJECT_NAME}

**add_executable**(<name> ${SOURCES} ${HEADERS}) — создаёт исполняемый файл из файлов .hpp (.h) и .cpp (.c)

В cmake есть переменные-макросы, и в процессе интерпретации файла(-ов) CMakeLists.txt cmake вычисляет ряд встроенных переменных для каждой цели, тем самым получая флаги. Затем cmake создаёт вторичный скрипт сборки (build.ninja), который будет напрямую вызывать компилятор, компоновщик и другие утилиты с вычисленными флагами.

**add_subdirectory**(<подкаталог>) — подключает скрипт CmakeLists.txt по пути <подкаталог>

**add_library**(<имя> [STATIC|SHARED|INTERFACE] <список исходников...>)

**set**(HEADERS

..... gomove.hpp

..... text.hpp

) — задаёт список файлов, соответствующих переменной HEADERS

PROJECT_SOURCE_DIR — макрос, указывающий на ближайшую директорию, в которой вызвана функция project(<name>)

# Cmake

**target_include_directories**(myapp [PUBLIC | PRIVATE | INTERFACE] includes) — показывает, где искать хедеры для указанной «цели». Это повзоляет в .cpp файлах писать просто названия хедеров без относительных путей.

PUBLIC делает настройку видимой для текущей цели и для всех зависящих от неё целей.

PRIVATE делает настройку видимой только для текущей цели.

INTERFACE делает настройку видимой только для всех зависящих от неё целей.

**enable_testing**() — включает тестирование, и команду add_test().

**add_test**(<name> <command>) — включает тест <name> в список тестов и его теперь будет видеть утилита ctest

<command> — имя исполняемой цели, являющейся тестом.

**target_link_libraries**(test_bubble

..... gtest_main

..... gmock_main

..... bubble_sort_lib

) — подключает для таргета test_bubble библиотеки gtest_main, gmock_main, bubble_sort_lib

**target_sources**(bubble_sort_lib

..... PRIVATE

.....  bubble_sort.cpp

..... PUBLIC

.....  bubble_sort.hpp

) — добавляет к таргету bubble_sort_lib исходники с областями их видимости.

# Batch

**@echo off** — Выключает вывод на экран запускаемых команд.

**setlocal EnableDelayedExpansion** — команда изменения переменных среды.

При использовании переменных окружения в командных файлах существует определенное ограничение, связанное с тем фактом, что присваиваемое значение остается без изменения при его модификации внутри группы команд (if или for), задаваемой скобками, т.к. код внутри ( ) выполняется в подпрограмме.

EnableDelayedExpansion для переменной позволяет изменить переменную вызовом команды set и использование !! (!var_name!) возвращает последнее присвоенное значение.

**Макросы** — это именованный код.

set NAME=kmms_prog_2023

Переменные вне скобок ( ) являются макросами. Значение берётся следующим образом %NAME%.

**rmdir** — удаляет каталог.

rmdir /S /Q %BUILD_FOLDER%

/S указывает, что надо удалить и подкаталоги.

/Q удаление произойдёт без подтверждения пользователем.

**Входные аргументы запуска**

Чтобы получить значение ходного аргумента запуска скрипта надо использовать % и номер аргумента: %1

**cmake -G %1 ..\%NAME%**

Команда создаёт проект типа, указанного во входном аргументе под номером 1 в каталоге ..\%NAME%.

# Batch

**cmake --build .**

Команда запускает компиляцию и сборку созданного проекта.

**if %2=="ninja" ( )**

Проверяет на равенство с «ninja» второй входной аргумент, а потом исполняет код в ().

**set arr[0]=value**

Задаёт значение элементу массива. Для разных типов значений используются разные ключи. В данном случае value — это строка.

**set arr[0].file=file**

**set arr[0].folder=folder**

Задание значения элементу массива, который является объектом.

**for /L %%i in (0,1,5) do (**

**…….. copy ..\%NAME%\!arr[%%i].folder!\!arr[%%i].file! .\!arr[%%i].folder!**

**)**

Итерация по массиву с помощью цикла for.

/L используется для перемещения по диапазонам в for для перебора массива.

%%i — переменная индекс цикла

**copy** — Копирует «что» «куда».

Санкт-Петербургский государственный университет

# Тестирование

**1. Модульное (Unit) тестирование** — это тестирование на корректность отдельных модулей (hpp + cpp) / классов исходного кода без взаимодействия/зависимости с другими модулями/классами.

**2. Функциональное тестирование** — это тестирование отдельных функциональностей в рамках одной независимой части системы. Пример: аутентификация пользователя, запрос в бд, отправка/обработка команды на сервер.

**3. Интеграционное тестирование** — это тестирование совместной работы нескольких частей системы (программных модулей, микросервисов, приложений распределённой системы). Пример: сценарий когда пользователь добавляет товар в корзину, оформляет его и оплачивает.

Санкт-Петербургский государственный университет

# Модульное тестирование

**Важны для разработчика** т.к.:

**1.** Уменьшают время на все остальные виды тестрования.

**2.** Защищают от регрессии.

**Регрессия** — это появление ошибок в уже когда-то проверенном коде.

> Если появился баг, то сначала пишем тест, который проявляет этот баг, а потом его исправляем.

**3.** Подтверждение исполнения документации.

Всё, что есть в тестах, работает. А остальное неизвестно. Ещё можно сказать, что тест — это возможность посмотреть как обращаться с кодом.

**4.** Менее связный код.

Чем больше % покрытия кода тестами, тем менее код системы менее связный.

> 100% покрытие кода тестами — это не про отсутствие ошибок, а про взаимозаменяемость объектов.

