C++. Базовый уровень

Тестирование приложений на C++: использование различных методов тестирования, таких как модульное и интеграционное тестирование, для обеспечения качества кода.

КОД будущего

Минцифры России

инжиниринум
МГТУ им. Н.Э. Баумана

Национальные проекты России
ЦИФРОВАЯ ЭКОНОМИКА

20.35 УНИВЕРСИТЕТ

2023

Тестирование

Тестирование ПО - процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранных определенным образом

Тестирование

В более широком смысле, тестирование — это одна из техник контроля **качества**, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Цели тестирования

- Повысить вероятность того, что приложение, предназначенное для тестирования, будет работать правильно при любых обстоятельствах.
- Повысить вероятность того, что приложение, предназначенное для тестирования, будет соответствовать всем описанным требованиям.
- Предоставление актуальной информации о состоянии продукта на данный момент.

Тестирование

Классификация видов тестирования

1. По запуску кода на исполнение - статическое/динамическое тестирование
2. По уровню детализации приложения(по уровню тестирования) - модульное/интеграционное/системное тестирование
3. По фокусировке на уровне архитектуры приложения - уровень представления/уровень бизнес-логики/уровень данных
4. По доступу к коду и архитектуре приложения - метод белого/черного/серого ящика

Тестирование

5. По (убыванию) степени важности тестируемых функций (по уровню функционального тестирования) - “дымовое”/критического пути/расширенное
6. По привлечению конечных пользователей - альфа/бета/гамма тестирование
7. По уровню автоматизации - ручное/автоматизированное (автоматическое)
8. По природе приложения - веб/мобильное/настольное/….
9. По степени формализации - на основе тест-кейсов/исследовательское /свободное (интуитивное)

Тестирование

Классификация тестирования по уровню детализации приложения.

Модульное тестирование (Unit Testing) – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения. Его цель заключается в том, чтобы проверить, что каждая единица программного кода работает должным образом. Данный вид тестирование выполняется разработчиками на этапе кодирования приложения. Модульные тесты изолируют часть кода и проверяют его работоспособность. Единицей для измерения может служить отдельная функция, метод, процедура, модуль или объект.

Тестирование

Интеграционное тестирование (Integration Testing, иногда Integration and Testing, аббревиатура I&T) — одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе.
Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определенные в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

Тестирование

Системное тестирование — это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы. Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований к системе в целом.

Тестирование

Приемочное тестирование - это заключительный этап тестирования перед развертыванием программного обеспечения. Приемочное тестирование делается для проверки готовности программного обеспечения выполнять задачи и функции, поставленные при разработке.

Существуют три базовые стратегии выполнить приемочное тестирование, а именно:

- Формальная приемка
- Неформальная приемка или альфа-тестирование
- Бета-тестирование

Выбор стратегии часто зависит от требований контракта, корпоративных стандартов и сферы применения приложения.

Тестирование

Функциональные и нефункциональные виды тестирования.

Функциональные тесты базируются на функциях и особенностях, а также на взаимодействии с другими системами и могут быть представлены на всех уровнях тестирования: компонентном или модульном (Component/Unit testing), интеграционном (Integration testing), системном (System testing), приемочном (Acceptance testing). Функциональные виды тестирования рассматривают внешнее поведение системы.

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

Тестирование

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, как система работает.

Примеры:

- Тестирование производительности
- Стрессовое тестирование
- Тестирование удобства пользования и др.

Тестирование

Google C++ Testing Framework (Google Test) — библиотека для модульного тестирования (англ. unit testing) на языке C++.

