# Лабораторная работа №9 Тестирование при помощи Google Test Framework

### **Вместо введения**

<https://www.youtube.com/watch?v=pJfe2kOHGwc&feature=emb_title>

**Цель:**

1. Познакомиться с Google Test и Google Mock Framework;
2. Изучить базовые понятия относящийся к тестированию кода;
3. Научиться тестировать классы в среде разработки Qt Creator.

### **Инструменты**

Для выполнения лабораторной работы вам понадобится:

* IDE Qt Creator;
* Google Test и Google Mock Framework;
* ~~Костюм железного человека Mark 2;~~
* Класс для тестирования.

Перед началом работы рекомендуется ознакомиться со списком источников (в конце). Не обязательно смотреть/читать всё полностью, можно просто просмотреть, т.к. там есть примеры, которыми можно будет воспользоваться в процессе выполнения работы.

### **Подготовка «боевого» проекта**

В нашем случае полноценного «боевого» проекта не будет, но для имитации реального рабочего процесса будем считать, что он у нас есть.

Скачайте [архив с проектом](https://neroid.ru/wp-content/uploads/2020/05/Lab9_Project_for_tests.zip) и распакуйте в любое удобное место. Сам проект ничего не делает, так что его можно даже не запускать.

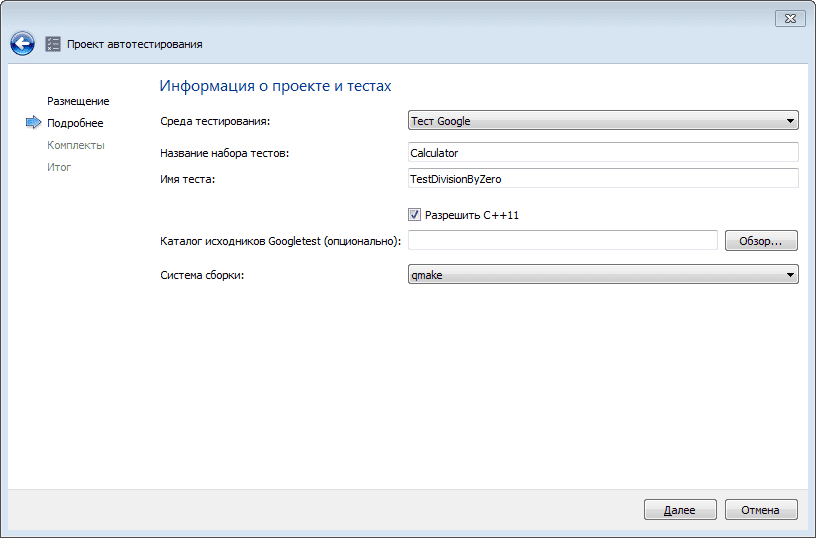
### **Создание тестового проекта в Qt Creator**

Обычно тесты помещают в отдельном проекте — также поступим и мы. Но прежде, чем что-либо создавать, нужно скачать свежую версию тестового фреймворка.

Клонируйте или скачайте архивом Google Test и Google Mock Framework из официального [репозитория](https://github.com/google/googletest). Где будет располагаться папка значения не имеет, но убедитесь, что в пути нет русских символов. Т.к. папка весит не больше 5МБ, то можно не проводить чистку репозитория, а оставить всё как есть.

Запустите Qt Creator. В меню **Файл** выберите **Создать файлы или проекты** (Ctrl+N). В открывшемся окне в списке **Проекты** выберите **Другой проект**, а в списке шаблонов **Проект автотестирования**.

Укажите путь и имя проекта (обычно начинается со слова Test) и нажмите **Далее**. В списке**Среда тестирования** выберите **Тест Google**. Название набора тестов и имя теста можно установить любое, т.к. их легко поменять в процессе. Довольно часто набор тестов называют по имени класса (Calculator), а конкретные тесты в соответствии с тестируемым функционалом (TestDivisionByZero). Установите галочку Разрешить С++11.

  
Рис. 1 Проект автотестирования

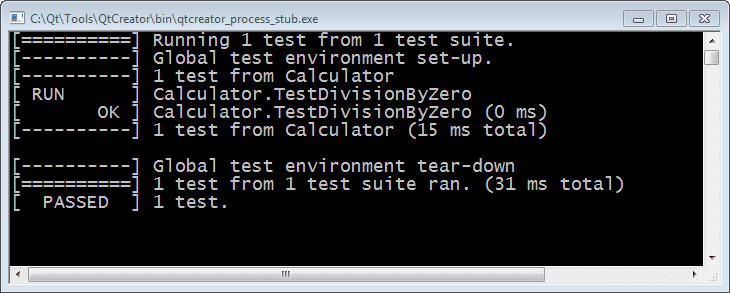
Нажмите на кнопку **Обзор** и выберите корневую папку Google Test (googletest-master). Нажимайте **Далее**пока проект не будет создан.

