

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
федеральное государственное автономное образовательное учреждение высшего образования  
«Санкт–Петербургский государственный университет  
аэрокосмического приборостроения»

ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
РУКОВОДИТЕЛЬ \_\_\_\_\_

преподаватель \_\_\_\_\_  
должность, уч. степень,  
звание \_\_\_\_\_ подпись, дата \_\_\_\_\_ инициалы, фамилия \_\_\_\_\_

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине МДК. 01.02

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № С326 \_\_\_\_\_ подпись, дата \_\_\_\_\_ Э.С. Тигранян  
инициалы, фамилия \_\_\_\_\_

Санкт-Петербург 2025

## **Лабораторная работа № 4. Модульное тестирование.**

Цель работы: закрепить теоретические знания и получить практические навыки в применении модульного тестирования.

### **Часть 1.**

Подготовка модуля с функциями для последующего тестирования.

Требуется создать консольное приложение, содержащее пару функций, предназначенных для последующего тестирования.

Вариант 23:

Даны две дроби A/B и C/D. (A, B, C, D натуральные числа). Составить функции,

возвращающие числитель и знаменатель дроби – результата деления дроби на дробь.

Ответ должен быть несократимой дробью.

Код:

```
#include <iostream>
#include <stdexcept>
#include <cassert>
using namespace std;

struct Fraction {
    Fraction(int numerator, int denominator) {
        if (denominator == 0) {
            throw invalid_argument("Знаменатель не может быть равен нулю");
        }
        this->numerator = numerator;
        this->denominator = denominator;
    }

    Fraction(const Fraction& other) {
        this->numerator = other.numerator;
        this->denominator = other.denominator;
    }

    int getNumerator() const {
        return numerator;
    }

    int getDenominator() const {
        return denominator;
    }

    void shortenAFraction() {
        int gcd = findGCD(abs(numerator), abs(denominator));
        if (gcd > 1) {
            numerator /= gcd;
            denominator /= gcd;
        }
        if (denominator < 0) {

```

```

        numerator = -numerator;
        denominator = -denominator;
    }

// (алгоритм Евклида)
int findGCD(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

private:
    int numerator;
    int denominator;
};

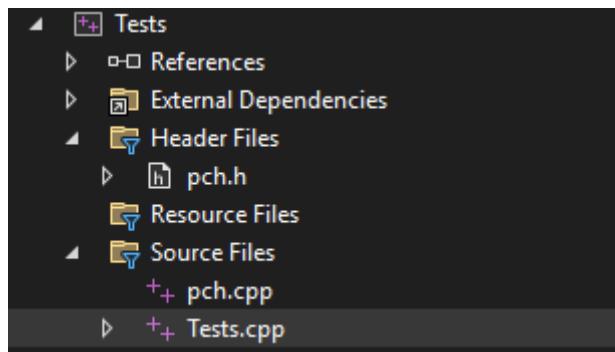
Fraction FractionDivision(Fraction first, Fraction second) {
    throw runtime_error("Ошибка: деление на ноль");
}

Fraction result(first.getNumerator() * second.getDenominator(),
               first.getDenominator() * second.getNumerator());
result.shortenAFraction();
return result;
}

```

## Часть 2.

В этой части работы требуется создать тестирующее приложение (проект), для проверки правильности работы ранее созданных функций.



Проект для тестов

Код:

```

#include "pch.h"
#include "CppUnitTest.h"
#include "../Code/FractionDivision.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace FractionDivisionTests
{
    TEST_CLASS(FractionDivisionTests)
    {

