Лабораторная работа № 4. Модульное тестирование.

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

1. Даны две дроби A/B и C/D (А, В, С, D — натуральные числа). Составить функции, возвращающие числитель и знаменатель дроби – результата умножения дроби на дробь. Ответ должен быть несократимой дробью.

В итоге был написан целый класс Fraction, который реализует множество функционала. Такое как сложение, умножение, красивый вывод и сокращение.

Так же чтобы не морочиться(с cmake и .h), я организовал тестирование с помощью google tests, прямо в этом файле. Код программы:

#include<iostream>

#include<string>

#include <gtest/gtest.h>

using namespace std;

class Fraction{

private:

void reduce(){

int max = denominator > numerator? denominator: numerator;

if (max == 1)

return;

for (int i = 2; i < max / 2 + 1; i++){

if (numerator % i == 0 && denominator % i == 0){

numerator /= i;

denominator /= i;

reduce();

}

}

if (numerator % max == 0 && denominator % max == 0){

numerator /= max;

denominator /= max;

reduce();

}

}

public:

// Могут быть приватными, но для облегчения тестов лежат здесь.

int numerator; // числитель

int denominator; // знаменатель

Fraction(int numerator, int denominator = 1){

this->numerator = numerator;

this->denominator = denominator;

reduce();

}

friend ostream& operator << (ostream& os, Fraction& fr){

string num\_str = to\_string(fr.numerator);

string den\_str = to\_string(fr.denominator);

// os << " " << num\_str << " " << endl;

if (den\_str.size() > num\_str.size()){

for (int i = 0; i < den\_str.size() / 2 + 1 - (num\_str.size() % 2 == 0 ? num\_str.size() / 2: num\_str.size() / 2 + 1); i++)

os << " ";

os << num\_str << endl;

os << "-";

for (auto ch: den\_str)

os << "-";

os << "-" << " (" << (fr.numerator / 1.0f) / fr.denominator << ")" << endl;

os << " " << den\_str << endl;

} else {

os << " " << num\_str << endl;

os << "-";

for (auto ch: num\_str)

os << "-";

os << "-" << " (" << (fr.numerator / 1.0f) / fr.denominator << ")" << endl;

for (int i = 0; i < num\_str.size() / 2 + 1 - den\_str.size() / 2; i++)

os << " ";

os << den\_str << endl;

}

os << '\r' << '\r';

return os;

}

Fraction& operator \*=(Fraction &frac1){

this->numerator \*= frac1.numerator;

this->denominator \*= frac1.denominator;

reduce();

return \*this;

}

Fraction& operator \*=(int num){

Fraction fr(num, 1);

\*this \*= fr;

return \*this;

}

Fraction operator \*(Fraction &frac1){

return Fraction(numerator \* frac1.numerator, denominator \*= frac1.denominator) ;

}

Fraction operator \*(int num){

return Fraction(numerator \* num, denominator) ;

}

Fraction& operator +=(Fraction &frac1){

numerator \*= frac1.denominator;

frac1.numerator \*= denominator;

this->numerator += frac1.numerator;

this->denominator \*= frac1.denominator;

reduce();

return \*this;

}

Fraction operator +(Fraction &frac1){

Fraction fr(numerator, denominator);

fr += frac1;

return fr;

}

};

// int main(){

// int a, b;

// cout << "Write numerator" << endl;

// cin >> a;

// cout << "Write denominator" << endl;

// cin >> b;

// Fraction fr(a, b);

// Fraction fr2 = fr;

// cout << fr2;

// fr2 += fr;

// cout << fr2;

// Fraction fr3 = fr2 + fr2;

// cout << fr3;

// string str = "123";

// }

TEST(FractionAddTest, ReduceCheckToOne) {

Fraction fr1(2, 3);

Fraction fr2(1, 3);

Fraction fr3 = fr1 + fr2;

EXPECT\_EQ(fr3.numerator, 1);

EXPECT\_EQ(fr3.denominator, 1);

}

TEST(FractionAddTest, VariousDenominator) {

Fraction fr1(2, 3);

Fraction fr2(1, 9);

Fraction fr3 = fr1 + fr2;

EXPECT\_EQ(fr3.numerator, 7);

EXPECT\_EQ(fr3.denominator, 9);

}

TEST(FractionAddTest, VariousDenominatorReduce) {

Fraction fr1(2, 4);

Fraction fr2(2, 5);

Fraction fr3 = fr1 + fr2;

EXPECT\_EQ(fr3.numerator, 9);

EXPECT\_EQ(fr3.denominator, 10);

}

TEST(FractionAddTest, ImproperFractionResult){

Fraction fr1(2, 3);

Fraction fr2(2, 3);

