

Программирование на C++

МИФИ, 2016

Роман Кузнецов

Добро пожаловать в Техноатом!

Mail.Ru Group совместно с НИЯУ МИФИ запустили проект в 2016 году.

Другие проекты:

- «Технопарк» - Mail.Ru Group и МГТУ им. Н.Э. Баумана;
- «Техносфера» - Mail.Ru Group и факультет ВМК МГУ имени М.В. Ломоносова;
- «Технотрек» - Mail.Ru Group и МФТИ.

**Просьба отметить на
портале!**

План лекции

- 1) Знакомство;
- 2) Представление программы курса;
- 3) Обсуждение курсового проекта;
- 4) Git и GitHub;
- 5) Проектирование объектно-ориентированных приложений;
- 6) Unit-тесты при помощи Google Test;
- 7) Кросс-платформенная разработка и CMake.

Обо мне

Меня зовут Роман Кузнецов.

- Родился в Новосибирске;
- Учился в НГТУ, в 2013 получил степень к.т.н.
- Ведущий разработчик в проекте MAPS.ME;

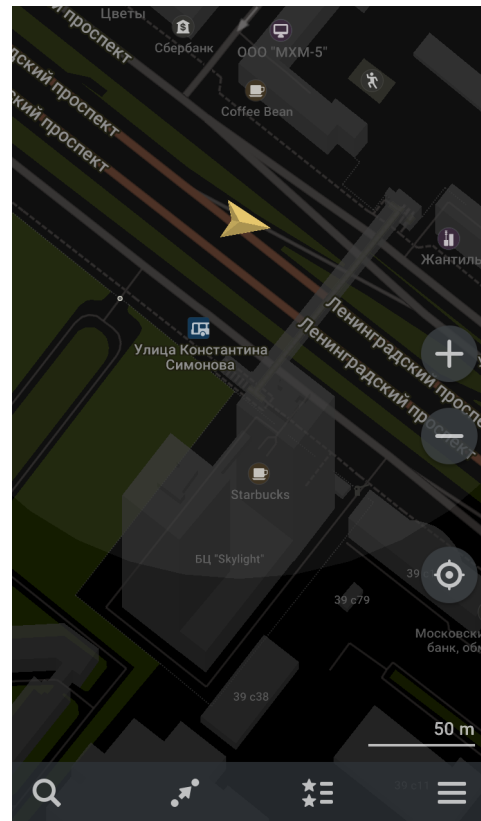
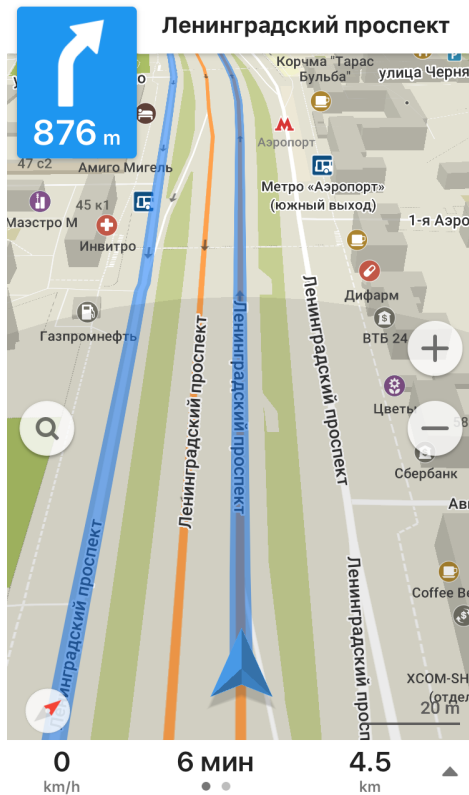
MAPS.ME

Оффлайн-карты на основе
Open Street Map для мобильных
платформ (iOS, Android).

Команда: 20 человек (расширяется)

Разработчики: 12 человек

300 000 строк кода (240 000 на C++)!



- 9 лет опыт разработки на C++;
- Основная специализация — рендеринг.