```

```

public:
    TEST_METHOD(DivideNormalFractions)
    {
        Fraction first(3, 5);
        Fraction second(6, 7);

        Fraction result = FractionDivision(first, second);

        Assert::AreEqual(result.getNumerator(), 7);
        Assert::AreEqual(result.getDenominator(), 10);
    }

    TEST_METHOD(DivideAndSimplifyResult)
    {
        Fraction first(4, 9);
        Fraction second(2, 3);

        Fraction result = FractionDivision(first, second);

        Assert::AreEqual(result.getNumerator(), 2);
        Assert::AreEqual(result.getDenominator(), 3);
    }

    TEST_METHOD(DivideByOne)
    {
        Fraction first(5, 8);
        Fraction second(1, 1);

        Fraction result = FractionDivision(first, second);

        Assert::AreEqual(result.getNumerator(), 5);
        Assert::AreEqual(result.getDenominator(), 8);
    }

    TEST_METHOD(DivideByFractionToGetReciprocal)
    {
        Fraction first(1, 1);
        Fraction second(3, 4);

        Fraction result = FractionDivision(first, second);

        Assert::AreEqual(result.getNumerator(), 4);
        Assert::AreEqual(result.getDenominator(), 3);
    }

    TEST_METHOD(DivideFractionByZero) {
        Fraction f1(1, 2);
        Fraction zero(0, 1);

        try {
            FractionDivision(f1, zero);
            assert(false && "Ожидаемое исключение не было вызвано");
        }
        catch (const runtime_error& e) {
            assert(string(e.what()) == "Ошибка: деление на ноль");
        }
        catch (...) {
            assert(false && "Ожидаемое исключение не было вызвано");
        }

        try {
            Fraction invalid(1, 0);
            assert(false && "Ожидаемое исключение не было вызвано");
        }
        catch (const invalid_argument& e) {
    
```

```

        assert(string(e.what()) == "Знаменатель не может быть равен нулю");
    }
    catch (...) {
        assert(false && "Ожидаемое исключение не было вызвано");
    }
}

TEST_METHOD(DivideNegativeFractions)
{
    Fraction first(-1, 2);
    Fraction second(3, -4);

    Fraction result = FractionDivision(first, second);

    Assert::AreEqual(result.getNumerator(), 2);
    Assert::AreEqual(result.getDenominator(), 3);
}

TEST_METHOD(DivideLargeNumbers)
{
    Fraction first(1000, 999);
    Fraction second(100, 999);

    Fraction result = FractionDivision(first, second);

    Assert::AreEqual(result.getNumerator(), 10);
    Assert::AreEqual(result.getDenominator(), 1);
}

TEST_METHOD(ResultIsIrreducible)
{
    Fraction first(7, 11);
    Fraction second(13, 17);

    Fraction result = FractionDivision(first, second);

    Assert::AreEqual(result.getNumerator(), 7 * 17);
    Assert::AreEqual(result.getDenominator(), 11 * 13);
}
};;
}

```

## Лабораторная работа № 5.

Интеграционное тестирование

Цель работы: получение навыков интеграционного тестирования.

Вариант 23:

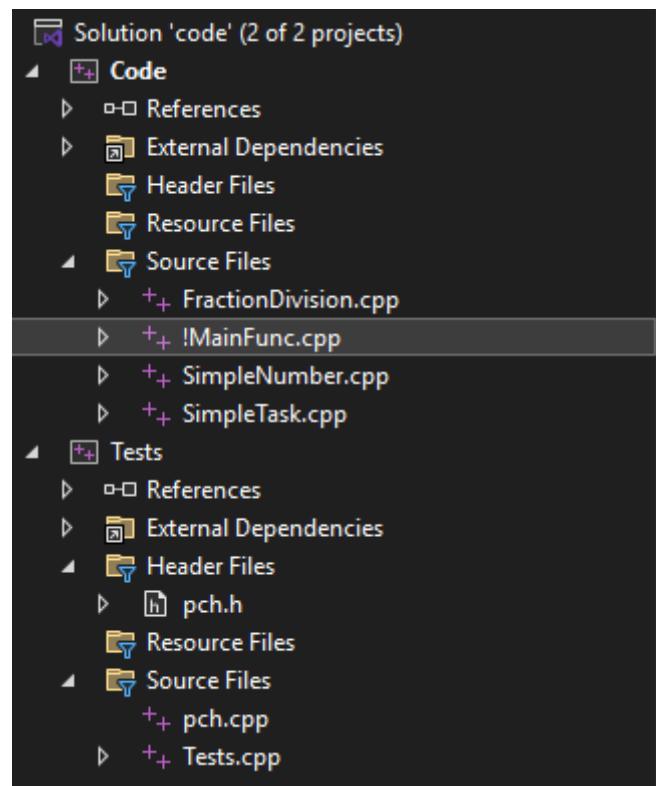
Даны две дроби A/B и C/D. (A, B, C, D натуральные числа). Составить функции, возвращающие числитель и знаменатель дроби – результата деления дроби на дробь.