Санкт-Петербургский государственный университет

# Модульное тестирование

**Критерии хорошего модульного теста**:

**1. Быстрый**.

Модульных тестов тысячи и их задача дать разработчику возможность быстрой "поверхностной" проверки любого своего изменения. Таймаутов быть не должно.

**2. Изолированный**.

Т.е., результат одного теста не должен влиять на результат другого теста.

— Как проверить?

— Поменять порядок тестов.

Изолированности могут помешать: глобальные, статические переменные; не восстановление состояния системы после работы с бд.

— А как это проверить?

— Если тесты можно запускать параллельно, то их надо запускать параллельно.

**3. Повторяемый**.

Не должно возникнуть ситуации, когда 1 раз на 100 запусков тест будет падать.

**4. Самопроверяемый**.

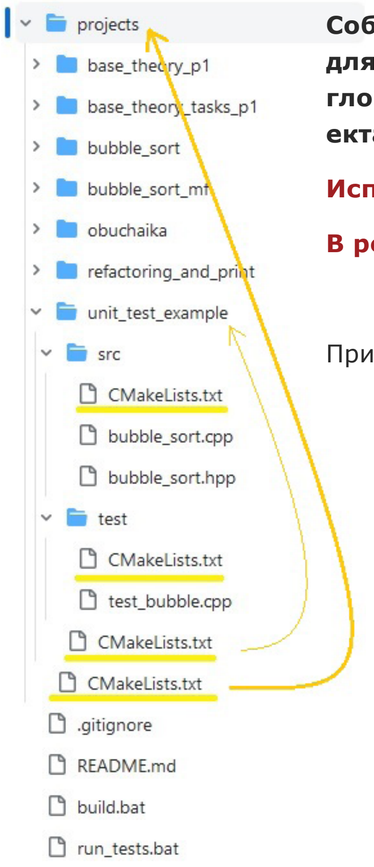
В тесте для запуска не надо ничего настраивать (примеры таких трудных тестов — это на графику, аудио)

**5. Уместный**.

Тест должен проверять то, что необходимо.

Санкт-Петербургский государственный университет

# Домашнее задание




**Собрать с помощью CMake нужные проекты I семестра в один глобальный проект. Написать пробный тест с помощью библиотеки GoogleTest для одного из проектов. Всё это залить на гитхаб, где глобальный проект будет репозиторием. Каждый шаг (создание подпроекта) по созданию глобального проекта должен быть оформлен в коммит с понятным описанием. В дальнейшем все учебные маленькие проекты станут его подпроектами.**

**Использовать IDE нельзя! (Блокнот животворящий — это главное оружие разработчика!)**

**В репозитории не должны попасть файлы сборки (build) и файлы используемых внешних библиотек (lib)!**

Пример построения такого глобального проекта: https://github.com/BuskoIlya/kmms_prog_2023

Санкт-Петербургский государственный университет

# Домашнее задание

**1. Проверить наличие и версии gcc, cmake, ninja**

gcc --version

> MinGW-W64

cmake --version

ninja --version

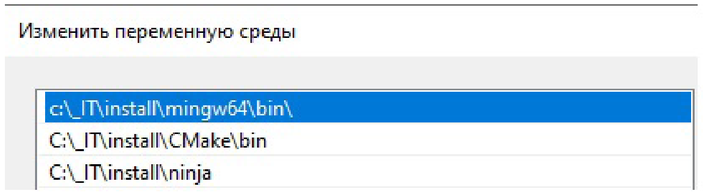
Если каких-либо этих программ нет, то их надо установить:

компилятор https://winlibs.com/ mingw-64

cmake https://cmake.org/download/

ninja https://github.com/ninja-build/ninja/releases

Добавить пути к этим приложениям в переменную среды PATH:




**2. Создать на github репозиторий для всех проектов kmms_prog_2023 и сделать его git clone**

Склонированный репозиторий станет глобальным проектом с подпроектами учебных задач. Структура репозитория:

build_<project_type_name[ninja | visual_studio | clion | ...]> — создаются cmake-ом

projects — каталог, в котором хранится код учебных проектов

.gitignore, build.bat, README, run_tests.bat

Санкт-Петербургский государственный университет

# Домашнее задание

**3. Первый проект и первый коммит (bubble_sort)**

Код проектов и билд проектов должны лежать в разных каталогах. Билды не должны отслеживаться git и отправляться на github.

buil.bat — это batch скрипт, который содержит команды для сборки всех проектов согласно CMakeLists.txt

**4. bubble_sort_mf**

Версия bubble sort с вынесенным кодом по разным файлам. Внимание требует написание CMakeLists.txt, когда часть кода проекта находится в других каталогах (см. CMakeLists.txt в каталоге sortings)

**5. Список добавленный учебных проектов I семестра:**

refactoring_and_print — первый пример рефакторинга

base_theory_p1 — теория в примерах кода перед 1-ым зачётом

base_theory_tasks_p1 — задачи на поиск ошибки в коде на первом зачёте

obuchaika

Санкт-Петербургский государственный университет

# Домашнее задание

**6. Добавление тестов, написанных с помощью библиотеки googletest и запускаемых командой ctest**

Создать в projects каталог lib и сделать в нём git clone https://github.com/google/googletest.git. При этом каталог lib надо исключить в .gitignore. В репозитории должен лежать только код и скрипты его касающиеся.

Создать отдельный проект с кодом и тестом для него unit_test_example. Обратить внимание как подключается код к тестам и как тесты собираются. Тесты всегда находятся в отдельном по отношению к коду каталоге test и собираются как отдельный таргет.

Санкт-Петербургский
государственн
университет