Контакты

E-mail: r.kuznetsov@mapswithme.com

GitHub: <https://github.com/rokuz>

Портал: <https://atom.mail.ru>

Наш курс

- 14 занятий (~1 семестр).
- 1 занятие в неделю.
- Занятие длится 4 ак.ч.
- Предлагаю разбить занятие на 2 части по ~2 ак.ч. с небольшим перерывом!
- Каждое занятие проводится в формате диалога.

Лекция 1 Введение в курс	Лекция 2 Классы в C++	Лекция 3 Указатели	Лекция 4 Иерархии классов и приведение типов
Лекция 5 Основные коллекции данных и обработка ошибок	Лекция 6 Шаблоны	Контрольное занятие	Лекция 7 Паттерны проектирования и их реализация на C++
Лекция 8 Введение в программирование UI при помощи Qt 5	Лекция 9 Введение в визуализацию при помощи OpenGL	Лекция 10 Работа с файлами	Лекция 11 Многопоточность
Специальные темы	Заключительное занятие Сдача проектов.		

Подробная программа курса

<https://atom.mail.ru/curriculum/program/discipline/1/>



Баллы

Максимум 100 баллов (будет отмечено в сертификате!).

- Курсовой проект — **50-70** баллов.
- Тест на контрольном занятии — **0-30** баллов.
- Домашние задания — **0-20** баллов.

Баллы

Итого:

«отлично» — **90-100** баллов;

«хорошо» — **70-90** баллов;

«удовлетворительно» — **50-70** баллов.

Домашние задания

- Всего 10 оцениваемых заданий (после лекций 2 - 11);
- 2 балла за каждое за сдачу в срок;
- 1 балл - за сдачу после срока;
- Срок сдачи будет объявляться в конце каждой лекции.

Рекомендуемая литература



Бьярне Страуструп
Программирование. Принципы и практика с использованием C++ (второе издание).

Курсовой проект

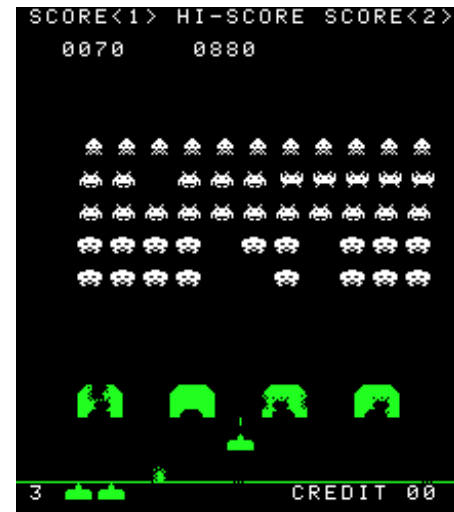
- Это будет игра.
- Если у вас есть крутой собственный проект на C++ — можно обсудить.
- Выполняем в группе из 2 человек (но не больше 2).
- Должен содержать материал лекций 1-7, как минимум.

Игра - клон «Space Invaders»

Должны быть:

- Пушка;
- Пришельцы;
- Препятствия.

Все остальное — на ваш вкус!