Ключевым понятием в Google test framework является понятие утверждения (**assert**). ***Утверждение*** представляет собой выражение, результатом выполнения которого может быть успех (**success**), некритический отказ (**nonfatal failure**) и критический отказ (**fatal failure**). Критический отказ вызывает завершение выполнения теста, в остальных случаях тест продолжается. Сам *тест* представляет собой набор утверждений. Кроме того, тесты могут быть сгруппированы в ***наборы* (test case).** Если сложно настраиваемая группа объектов должна быть использована в различных тестах, можно использовать ***фиксации* (fixture).** Объединенные наборы тестов являются ***тестовой программой* (test program).**

Тестирование

Утверждения, порождающие в случае их ложности критические отказы начинаются с **ASSERT**_, некритические — **EXPECT**_. Следует иметь ввиду, что в случае критического отказа выполняется немедленный возврат из функции, в которой встретилось вызвавшее отказ утверждение. Если за этим утверждением идет какой-то очищающий память код или какие-то другие завершающие процедуры, можете получить утечку памяти.

Тестирование

Простейшие логические:
- ASSERT_TRUE(condition);
- ASSERT_FALSE(condition);

Сравнение:
- ASSERT_EQ(expected, actual); — =
- ASSERT_NE(val1, val2); — !=
- ASSERT_LT(val1, val2); — <
- ASSERT_LE(val1, val2); — <=
- ASSERT_GT(val1, val2); — >
- ASSERT_GE(val1, val2); — >=

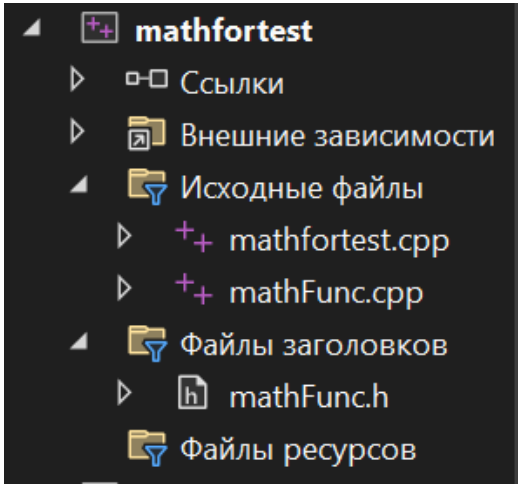
Проверка на исключения:
- ASSERT_THROW(statement, exception_type);
- ASSERT_ANY_THROW(statement);
- ASSERT_NO_THROW(statement);

Сравнение строк:
- ASSERT_STREQ(expected_str, actual_str);
- ASSERT_STRNE(str1, str2);
- ASSERT_STRCASEEQ(expected_str, actual_str); — регистронезависимо
- ASSERT_STRCASENE(str1, str2); — регистронезависимо

Сравнение чисел с плавающей точкой:
- ASSERT_FLOAT_EQ(expected, actual); — неточное сравнение float
- ASSERT_DOUBLE_EQ(expected, actual); — неточное сравнение double
- ASSERT_NEAR(val1, val2, abs_error); — разница между val1 и val2 не превышает погрешность abs_error

Тестирование

Попробуем написать свои тесты.
Для начала создадим консольное приложение, добавим туда хедер и исходный файл:



Тестирование

Объявим в хедер 4 функции:

```
#pragma once
#include <iostream>

int factorial(int n);
float divAtoB(float a, float b);
bool isPositiv(int n);
std::string reverseStr(std::string str);
```

Первая считает факториал, вторая делит первое число на второе, третья проверят, что число положительное, четвертая переворачивает строку.

Задание

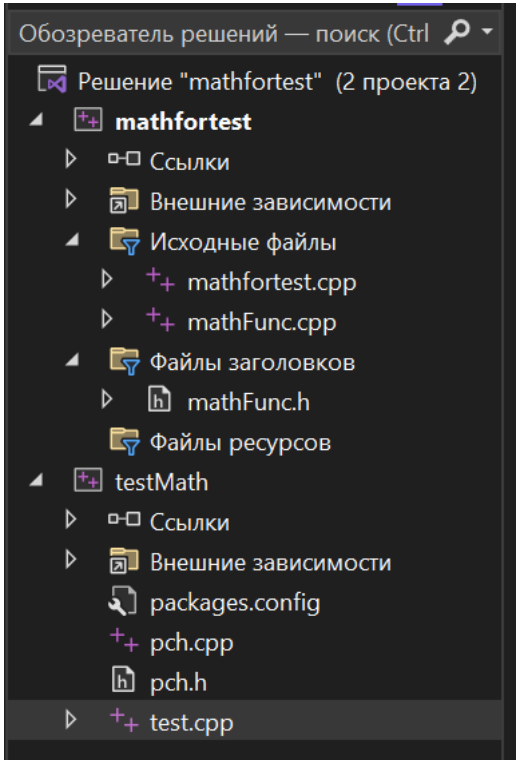
Реализуйте указанные функции в файле **mathFunc.cpp**