Ответ должен быть несократимой дробью.

Вариант 25. Составить функцию, проверяющую, являются ли данное число простым. Вторая функция должна вернуть ближайшее большее данного простое число. Например, для числа 11 первая функция возвращает true, вторая – 13. Для числа 14 первая функция должна вернуть false, а вторая – 17.

Вариант 1. Даны 3 целых числа. Написать функции Min3 и Max3 нахождения большего и меньшего из 3-х чисел.

Структура проекта:



## Код модуля FractionDivision

```
#include <iostream>
#include <stdexcept>
#include <cassert>
using namespace std;

struct Fraction {
    Fraction(int numerator, int denominator) {
        if (denominator == 0) {
            throw invalid_argument("Знаменатель не может быть равен нулю");
        }
        this->numerator = numerator;
        this->denominator = denominator;
    }

    Fraction(const Fraction& other) {
        this->numerator = other.numerator;
        this->denominator = other.denominator;
    }

    int getNumerator() const {
        return numerator;
    }

    int getDenominator() const {
        return denominator;
    }

    void shortenAFraction() {
        int gcd = findGCD(abs(numerator), abs(denominator));
        if (gcd > 1) {
            numerator /= gcd;
            denominator /= gcd;
        }
        if (denominator < 0) {
            numerator = -numerator;
            denominator = -denominator;
        }
    }

    // (алгоритм Евклида)
    int findGCD(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

private:
    int numerator;
    int denominator;
};

Fraction FractionDivision(Fraction first, Fraction second) {
    // Проверка деления на ноль (если вторая дробь равна 0)
    if (second.getNumerator() == 0) {
        throw runtime_error("Ошибка: деление на ноль");
    }

    Fraction result(first.getNumerator() * second.getDenominator(),
                   first.getDenominator() * second.getNumerator());
    result.shortenAFraction();
}
```

```
        return result;
    }
```

## Код модуля SimpleNumber

```
//25. Составить функцию, проверяющую, являются ли данное число простым.  
//Вторая функция должна вернуть ближайшее большее данного простое  
//число.Например, для числа 11 первая функция возвращает true, вторая –  
//13. Для числа 14 первая функция должна вернуть false, а вторая – 17.
```

```
#include <iostream>  
#include <cmath>  
  
// Функция проверки, является ли число простым  
bool isPrime(int n) {  
    if (n <= 1) {  
        return false;  
    }  
    if (n == 2) {  
        return true;  
    }  
    if (n % 2 == 0) {  
        return false;  
    }  
    for (int i = 3; i <= std::sqrt(n); i += 2) {  
        if (n % i == 0) {  
            return false;  
        }  
    }  
    return true;  
}  
  
// Функция поиска ближайшего большего простого числа  
int nextPrime(int n) {  
    int candidate = n + 1;  
    while (true) {  
        if (isPrime(candidate)) {  
            return candidate;  
        }  
        candidate++;  
    }  
}
```

## Код модуля SimpleTask

```
//1. Даны 3 целых числа.Написать функции Min3 и Max3 нахождения большего и  
//меньшего из 3 – х чисел.
```

```
#include <iostream>  
  
int Min3(int a, int b, int c) {  
    int min_ab = (a < b) ? a : b;  
    return (min_ab < c) ? min_ab : c;  
}  
int Max3(int a, int b, int c) {  
    int max_ab = (a > b) ? a : b;  
    return (max_ab > c) ? max_ab : c;  
}
```