### **Проверяем работоспособность пустого тестового проекта**

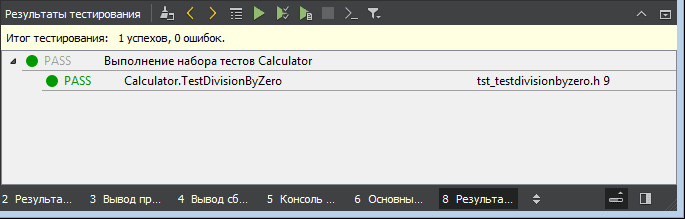
Стандартный шаблон создаёт проект уже готовый к запуску. В этом проекте уже есть один тест, поэтому сразу запустим и посмотрим на результат.

Qt Creator позволяет увидеть результаты тестирования двумя способами.

Начнём с классического. Запустите проект при помощи зелёного треугольника (Ctrl+R), как и любой другой. Перед вами откроется обычное консольное окно, с подробно расписанным процессом тестирования. Текста довольно много и сложновато сразу оценить результат.

  
Рис. 2 Вывод в консоль

Второй способ более наглядный, им и рекомендуется воспользоваться. Внизу окна найдите вкладку **Результаты тестирования** и откройте её (Alt+8). Вверху вкладки вы увидите несколько кнопок, нас интересует зелёный треугольник, он запустит все тесты на выполнение. Результат вы увидите в этом же окне.

  
Рис. 3 Вывод в Результаты тестирования

В процессе работы вы можете получить предупреждения вида:



|  |  |
| --- | --- |
| 1  2 | Using googletest src dir specified at Qt Creator wizard  set GOOGLETEST\_DIR as environment variable or qmake variable to get rid of this message |

Можете просто не обращать на них внимания, на тестирование они никак не повлияет. Но если вы всё же хотите от них избавиться, то перейдите на вкладку **Проекты**(на левой панели) и на странице **Настройки сборки** в категории **Среда сборки** (в самом низу) добавьте переменную среды **GOOGLETEST\_DIR**. В поле **Значение** вставьте путь к корневой папке Google Test.

### **Настройка тестового проекта**

Подготовим тестовый проект к работе.

Откройте .pro файл тестового проекта и добавьте строку:



|  |  |
| --- | --- |
| 1 | INCLUDEPATH += путь\_к\_проекту |

здесь путь\_к\_проекту — это абсолютный или относительный путь к корневой папке «боевого» проекта. Данная строка подскажет компилятору где искать заголовочные файлы, которые мы будем тестировать.

В дереве проекта щёлкните ПКМ по названию тестового проекта (не по .pro файлу) и выберите пункт **Добавить существующие файлы…** В открывшемся окне перейдите в папку с «боевым» проектом и выберите файлы: lockcontroller.h и lockcontroller.cpp (это тестируемый класс), IKeypad.h и ILatch.h (зависимости). В дереве проекта отобразятся подключенные файлы.

Таким образом, если бы мы продолжали вести работу над «боевым» проектом, можно было бы легко прогонять тесты без необходимости копирования файлов из одного проекта в другой.

Т.к. классы IKeypad и ILatch представлены только в виде интерфейсов, в тестовый проект нужно будет добавить два mock класса, которые будут предоставлять «поддельную» реализацию указанных интерфейсов.

Файл main.cpp трогать не нужно вообще, всё, что нужно там уже прописано. Найдите и откройте файл название которого начинается на **tst\_**. Здесь вы как раз и будете писать тесты. В шаблоне, по умолчанию, подключается только gmock-matchers.h, исправьте на gmock.h, чтобы получить достук ко всем возможностям фреймворка. Здесь уже присутствует один тест, удалите его.

О том как замокать классы и как писать тесты вы можете узнать из официальной документации и видео из раздела **Источники**. Для выполнения работы вполне достаточно самых базовых знаний.