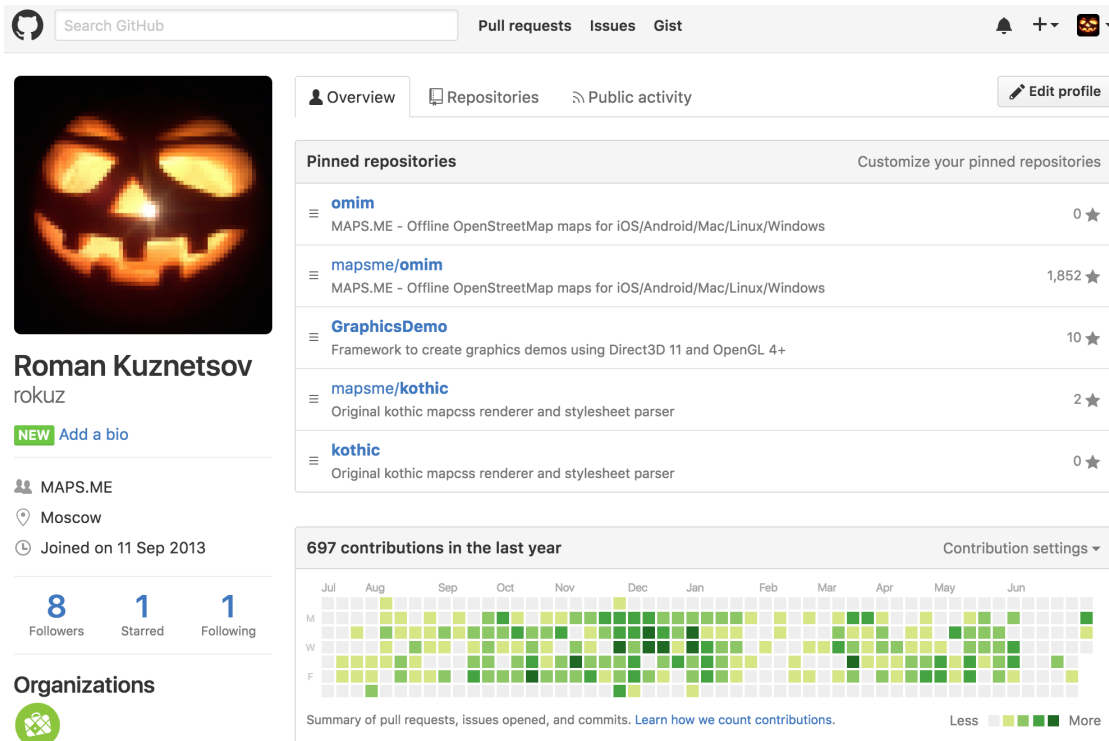


Формат выполнения курсового проекта

- Работаем на GitHub!
- Вы создаете там свой проект, я подписываюсь на изменения.
- Все изменения кода вы оформляете в виде Pull Request (PR).
- Я выполняю code-review ваших PR.
- Вы тоже можете выполнять code-review PR друг друга!

GitHub

- Хостинг git-репозиториев;
- Бесплатно для open-source;
- Удобный интерфейс для code-review.



The screenshot shows the GitHub profile of Roman Kuznetsov (rokuZ). The profile includes a profile picture of a jack-o'-lantern, the name "Roman Kuznetsov", and the username "rokuZ". It also shows a bio, location (Moscow), and join date (11 Sep 2013). The profile has 8 followers, 1 starred repository, and 1 following. The "Pinned repositories" section lists four repositories: "omim", "mapsme/omim", "GraphicsDemo", and "mapsme/kothic". The "697 contributions in the last year" section shows a heatmap of contributions from July to June, with a legend indicating the number of contributions per day (Less, More).

Search GitHub Pull requests Issues Gist

Overview Repositories Public activity Edit profile

Pinned repositories Customize your pinned repositories

- omim** 0 ★
MAPS.ME - Offline OpenStreetMap maps for iOS/Android/Mac/Linux/Windows
- mapsme/omim** 1,852 ★
MAPS.ME - Offline OpenStreetMap maps for iOS/Android/Mac/Linux/Windows
- GraphicsDemo** 10 ★
Framework to create graphics demos using Direct3D 11 and OpenGL 4+
- mapsme/kothic** 2 ★
Original kothic mapcss renderer and stylesheet parser
- kothic** 0 ★
Original kothic mapcss renderer and stylesheet parser