## Код файла Tests

```
#include "pch.h"  
#include "CppUnitTest.h"
```

```

#include "../Code/FractionDivision.cpp"
#include "../Code/SimpleNumber.cpp"
#include "../Code/SimpleTask.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace FractionDivisionTests
{
    TEST_CLASS(SimpleNumberTest) {

        // Тесты для функции isPrime()
        TEST_METHOD(IsPrime_ForNegativeNumbers) {
            Assert::IsFalse(isPrime(-1));
            Assert::IsFalse(isPrime(-10));
            Assert::IsFalse(isPrime(-100));
        }

        TEST_METHOD(IsPrime_ForZeroAndOne) {
            Assert::IsFalse(isPrime(0));
            Assert::IsFalse(isPrime(1));
        }

        TEST_METHOD(IsPrime_ForSmallPrimes) {
            Assert::IsTrue(isPrime(2));
            Assert::IsTrue(isPrime(3));
            Assert::IsTrue(isPrime(5));
            Assert::IsTrue(isPrime(7));
            Assert::IsTrue(isPrime(11));
            Assert::IsTrue(isPrime(13));
        }

        TEST_METHOD(IsPrime_ForSmallNonPrimes) {
            Assert::IsFalse(isPrime(4));
            Assert::IsFalse(isPrime(6));
            Assert::IsFalse(isPrime(8));
            Assert::IsFalse(isPrime(9));
            Assert::IsFalse(isPrime(10));
            Assert::IsFalse(isPrime(12));
        }

        TEST_METHOD(IsPrime_ForLargePrimes) {
            Assert::IsTrue(isPrime(997));
            Assert::IsTrue(isPrime(7919));
            Assert::IsTrue(isPrime(104729));
        }

        TEST_METHOD(IsPrime_ForLargeNonPrimes) {
            Assert::IsFalse(isPrime(1000));
            Assert::IsFalse(isPrime(7920));
            Assert::IsFalse(isPrime(104730));
        }

        // Тесты для функции nextPrime()
        TEST_METHOD(NextPrime_ForNegativeNumbers) {
            Assert::AreEqual(nextPrime(-10), 2);
            Assert::AreEqual(nextPrime(-1), 2);
        }

        TEST_METHOD(NextPrime_ForZeroAndOne) {
            Assert::AreEqual(nextPrime(0), 2);
            Assert::AreEqual(nextPrime(1), 2);
        }

        TEST_METHOD(NextPrime_ForPrimes) {
            Assert::AreEqual(nextPrime(2), 3);
        }
    }
}

```

```

        Assert::AreEqual(nextPrime(3), 5);
        Assert::AreEqual(nextPrime(11), 13);
        Assert::AreEqual(nextPrime(997), 1009);
    }

    TEST_METHOD(NextPrime_ForNonPrimes) {
        Assert::AreEqual(nextPrime(4), 5);
        Assert::AreEqual(nextPrime(10), 11);
        Assert::AreEqual(nextPrime(14), 17);
        Assert::AreEqual(nextPrime(1000), 1009);
    }
};

TEST_CLASS(SimpleTask) {

    // Тесты для функции Min3()
    TEST_METHOD(Min3_AllEqual) {
        Assert::AreEqual(Min3(5, 5, 5), 5);
        Assert::AreEqual(Min3(0, 0, 0), 0);
        Assert::AreEqual(Min3(-3, -3, -3), -3);
    }

    TEST_METHOD(Min3_FirstIsMin) {
        Assert::AreEqual(Min3(1, 2, 3), 1);
        Assert::AreEqual(Min3(-5, 0, 5), -5);
        Assert::AreEqual(Min3(10, 20, 30), 10);
    }

    TEST_METHOD(Min3_SecondIsMin) {
        Assert::AreEqual(Min3(2, 1, 3), 1);
        Assert::AreEqual(Min3(0, -5, 5), -5);
        Assert::AreEqual(Min3(20, 10, 30), 10);
    }

    TEST_METHOD(Min3_ThirdIsMin) {
        Assert::AreEqual(Min3(3, 2, 1), 1);
        Assert::AreEqual(Min3(5, 0, -5), -5);
        Assert::AreEqual(Min3(30, 20, 10), 10);
    }

    TEST_METHOD(Min3_WithNegativeNumbers) {
        Assert::AreEqual(Min3(-1, -2, -3), -3);
        Assert::AreEqual(Min3(-10, -5, -1), -10);
        Assert::AreEqual(Min3(0, -1, 1), -1);
    }

    // Тесты для функции Max3()
    TEST_METHOD(Max3_AllEqual) {
        Assert::AreEqual(Max3(7, 7, 7), 7);
        Assert::AreEqual(Max3(0, 0, 0), 0);
        Assert::AreEqual(Max3(-4, -4, -4), -4);
    }

    TEST_METHOD(Max3_FirstIsMax) {
        Assert::AreEqual(Max3(3, 2, 1), 3);
        Assert::AreEqual(Max3(5, 0, -5), 5);
        Assert::AreEqual(Max3(30, 20, 10), 30);
    }

    TEST_METHOD(Max3_SecondIsMax) {
        Assert::AreEqual(Max3(1, 3, 2), 3);
        Assert::AreEqual(Max3(-5, 0, 5), 5);
        Assert::AreEqual(Max3(10, 30, 20), 30);
    }

    TEST_METHOD(Max3_ThirdIsMax) {

