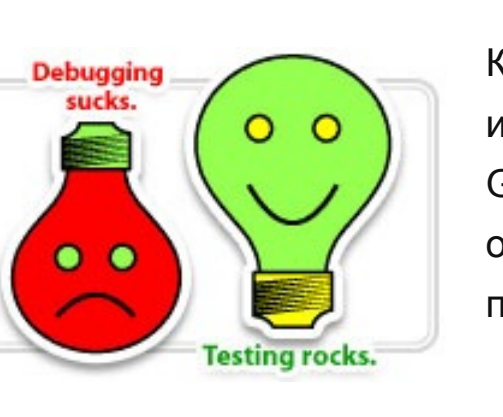
hoxnox 11 мая 2011 в 15:18

Google testing framework (gtest)

C++*

Tutorial



Когда вставал вопрос о тестировании кода, я не задумываясь использовал boost::test. Для расширения кругозора попробовал Google Test Framework. Помимо всяких имеющихся в нем плюшек, в отличии от boost::test проект бурно развивается. Хотел бы поделиться приобретенными знаниями. Всем кому интересно прошу

Ключевые понятия

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).

Утверждения (assertion)

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.

Имеются следующие утверждения (некритические начинаются не с ASSERT_, а с EXPECT_):

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); — >=

Сравнение строк

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

Проверка на исключения

- ASSERT_THROW(statement, exception_type);
- ASSERT_ANY_THROW(statement);
- ASSERT_NO_THROW(statement);

Проверка предикатов

- ASSERT_PREDN(pred, val1, val2, ..., valN); — N <= 5
- ASSERT_PRED_FORMATN(pred_format, val1, val2, ..., valN); — работает аналогично предыдущей, но позволяет контролировать вывод

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

Вызов отказа или успеха

- SUCCEED();
- FAIL();
- ADD_FAILURE();
- ADD_FAILURE_AT(«file_path», line_number);

Можно написать собственную функцию, возвращающую AssertionResult

```
::testing::AssertionResult IsTrue(bool foo)
{
    if (foo)
        return ::testing::AssertionSuccess();
    else
        return ::testing::AssertionFailure() << foo << " is not true";
}

TEST(MyFunCase, TestIsTrue)
{
    EXPECT_TRUE(IsTrue(false));
}
```

Можно контролировать типы данных с помощью функции ::testing::StaticAssertTypeEq<T1, T2>(). Компиляция пройдет с ошибкой в случае несовпадения типов T1 и T2.

В случае ложности утверждения, выдаются данные, использованные в утверждении. Кроме того, можно задать собственный комментарий:

```
ASSERT_EQ(1, 0) << "1 is not equal 0";
```

Можно использовать расширенные наборы символов (wchar_t) как в комментариях, так и в утверждениях, касающихся строк. При этом выдача будет в UTF-8 кодировке.

Тесты (tests)

Для определения теста используется макрос TEST. Он определяет void функцию, в которой можно использовать утверждения. Как отмечалось ранее, критический отказ вызывает немедленный возврат из функции.

```
TEST(test_case_name, test_name)
{
    ASSERT_EQ(1, 0) << "1 is not equal 0";
}
```

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. Если название начинается с DISABLED_, это означает, что вы пометили тест (набор тестов) как временно не используемый.

Можно использовать утверждения не только в составе теста, но и вызывать их из любой функции. Имеется лишь одно ограничение — утверждения, порождающие критические отказы не могут быть вызваны из не void функций.

Фиксации (fixtures)

Случается, что объекты, участвующие в тестировании, сложно настраиваются для каждого теста. Можно задать процесс настройки один раз и исполнять его для каждого теста автоматически. В таких ситуациях используются фиксации.

Фиксация представляет собой класс, унаследованный от ::testing::Test, внутри которого объявлены все необходимые для тестирования объекты при этом в конструкторе либо функции SetUp() выполняется их настройка, а в функции TearDown() освобождение ресурсов. Сами тесты, в которых используются фиксации, должны быть объявлены с помощью макроса TEST_F, в качестве первого параметра которого должно быть указано не название набора тестов, а название фиксации.