697 contributions in the last year Contribution settings

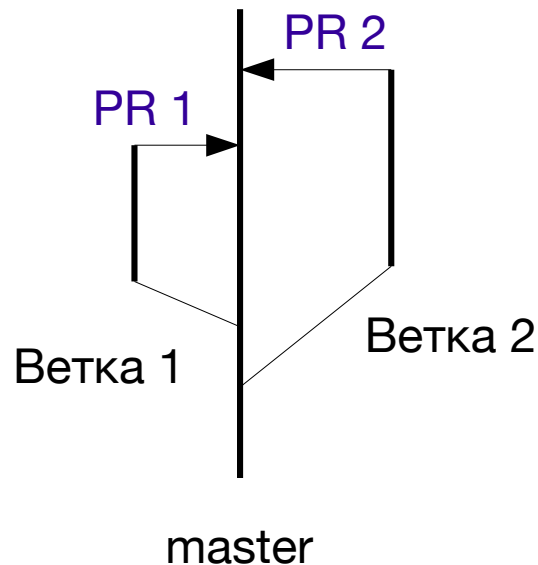
Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May Jun

M W F

Summary of pull requests, issues opened, and commits. [Learn how we count contributions.](#) Less More

Git

- Система контроля версий;
- Создает локальный репозиторий;
- Полнофункциональна в командной строке;
- Удобна для feature branching.



Команды Git

git clone URL

Создает локальную копию удаленного репозитория в текущей папке.

Например,

git clone <https://github.com/mapsme/omim>.git

Команды Git

git submodule init

git submodule update

Инициализация и загрузка в локальный репозиторий всех подмодулей.

Команды Git

git checkout <имя ветки>

Переключается на заданную ветку.

Например,

git checkout master

Команды Git

git checkout -b <имя ветки>

Создает новую ветку и переключается на нее.

Например,

git checkout -b new-feature

Команды Git

git add <файлы через пробел>

Помечает файлы, изменения в которых вы хотите сохранить (индексация).

Например,

git add main.cpp — помечает файл main.cpp.

git add --all — помечает все файлы.

Команды Git

`git commit -m "Message"`

Сохраняет помеченные файлы с изменениями.

Например,

`git commit -m "Added new feature"`

Команды Git

git push [origin <имя ветки>]

Отправляет изменения из локального репозитория в удаленный.

Например,

git push — отправляет изменения из текущей ветки в соответствующую ей в удаленном репозитории.

git push origin my-branch — можно указать конкретную ветку.

Команды Git

git pull

Забирает изменения из ветки удаленного репозитория в текущую локальную ветку.

git status

Выводит информацию о состоянии локального репозитория.