```

```

        Assert::AreEqual(Max3(1, 2, 3), 3);
        Assert::AreEqual(Max3(-5, 0, 5), 5);
        Assert::AreEqual(Max3(10, 20, 30), 30);
    }

TEST_METHOD(Max3_WithNegativeNumbers) {
    Assert::AreEqual(Max3(-3, -2, -1), -1);
    Assert::AreEqual(Max3(-1, -5, -10), -1);
    Assert::AreEqual(Max3(-1, 0, 1), 1);
}

TEST_METHOD(Max3_EdgeCases) {
    Assert::AreEqual(Max3(INT_MAX, 0, -1), INT_MAX);
    Assert::AreEqual(Max3(INT_MIN, INT_MIN + 1, INT_MIN + 2), INT_MIN + 2);
}

TEST_METHOD(Min3_EdgeCases) {
    Assert::AreEqual(Min3(INT_MAX, INT_MAX - 1, INT_MAX - 2), INT_MAX - 2);
    Assert::AreEqual(Min3(INT_MIN, 0, 1), INT_MIN);
}
};

TEST_CLASS(FractionDivisionTests)
{
public:
    TEST_METHOD(DivideNormalFractions)
    {
        Fraction first(3, 5);
        Fraction second(6, 7);

        Fraction result = FractionDivision(first, second);

        Assert::AreEqual(result.getNumerator(), 7);
        Assert::AreEqual(result.getDenominator(), 10);
    }

    TEST_METHOD(DivideAndSimplifyResult)
    {
        Fraction first(4, 9);
        Fraction second(2, 3);

        Fraction result = FractionDivision(first, second);

        Assert::AreEqual(result.getNumerator(), 2);
        Assert::AreEqual(result.getDenominator(), 3);
    }

    TEST_METHOD(DivideByOne)
    {
        Fraction first(5, 8);
        Fraction second(1, 1);

        Fraction result = FractionDivision(first, second);

        Assert::AreEqual(result.getNumerator(), 5);
        Assert::AreEqual(result.getDenominator(), 8);
    }

    TEST_METHOD(DivideByFractionToGetReciprocal)
    {
        Fraction first(1, 1);
        Fraction second(3, 4);

        Fraction result = FractionDivision(first, second);

        Assert::AreEqual(result.getNumerator(), 4);
    }
};