Для каждого теста будет создана новая фиксация, настроена с помощью SetUp(), запущен тест, освобождены ресурсы с помощью TearDown() и удален объект фиксации. Таким образом каждый тест будет иметь свою копию фиксации «не испорченную» предыдущим тестом.

```
#include <gtest/gtest.h>
#include <iostream>

class Foo
{
public:
    Foo()
        : i(0)
    {
        std::cout << "CONSTRUCTED" << std::endl;
    }
    ~Foo()
    {
        std::cout << "DESTRUCTED" << std::endl;
    }
    int i;
};

class TestFoo : public ::testing::Test
{
protected:
    void SetUp()
    {
        foo = new Foo;
        foo->i = 5;
    }
    void TearDown()
    {
        delete foo;
    }
    Foo *foo;
};

TEST_F(TestFoo, test1)
{
    ASSERT_EQ(foo->i, 5);
    foo->i = 10;
}

TEST_F(TestFoo, test2)
{
    ASSERT_EQ(foo->i, 5);
}

int main(int argc, char *argv[])
{
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

В некоторых случаях создание тестируемых объектов является очень дорогой операцией, а тесты не вносят никаких изменений в объекты. В таком случае можно не создавать фиксации заново для каждого теста, а использовать распределенную фиксацию с глобальным SetUp()и TearDown(). Фиксация автоматически становится распределенной, если в классе имеется хотя бы один статический член. Статические функции SetUpTestCase() и TearDownTestCase() будут вызываться для настройки объекта и освобождения ресурсов соответственно. Таким образом, набор тестов перед первым тестом вызовет SetUpTestCase(), а после последнего TearDownTestCase().

Если существует потребность в SetUp() и TearDown() для всей программы тестирования, а не только для набора теста, необходимо создать класс-наследник для ::testing::Environment, переопределить SetUp() и TearDown() и зарегистрировать его с помощью функции AddGlobalTestEnvironment.

Запуск тестов

Объявив все необходимые тесты, мы можем запустить их с помощью функции RUN_ALL_TESTS(). Функцию можно вызывать только один раз. Желательно, чтобы тестовая программа возвращала результат работы функции RUN_ALL_TESTS(), так как некоторые автоматические средства тестирования определяют результат выполнения тестовой программы по тому, что она возвращает.

Флаги

Вызванная перед RUN_ALL_TESTS() функция InitGoogleTest(argc, argv) делает вашу тестовую программу не просто исполняемым файлом, выводящим на экран результаты тестирования. Это целостное приложение, принимающие на вход параметры, меняющие его поведение. Как обычно ключи -h, --help дадут вам список всех поддерживаемых параметров. Перечислю некоторые из них (за полным списком можно обратиться к документации).

- ./test --gtest_filter=TestCaseName.*-TestCaseName.SomeTest — запустить все тесты набора TestCaseName за исключением SomeTest
- ./test --gtest_repeat=1000 --gtest_break_on_failure — запустить тестирующую программу 1000 раз и остановиться при первой неудаче
- ./test --gtest_output=«xml:out.xml» — помимо выдачи в std::out будет создан out.xml — XML отчет с результатами выполнения тестовой программы
- ./test --gtest_shuffle — запускать тесты в случайном порядке