Схема работы с Git

`git checkout master`

`git pull`

`git checkout -b new-feature`

Пишем код

`git add --all`

`git commit -m "Added new feature"`

`git push origin new-feature`

`git checkout master`

`git pull`

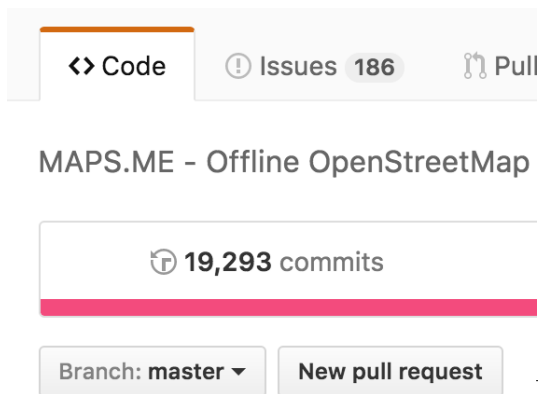
`git checkout new-feature`

`git rebase master`

Исправляем конфликты, если есть

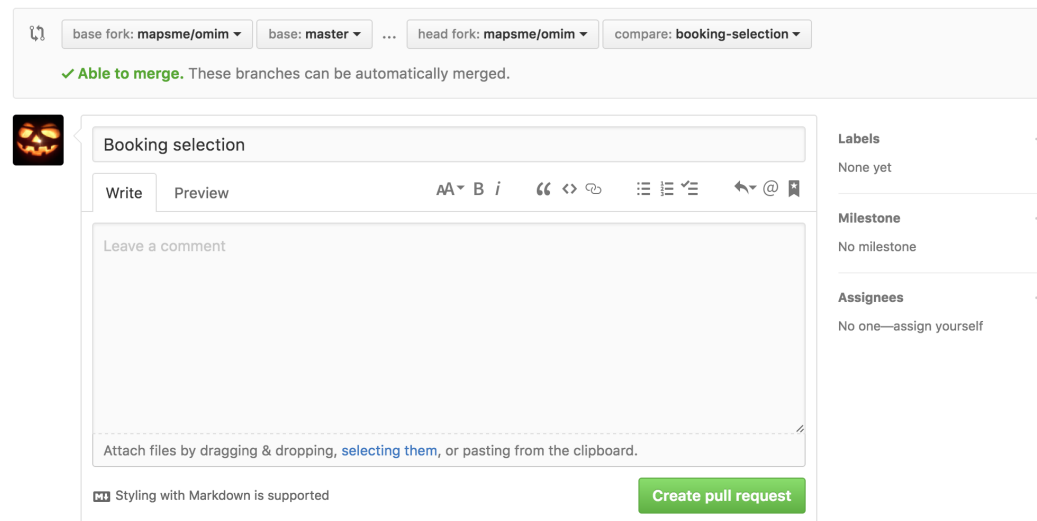
`git rebase --continue`

Переходим к GitHub!



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



base fork: mapsme/omim base: master ... head fork: mapsme/omim compare: booking-selection

✓ Able to merge. These branches can be automatically merged.

Booking selection

Write Preview AA B i “ < > ↺ ⋮ ⋮ ⋮ ↶ @

Leave a comment

Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.

Styling with Markdown is supported

Create pull request

Labels: None yet

Milestone: No milestone

Assignees: No one—assign yourself

Рекомендую почитать про работу с Git!

<https://ru.atlassian.com/git/tutorials/>

<https://try.github.io/>

<https://habrahabr.ru/post/125799/>

Проектирование в объектно-ориентированном стиле!

Задача: необходимо реализовать хранение данных о размере прямоугольника, реализовать возможность вычисления его периметра и площади.

Функциональный подход

```
float CalcRectangleSquare(float a, float b)
{
    return a * b;
}
```

```
float CalcRectanglePerimeter(float a, float b)
{
    return (a + b) * 2.0f;
}
```

// Стороны прямоугольника

```
float a = 3.0f;
```

```
float b = 5.0f;
```

// Площадь

```
float s = CalcRectangleSquare(a, b);
```

// Периметр

```
float p = CalcRectanglePerimeter(a, b);
```

Объектно-ориентированный подход

```
class Rectangle
{
public:
    Rectangle() = default;
    Rectangle(float width, float height) : m_width(width), m_height(height) {}

    float Square() const { return m_width * m_height; }
    float Perimeter() const { return (m_width + m_height) * 2.0f; }

private:
    float m_width = 0.0f;
    float m_height = 0.0f;
};
```

Объектно-ориентированное программирование (ООП)

Основная единица программирования - **класс**.

Классы являются программными аналогами понятий и предметов реального мира.

Класс – это данные и методы для работы с ними, объединенные в целостную группу (Класс = Данные + Методы).



Лозунг ООП: «**Всё есть объект**».

Декомпозиция проекта

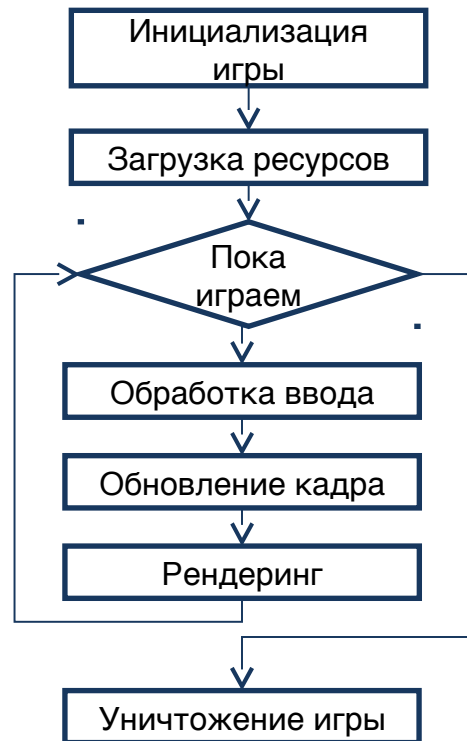
- разделение проекта на части.