Тестирование

С помощью обозревателя решений добавьте новый проект. Для этого нажмите на «Решение mathfotest» правой кнопкой мыши -> Добавить -> Создать проект.
В списке проектов найдите Google Test. Привяжите созданный проект к проекту mathfotest.

Напишем первый тест. Для этого в файле теста подключим mathFunc.cpp (если не видит, укажите полный путь к файлу)

Тестирование



Тестирование

```
TEST(TestCaseName, TestInt) {
  ASSERT_EQ(120, factorial(5));
}
```

TEST принимает 2 параметра, уникально идентифицирующие тест, — **название тестового набора** и **название теста**. В рамках одного и того же тестового набора названия тестов не должны совпадать.

В данном тесте мы проверяем, что функция факториала при переданном значении 5 вернет нам 120.

Тестирование

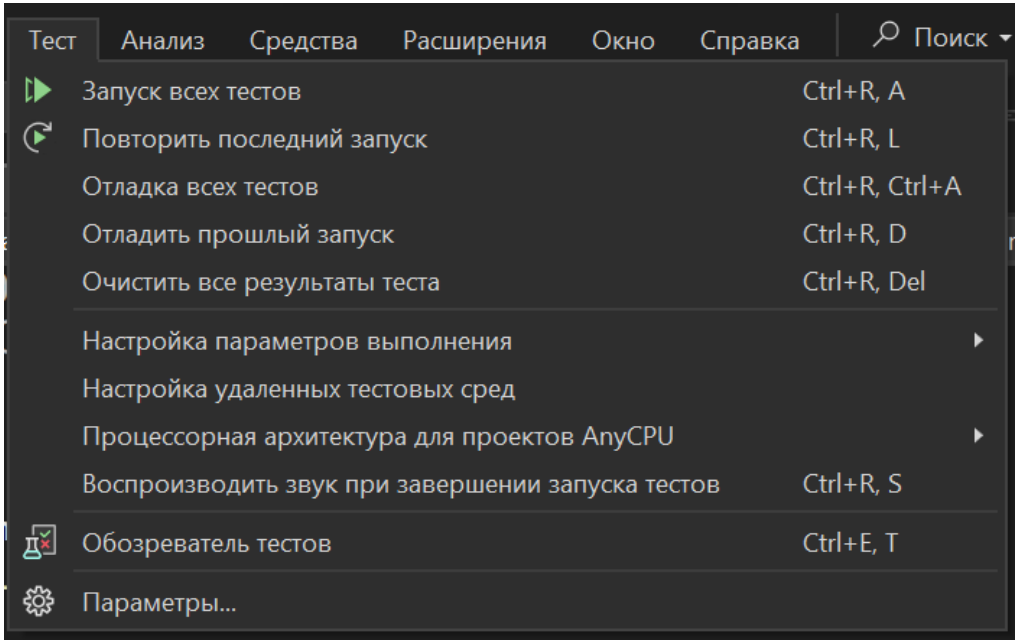
```
TEST(TestCaseName, TestFloat) {
    ASSERT_FLOAT_EQ(2.5, divAtoB(5.0, 2.0));
}

TEST(TestCaseName, TestBool) {
    ASSERT_EQ(1, isPositiv(5));
}

TEST(TestCaseName, TestString) {
    std::string str = "12345";
    std::string res = "54321";
    ASSERT_TRUE(res == reverseStr(str));
}
```

Тестирование

Чтобы запустить тесты, перейдите в Тест -> Запуск всех тестов



Тестирование

Как видим, все тесты пройдены