### **Тест кейсы которые нужно реализовать**

**Внимание:** один из тестов должен получится красным, т.к. в классе LockController намеренно допущена ошибка. Исправлять ничего не нужно. Вносить изменения, в файлы предоставленные для тестирования, запрещено.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119 | Тест 1  Исходные данные:  - Всё в порядке никаких проблем нет;  Действие:  - Вызывается метод wait() класса LockController;  Ожидается:  - Метод wait() класса IKeypad будет вызван один раз;    Тест 2  Исходные данные:  - Дверь закрыта.  Действие:  - Вызывается метод isDoorOpen() класса LockController;  Ожидается:  - Метод isDoorOpen() класса LockController вернёт false;  - Метод getDoorStatus() класса ILatch будет вызван один раз;    Тест 3  Исходные данные:  - Дверь отрыта.  Действие:  - Вызывается метод isDoorOpen() класса LockController;  Ожидается:  - Метод isDoorOpen() класса LockController вернёт true;  - Метод getDoorStatus() класса ILatch будет вызван один раз;    Тест 4  Исходные данные:  - если нужно, защёлка двери успешно открывается и закрывается. Никаких помех нет.  Действие:  - Вызывается метод unlockDoor() класса LockController;  Ожидается:  - Метод unlockDoor() класса LockController вернёт DoorStatus::OPEN;  - Метод open() класса ILatch будет вызван один раз;    Тест 5  Исходные данные:  - если нужно, защёлка двери успешно открывается и закрывается. Никаких помех нет.  Действие:  - Вызывается метод lockDoor() класса LockController;  Ожидается:  - Метод close() класса ILatch будет вызван один раз;  - Метод lockDoor() класса LockController вернёт DoorStatus::CLOSE;    Тест 6  Исходные данные:  - Объекты классов IKeypad и ILatch созданы, валидны и переданы в конструктор LockController;  - С оборудованием всё в порядке.  Действие:  - Вызывается метод hardWareCheck() класса LockController;  Ожидается:  - Метод hardWareCheck() класса LockController вернёт HardWareStatus::OK;  - Метод isActive() класса IKeypad будет вызван один раз;  - Метод isActive() класса ILatch будет вызван один раз;    Тест 7  Исходные данные:  - Объект класса ILatch создан, валиден и передан в конструктор LockController;  - С задвижкой (ILatch) всё в порядке.  - Вместо объекта класса IKeypad передан nullptr;  Действие:  - Вызывается метод hardWareCheck() класса LockController;  Ожидается:  - Метод hardWareCheck() класса LockController вернёт HardWareStatus::ERROR;  - Метод isActive() класса IKeypad будет вызван ноль или более раз;  - Метод isActive() класса ILatch будет вызван ноль или более раз;    Тест 8  Исходные данные:  - Объекты классов IKeypad и ILatch созданы, валидны и переданы в конструктор LockController;  - С клавиатурой (IKeypad) всё в порядке.  - Задвижка (ILatch) не активна.  Действие:  - Вызывается метод hardWareCheck() класса LockController;  Ожидается:  - Метод hardWareCheck() класса LockController вернёт HardWareStatus::ERROR;  - Метод isActive() класса IKeypad будет вызван ноль или более раз;  - Метод isActive() класса ILatch будет вызван ноль или более раз;    Тест 9  Исходные данные:  - LockController только создан, поэтому установлен дефолтный пароль("0000");  - Пользователь вводит правильный пароль (когда у него попросят).  Действие:  - Вызывается метод isCorrectPassword() класса LockController;  Ожидается:  - Метод isCorrectPassword() класса LockController вернёт true;  - Метод requestPassword() класса IKeypad будет вызван один раз;    Тест 10  Исходные данные:  - LockController только создан, поэтому установлен дефолтный пароль("0000");  - Пользователь вводит НЕ правильный пароль.  Действие:  - Вызывается метод isCorrectPassword() класса LockController;  Ожидается:  - Метод isCorrectPassword() класса LockController вернёт false;  - Метод requestPassword() класса IKeypad будет вызван один раз;    Тест 11  Исходные данные:  - LockController только создан, поэтому установлен дефолтный пароль("0000");  - Старый пароль вводится правильно;  Действие:  - Вызывается метод resetPassword() класса LockController;  Ожидается:  - будет запрошен старый пароль;  - будет установлен новый пароль, введённый пользователем (нельзя на прямую проверить, т.к. password не public и геттера нет);    Тест 12  Исходные данные:  - Пароль изменён на "9876";  - Пользователь правильно вводит пароль;  Действие:  - Вызывается метод resetPassword() класса LockController;  Ожидается:  В результате вызова метода resetPassword() класса LockController будет:  - будет запрошен старый пароль;  - будет установлен новый пароль, введённый пользователем (нельзя на прямую проверить, т.к. password не public и геттера нет); |

### **Что должно быть в отчёте**

* Итоговый тестовый проект добавьте в репозиторий;
* В отчёт вставьте ссылку на этот проект;
* Скриншоты проведённого тестирования. Все пункты должны быть развёрнуты. Если необходимо вставьте несколько скриншотов. Если вы предпочитаете вывод в консоль, можно вставить скриншоты консоли, но все тесты должны быть видны;
* Изучите метод с ошибкой и добавьте в отчёт правильную реализацию метода (всего метода, не только тело);
* Как вы считаете, каким образом могла возникнуть эта ошибка. Ответ добавить в отчёт.

## Источники

1. [Открытый вебинар «Практика использования Google Test Framework»](https://youtu.be/XWP-PeygqDA?t=3619) [видео с таймингом]
2. [Googletest Primer](https://github.com/google/googletest/blob/master/googletest/docs/primer.md)
3. [gMock for Dummies](https://github.com/google/googletest/blob/master/googlemock/docs/for_dummies.md)
4. [Остальная документация из репозитория](https://github.com/google/googletest)
5. [TDD для начинающих](https://vimeo.com/9541997) [видео]
6. [Кент Бек — Экстремальное программирование. Разработка через тестирование](https://neroid.ru/wp-content/uploads/2020/05/Kent-Bek-Jekstremalnoe-programmirovanie.-Razrabotka-cherez-testirovanie.pdf) (Часть II не обязательно)