Полезные приемы:

- 1) Выделение систем и подсистем;
- 2) Выделение бизнес-логики (логики предметной области);
- 3) Индукция (от частного к общему);
- 4) Определение зависимостей.

Организация игры

- 1) Инициализация игры (создание окна, инициализация платформенных систем, инициализация бизнес-логики);
- 2) Загрузка ресурсов (текстуры, модели, звуки и т.д.);
- 3) Обработка ввода (опрос устройств ввода);
- 4) Обновление кадра (бизнес-логика игры);
- 5) Рендеринг (отображение игры на экране);
- 6) Уничтожение игры (выгрузка ресурсов, освобождение занимаемой памяти).



Выделение систем и подсистем

GameController

Бизнес-логика игры

InputController

Система ввода

GraphicsEngine

Графический движок

Выделение систем и подсистем

GameController

Бизнес-логика игры

EntityFactory

Игровые объекты

AI

Искусственный интеллект

InputController

Система ввода

ActionManager

Игровые действия

GraphicsEngine

Графический движок

ResourceManager

Графические ресурсы

Renderer

Рисование графических представлений игровых объектов

Выделение бизнес-логики

Gun

Пушка

Действия:

- стрелять;
- перемещаться влево/вправо;

Свойства:

- положение;
- габариты;
- скорострельность (пуль/сек);
- здоровье.

Выделение бизнес-логики

Gun

Пушка

Действия:

- стрелять;
- перемещаться влево/вправо.

Свойства:

- положение;
- габариты;
- скорострельность (пуль/сек);
- здоровье.

Bullet

Пуля

Действия:

- лететь.

Свойства:

- положение;
- направление;
- габариты;
- скорость (пикс/сек);
- урон.

Выделение бизнес-логики

Gun

Пушка

Действия:

- стрелять;
- перемещаться влево/вправо.

Свойства:

- положение;
- габариты;
- скорострельность (пуль/сек);
- здоровье.

Bullet

Пуля

Действия:

- лететь.

Свойства:

- положение;
- направление;
- габариты;
- скорость (пикс/сек);
- урон.

Alien

Пришелец

Действия:

- стрелять;
- перемещаться вниз;

Свойства:

- положение;
- габариты;
- скорость;
- скорострельность (пуль/сек);
- здоровье.

Индукция

- ищем общие черты и обобщаем.



Определение зависимостей

GameEntity

Игровая сущность

Свойства:

- положение;
- габариты.

Vector2D

2D-вектор

Методы:

- сложение;
- вычитание;
- умножение на число;
- нормализация;
- скалярное произведение.

Поля:

- x, y;

BoundingBox

Габаритный прямоугольник.

Методы:

- получить центр;
- сместить;
- проверить на пересечение с другим.

Поля:

- min (левый нижний угол);
- max (правый верхний угол).

Определение зависимостей

BoundingBox

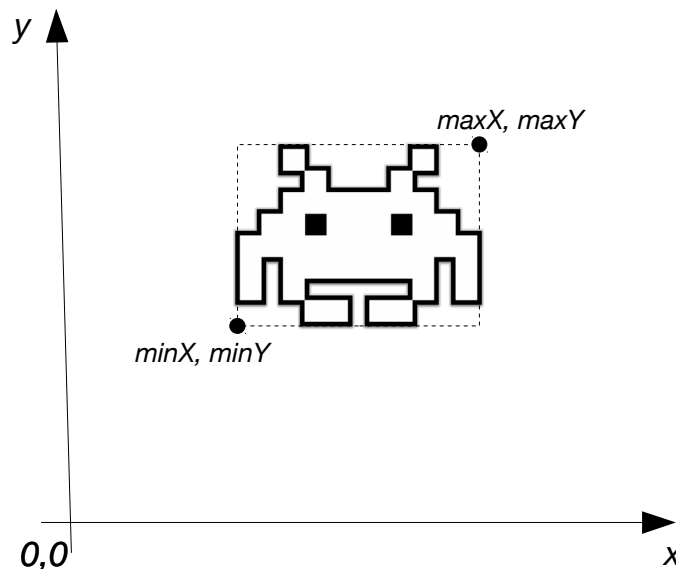
Габаритный прямоугольник.