```

```

        Assert::AreEqual(result.getDenominator(), 3);
    }

TEST_METHOD(DivideFractionByZero) {
    Fraction f1(1, 2);
    Fraction zero(0, 1);

    try {
        FractionDivision(f1, zero);
        assert(false && "Ожидаемое исключение не было вызвано");
    }
    catch (const runtime_error& e) {
        assert(string(e.what()) == "Ошибка: деление на ноль");
    }
    catch (...) {
        assert(false && "Ожидаемое исключение не было вызвано");
    }

    try {
        Fraction invalid(1, 0);
        assert(false && "Ожидаемое исключение не было вызвано");
    }
    catch (const invalid_argument& e) {
        assert(string(e.what()) == "Знаменатель не может быть равен нулю");
    }
    catch (...) {
        assert(false && "Ожидаемое исключение не было вызвано");
    }
}

TEST_METHOD(DivideNegativeFractions)
{
    Fraction first(-1, 2);
    Fraction second(3, -4);

    Fraction result = FractionDivision(first, second);

    Assert::AreEqual(result.getNumerator(), 2);
    Assert::AreEqual(result.getDenominator(), 3);
}

TEST_METHOD(DivideLargeNumbers)
{
    Fraction first(1000, 999);
    Fraction second(100, 999);

    Fraction result = FractionDivision(first, second);

    Assert::AreEqual(result.getNumerator(), 10);
    Assert::AreEqual(result.getDenominator(), 1);
}

TEST_METHOD(ResultIsIrreducible)
{
    Fraction first(7, 11);
    Fraction second(13, 17);

    Fraction result = FractionDivision(first, second);

    Assert::AreEqual(result.getNumerator(), 7 * 17);
    Assert::AreEqual(result.getDenominator(), 11 * 13);
}
};
}
```

## Результат:

The screenshot shows the Test Explorer window from a .NET development environment. The results of a test run are displayed, indicating 30 tests were run, all of which passed. The results are organized into groups: Tests (30), FractionDivisionTests (30), SimpleNumberTest (10), and SimpleTask (12). Each test case is listed with its name and a duration of less than 1 ms. The interface includes a toolbar at the top with various icons for navigating and filtering tests, and a summary pane on the right providing a high-level overview of the test outcomes.

Test	Duration	Traits	Error Message
Tests (30)	< 1 ms		
FractionDivisionTests (30)	< 1 ms		
FractionDivisionTests (8)	< 1 ms		
DivideAndSimplifyResult	< 1 ms		
DivideByFractionToGetReciprocal	< 1 ms		
DivideByOne	< 1 ms		
DivideFractionByZero	< 1 ms		
DivideLargeNumbers	< 1 ms		
DivideNegativeFractions	< 1 ms		
DivideNormalFractions	< 1 ms		
ResultIsIrreducible	< 1 ms		
SimpleNumberTest (10)	< 1 ms		
IsPrime_ForLargeNonPrimes	< 1 ms		
IsPrime_ForLargePrimes	< 1 ms		
IsPrime_ForNegativeNumbers	< 1 ms		
IsPrime_ForSmallNonPrimes	< 1 ms		
IsPrime_ForSmallPrimes	< 1 ms		
IsPrime_ForZeroAndOne	< 1 ms		
NextPrime_ForNegativeNumbers	< 1 ms		
NextPrime_ForNonPrimes	< 1 ms		
NextPrime_ForPrimes	< 1 ms		
NextPrime_ForZeroAndOne	< 1 ms		
SimpleTask (12)	< 1 ms		
Max3_AllEqual	< 1 ms		
Max3_EdgeCases	< 1 ms		
Max3_FirstIsMax	< 1 ms		
Max3_SecondIsMax	< 1 ms		
Max3_ThirdIsMax	< 1 ms		
Max3_WithNegativeNumbers	< 1 ms		
Min3_AllEqual	< 1 ms		
Min3_EdgeCases	< 1 ms		
Min3_FirstIsMin	< 1 ms		
Min3_SecondIsMin	< 1 ms		
Min3_ThirdIsMin	< 1 ms		
Min3_WithNegativeNumbers	< 1 ms		

Search (Ctrl+I) ▾ 2 Warnings ✘ 0 Errors

Run | Debug

Group Summary

Tests  
Tests in group : 30

Outcomes  
30 Passed