Если вы используете какие-то параметры постоянно, можете задать соответствующую переменную окружения и запускать исполняемый файл без параметров. Например задание переменной GTEST_ALSO_RUN_DISABLED_TESTS ненулевого значения эквивалентно использованию флага --gtest_also_run_disabled_tests.

Вместо заключения

В данном посте я кратко пробежался по основным функциям Google Test Framework. За более подробными сведениями следует обратиться к документации. Оттуда вы сможете почерпнуть информацию о ASSERT_DEATH используемом при падении программы, о ведении дополнительных журналов, о параметризованных тестах, настройке вывода, тестировании закрытых членов класса и многое другое.

UPD: По справедливому замечанию хабрапользователя @nikel добавлена краткая инофрмация по использованию флагов.
UPD 2: Исправление разметки после изменений на Хабре (нативный тег source).

Теги: Google testing framework, unit testing, google, test, gtest

Хабы: C++

Редакторский дайджест

Присылаем лучшие статьи раз в месяц

Электропочта

59 Карма | 0 Рейтинг

@hoxnox

Пользователь

Комментарии 23

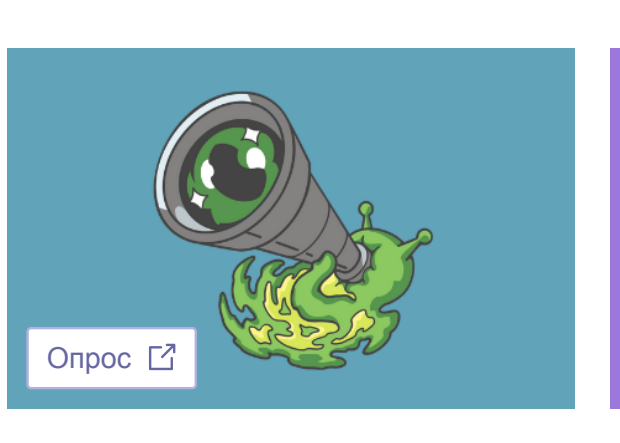
ПОХОЖИЕ ПУБЛИКАЦИИ

19 апреля в 11:04
Google начала работать над возвращением Google Wallet
+3 | 2.8K | 1 | 0

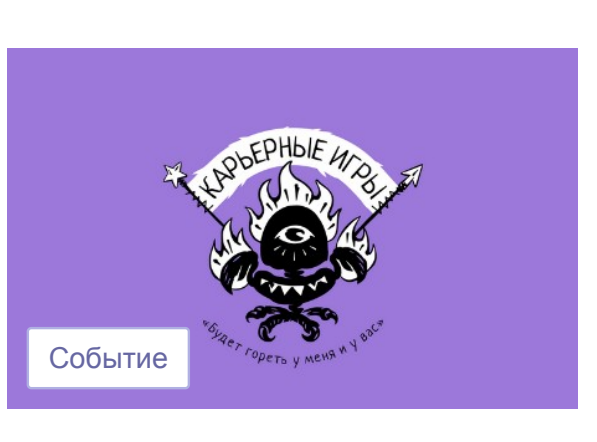
20 сентября 2013 в 19:15
Обновился Google Testing Framework
+5 | 4.9K | 10 | 0

24 ноября 2011 в 15:09
Автоматический запуск unit-тестов для C
+3 | 9.5K | 44 | 5 +5

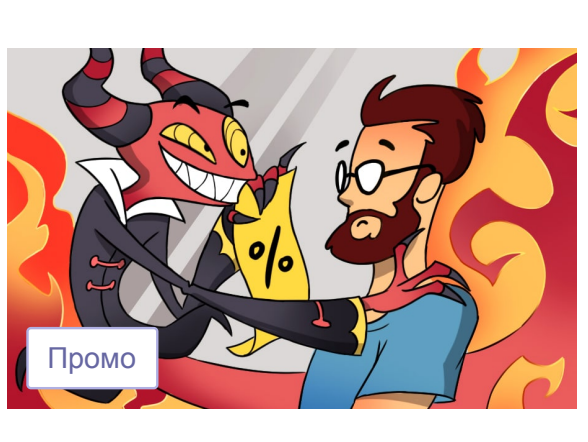
МИНУТОЧКУ ВНИМАНИЯ | Разместить



Хотите рассказать о себе в наших социальных сетях?



Дарим бесплатный билет на РИФ за профиль на Хабр Карьере

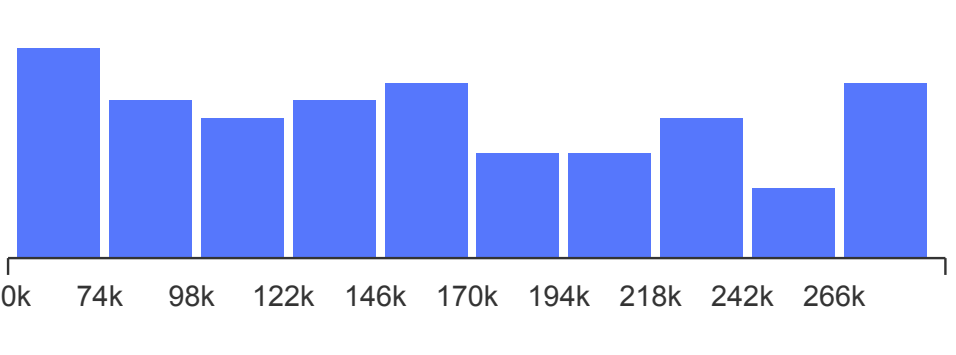


Промокод — твой билет в общество потребления

СРЕДНЯЯ ЗАРПЛАТА В IT

153 765 /мес.

— средняя зарплата во всех IT-специализациях по данным из 9 614 анкет, за 1-ое пол. 2022 года. Проверьте «в рынке» ли ваша зарплата или нет!



Проверить свою зарплату

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

вчера в 02:27
Как отлаживать bash-script-ы по шагам или, возможно, самая короткая статья о программировании/отладке на Хабре
+123 | 9.7K | 261 | 18 +18

вчера в 13:00
Смайлики: как тексты в интернете обрели эмоции
+33 | 1.4K | 17 | 4 +4

вчера в 13:29
Во что поиграть, если шахматы ещё не придумали, а ты фараон / викинг / вождь майя
+22 | 2.9K | 29 | 4 +4

вчера в 06:42
Неочевидные способы оценки городов для переезда номада
+20 | 5.4K | 18 | 48 +48

вчера в 10:00
Умный дом — теперь и у геккона
+16 | 2.3K | 14 | 10 +10

Как мы научились запускать 10-часовые UI-тесты за 5 минут в условиях 30 релизов в день
Турбо

ЧИТАЮТ СЕЙЧАС

Создатель Ethereum Виталик Бутерин заявил, что он больше не миллиардер
3.8K | 7 +7

Программное обеспечение на российской ОС
1.1K | 8 +8

В РФ взялись за импортозамещение оборудования для производства 28-нм чипов: началась разработка литографических систем
58K | 180 +180

В России разрешат использовать музыку и фильмы правообладателей, прекративших работать в стране
1.5K | 6 +6

Солнце и нененависть в Лондоне
4.7K | 27 +27

Как создать цифровой двойник электроэнергетического объекта: Sampled Values
Турбо

РАБОТА

Программист C++
111 вакансий

QT разработчик
12 вакансий

Все вакансии