Методы:

- получить центр;
- сместить;
- проверить на пересечение с другим.

Поля:

- min (левый нижний угол);
- max (правый верхний угол).



Codestyle

- набор правил и соглашений, используемых при написании исходного кода.

Зачем он нужен?

- 1) Улучшается читаемость кода всеми участниками команды;
- 2) Легче находить баги при code review;
- 3) Упрощается поддержка кода.

Codestyle курсового проекта

- 1) Используем **пробелы**, а не табуляцию!
- 2) Отступ — **2 пробела**.
- 3) namespace не добавляет отступа.
- 4) Имена классов и методов — **CamelCase** (начинаются с большой буквы).
- 5) Имена переменных — **camelCase** (начинаются с маленькой буквы).
- 6) Имена переменных-членов класса предваряются префиксом **m_**.
- 7) Фигурная скобочка **всегда с новой строки** (за исключением однострочных методов).
- 8) **const** пишется после типа.
- 9) В классе сначала идет секция **public**, затем **protected**, затем **private**.
- 10) **Транслитерация запрещена!**

```
#pragma once

namespace my
{

class Dummy
{
public:
    Dummy() = default;

    Dummy(int const potatoes, int const tomatoes)
        : m_potatoes(potatoes)
        , m_tomatoes(tomatoes)
    {}

    int GetPotatoes() const { return m_potatoes; }
    int GetTomatoes() const { return m_tomatoes; }

    int Sum() const;

private:
    int m_potatoes = 0;
    int m_tomatoes = 0;
};

} // namespace my
```


Кроссплатформенная разработка

- написание исходного кода, способного с минимальными изменениями быть собранным и исполненным на различных платформах.

Зачем это нужно?

- 1) Единая кодовая база;
- 2) Самая широкая аудитория пользователей;
- 3) Ведет к стандартизации процесса разработки ПО.

Кроссплатформенная разработка

Трудности:

- 1) Существенные различия в платформах;
- 2) Проприетарные технологии (например, Apple Metal);
- 3) Сторонние библиотеки должны поддерживать все ваши целевые платформы;
- 4) Отказ от единой IDE;
- 5) Острая необходимость использовать модульное и интеграционное тестирование.

Технологии, которые мы будем использовать

- 1) Язык **C++ 11**. Компилятор **clang** работает на большом кол-ве платформ;
- 2) **CMake**. Кроссплатформенная система сборки;
- 3) **GoogleTest**. Кроссплатформенная библиотека для написания unit-тестов;
- 4) **Qt 5**. Кроссплатформенная библиотека для создания пользовательских интерфейсов;
- 5) **OpenGL**. Технология отображения графики.

CMake



- кроссплатформенная утилита для генерации сборочных файлов и проектов для различных IDE.

<https://cmake.org/> - можно скачать под почти любую десктопную платформу.

<https://habrahabr.ru/post/155467/> - полезный пост про введение в данную технологию.

СMake-скрипт

- содержится в файлах CMakeLists.txt;
- определяет набор исходных кодов;
- определяет директивы сборки проекта;
- поддерживает зависимости.

```
cmake_minimum_required(VERSION 2.8)
```

```
add_definitions(--std=c++11)
```

```
# Set up your project name.
```

```
project(template)
```

```
# Set up the main source folder.
```

```
set(SOURCE_ROOT src)
```

```
# Scan source folder to find all sources and put the result to SRC_LIST.
```

```
aux_source_directory(${SOURCE_ROOT} SRC_LIST)
```

```
# Create executable by SRC_LIST.
```

```
add_executable(${PROJECT_NAME} ${SRC_LIST})
```