Fraction fr3 = fr1 + fr2;

EXPECT\_EQ(fr3.numerator, 4);

EXPECT\_EQ(fr3.denominator, 3);

}

TEST(DisplayTest, OneToOneBadTest){

Fraction fr(1, 1);

stringstream ss;

ss << fr;

EXPECT\_STREQ(" 1\n--- (1)\n 1\n", ss.str().c\_str());

}

TEST(DisplayTest, OneToOneGoodTest){

Fraction fr(1, 1);

stringstream ss;

ss << fr;

EXPECT\_STREQ(" 1\n--- (1)\n 1\n\r\r", ss.str().c\_str());

}

TEST(DisplayTest, IrrationalFraction){

Fraction fr(1, 3);

stringstream ss;

ss << fr;

EXPECT\_STREQ(" 1\n--- (0.333333)\n 3\n\r\r", ss.str().c\_str());

}

TEST(FractionMultiplyTest, Multiply){

Fraction fr1(2, 3);

Fraction fr2(2, 3);

Fraction fr3 = fr1 \* fr2;

EXPECT\_EQ(fr3.numerator, 4);

EXPECT\_EQ(fr3.denominator, 9);

}

TEST(FractionMultiplyTest, MultiplyAndReduce){

Fraction fr1(2, 3);

Fraction fr2(1, 2);

Fraction fr3 = fr1 \* fr2;

EXPECT\_EQ(fr3.numerator, 1);

EXPECT\_EQ(fr3.denominator, 3);

}

TEST(FractionMultipieTest, MultiplyWithNumber){

Fraction fr1(2, 3);

Fraction fr3 = fr1 \* 3;

EXPECT\_EQ(fr3.numerator, 2);

EXPECT\_EQ(fr3.denominator, 1);

}

После сборки и запуска мы получим такой результат:

Running main() from C:\Users\user\Desktop\UNIT\_tests\\_deps\googletest-src\googletest\src\gtest\_main.cc

[==========] Running 10 tests from 4 test suites.

[----------] Global test environment set-up.

[----------] 4 tests from FractionAddTest

[ RUN ] FractionAddTest.ReduceCheckToOne

[ OK ] FractionAddTest.ReduceCheckToOne (0 ms)

[ RUN ] FractionAddTest.VariousDenominator

[ OK ] FractionAddTest.VariousDenominator (0 ms)

[ RUN ] FractionAddTest.VariousDenominatorReduce

[ OK ] FractionAddTest.VariousDenominatorReduce (0 ms)

[ RUN ] FractionAddTest.ImproperFractionResult

[ OK ] FractionAddTest.ImproperFractionResult (0 ms)

[----------] 4 tests from FractionAddTest (15 ms total)

[----------] 3 tests from DisplayTest

[ RUN ] DisplayTest.OneToOneBadTest

C:\Users\user\Desktop\UNIT\_tests\1.cpp:153: Failure

Expected equality of these values:

" 1\n--- (1)\n 1\n"

ss.str().c\_str()

Which is: " 1\n--- (1)\n 1\n\r\r"

With diff:

@@ -1,3 +1,4 @@

1

--- (1)

- 1\n

+ 1

+\r\r

[ FAILED ] DisplayTest.OneToOneBadTest (2 ms)

[ RUN ] DisplayTest.OneToOneGoodTest

[ OK ] DisplayTest.OneToOneGoodTest (0 ms)

[ RUN ] DisplayTest.IrrationalFraction

[ OK ] DisplayTest.IrrationalFraction (0 ms)

[----------] 3 tests from DisplayTest (14 ms total)

[----------] 2 tests from FractionMultiplyTest

[ RUN ] FractionMultiplyTest.Multiply

[ OK ] FractionMultiplyTest.Multiply (0 ms)

[ RUN ] FractionMultiplyTest.MultiplyAndReduce

[ OK ] FractionMultiplyTest.MultiplyAndReduce (0 ms)

[----------] 2 tests from FractionMultiplyTest (13 ms total)

[----------] 1 test from FractionMultipieTest

[ RUN ] FractionMultipieTest.MultiplyWithNumber

[ OK ] FractionMultipieTest.MultiplyWithNumber (0 ms)

[----------] 1 test from FractionMultipieTest (4 ms total)

[----------] Global test environment tear-down

[==========] 10 tests from 4 test suites ran. (67 ms total)

[ PASSED ] 9 tests.

[ FAILED ] 1 test, listed below:

[ FAILED ] DisplayTest.OneToOneBadTest

1 FAILED TEST

Как мы можем заметить, "плохой" тест функции отображения, выдал что обннаружил ошибку и в чём именно различия. Эти различия мы исправили в "хорошем тесте"