Добавление зависимости в CMake-скрипт

Add subdirectory with Google Test Library.

```
add_subdirectory(3party/googletest)
```

Enable unit testing.

```
enable_testing()
```

Set up testing project name.

```
set(PROJECT_TEST_NAME ${PROJECT_NAME}_test)
```

Add include directories for testing project.

```
include_directories(3party/googletest/googletest/include ${SOURCE_ROOT})
```

Set up testing project.

```
set(TESTS_SOURCE_ROOT tests)
```

```
aux_source_directory(${TESTS_SOURCE_ROOT} TEST_SRC_FILES)
```

```
set(TEST_SRC_FILES ${SRC_LIST} ${TEST_SRC_FILES})
```

```
list(REMOVE_ITEM TEST_SRC_FILES src/main.cpp)
```

```
add_executable(${PROJECT_TEST_NAME} ${TEST_SRC_FILES})
```

Link gtest and gtest_main libraries.

```
target_link_libraries(${PROJECT_TEST_NAME} gtest gtest_main)
```

Finish tests setting up.

```
add_test(test ${PROJECT_TEST_NAME})
```

Рекомендуемые IDE с поддержкой CMake

1) **Qt Creator** — бесплатный, open-source, входит в поставку библиотеки Qt.

При установке под Windows, выбираете компиляцию через **MinGW!**

2) **JetBrains CLion** — платный, но очень удобный.

Можете попробовать запросить академическую лицензию.

GoogleTest

- кроссплатформенная open-source библиотека для создания и запуска unit-тестов.

<https://github.com/google/googletest> - хранится здесь.

<https://github.com/google/googletest/blob/master/googletest/docs/Primer.md>

- быстрый старт.

Unit-тест

- программный модуль, проверяющий корректность выполнения другого программного модуля.

Проверяемый программный модуль по заданным входным параметрам должен выдавать ожидаемый результат.

Например,

`EXPECT_EQ(1 + 2, 3);`

GoogleTest. Проверки

Строгая проверка	Нестрогая проверка	Проверяет
<code>ASSERT_EQ(val1, val2);</code>	<code>EXPECT_EQ(val1, val2);</code>	<code>val1 == val2</code>
<code>ASSERT_NE(val1, val2);</code>	<code>EXPECT_NE(val1, val2);</code>	<code>val1 != val2</code>
<code>ASSERT_LT(val1, val2);</code>	<code>EXPECT_LT(val1, val2);</code>	<code>val1 < val2</code>
<code>ASSERT_LE(val1, val2);</code>	<code>EXPECT_LE(val1, val2);</code>	<code>val1 <= val2</code>
<code>ASSERT_GT(val1, val2);</code>	<code>EXPECT_GT(val1, val2);</code>	<code>val1 > val2</code>
<code>ASSERT_GE(val1, val2);</code>	<code>EXPECT_GE(val1, val2);</code>	<code>val1 >= val2</code>

GoogleTest. Пример теста

```
#include "gtest/gtest.h"
```

```
#include "dummy.hpp"
```

```
TEST(dummy_test, test_sum)
{
    my::Dummy dummy(1, 2);
    EXPECT_EQ(dummy.Sum(), 3);
}
```

GoogleTest. Пример вывода результатов

```
Running main() from gtest_main.cc
[=====] Running 1 test from 1 test case.
[-----] Global test environment set-up.
[-----] 1 test from dummy_test
[ RUN    ] dummy_test.test_sum
[      OK ] dummy_test.test_sum (0 ms)
[-----] 1 test from dummy_test (0 ms total)

[-----] Global test environment tear-down
[=====] 1 test from 1 test case ran. (0 ms total)
[ PASSED ] 1 test.
```

Шаблон проекта

<https://github.com/rokuz/template>



Домашнее задание

- 1) Создать страничку своего проекта на GitHub;
- 2) Подумать об особенностях вашего проекта;
- 3) Декомпонировать ваш проект.

**Просьба оставить отзыв о
данном занятии на портале!**

Спасибо за внимание!

Роман Кузнецов

r.kuznetsov@mapswithme.com