Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

**Лабораторная работа №1**

**«Абстрактный тип данных простая дробь»**

Выполнил: студент IV курса

ИВТ, гр. ИП-713

Михеев Н.А.

Проверил: ассистент кафедры

ПМиК

Агалаков А.А.

Новосибирск, 2020 г.

**Цель**

Сформировать практические навыки реализации абстрактных типов данных в соответствии с заданной спецификацией с помощью классов С++ и их модульного тестирования.

**Задание**

1. Реализовать абстрактный тип данных «простая дробь», используя класс

С++ в соответствии с приведенной ниже спецификацией.

2. Протестировать каждую операцию, определенную на типе данных,

используя средства модульного тестирования.

3. Если необходимо, предусмотрите возбуждение исключительных

ситуаций.

**Реализация**

В ходе выполнения задания был реализован класс «простая дробь», который хранит в себе два поля – num и den – числитель и знаменатель соответственно. Были реализованы соответствующие конструкторы, методы класса в соответствии с заданием, перегрузки операторов сравнения и знака минус.

Далее о каждом из объектов класса подробнее:

TFrac::TFrac() – базовый конструктор класса, формирует начальную дробь 0/1.

TFrac(int, int) – конструктор дроби из целых чисел.

TFrac(std::string) – конструктор дроби из строки.

TFrac copy() – метод копирования одной дроби в другую, вызывается как a.copy(b).

TFrac add(TFrac&) – метод суммирования двух дробей.

TFrac multiply(TFrac&) – метод умножения двух дробей.

TFrac subtract(TFrac&) – метод вычитания двух дробей.

TFrac divide(TFrac&) - метод деления одной дроби на другую.

TFrac square() – метод возведения дроби во вторую степень.

TFrac reverse() – метод переворачивания дроби.

TFrac operator-() – перегрузка оператора «-».

bool operator==(TFrac) -перегрузка булейвого оператора «==», необходимого для сравнения двух дробей.

bool operator>(TFrac) – перегрузка булейвого оператора «>», необходимого для сравнения двух дробей.

int getNumeratorNum() – метод, возвращающий числовое значение числителя дроби.

int getDenominatorNum() - метод, возвращающий числовое значение знаменателя дроби.

std::string getNumeratorString() - метод, возвращающий строковое значение числителя дроби.

std::string getDenominatorString() - метод, возвращающий строковое значение знаменателя дроби.

std::string getFractionString() – метод, возвращающий строковое значение дроби.

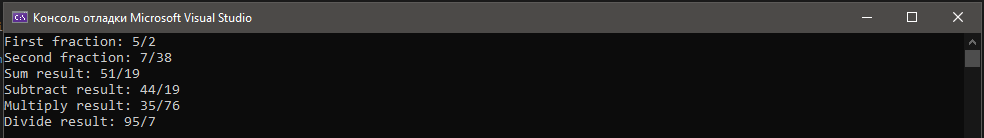


Рис.1 – Результат проверки работоспособности программы

Так же были реализованы тесты всех методов и перегрузок.

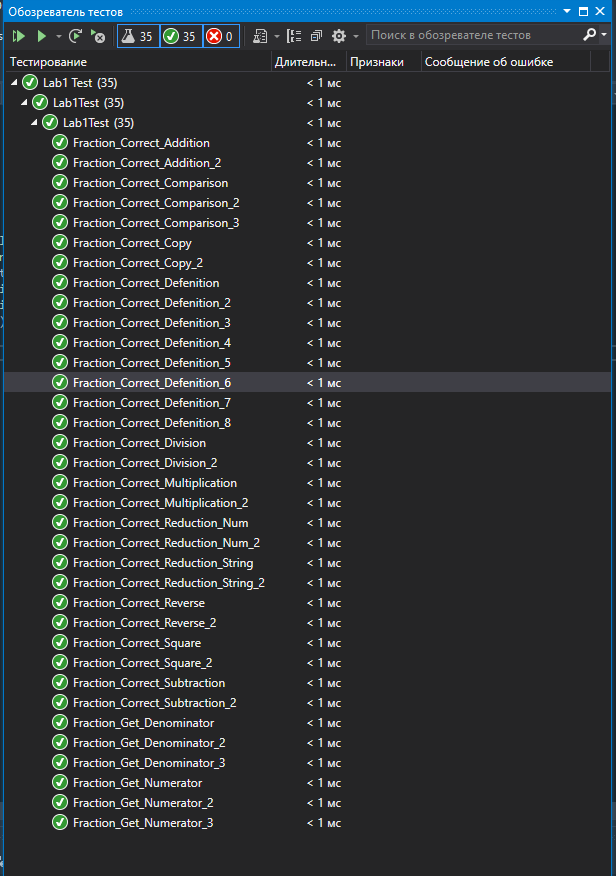


Рис.2 – демонстрация результатов проведенного тестирования

**Вывод**

В результате работы были сформированы практические навыки реализации абстрактных типов данных, реализован такой тип данных «простая дробь», разработаны соответствующие методы и перегрузки в соответствии с приведенной в задании спецификацией. Написаны тесты для каждой операции, использую средства модульного тестирования.

**Листинг программы:**

**TFrac.H:**

#pragma once

#include <string>

#include <numeric>

#include <algorithm>

#include <iostream>

#include <stdexcept>

#include "pch.h"

class TFrac

{

private:

int num;

int den;

public:

TFrac();

TFrac(int, int);

TFrac(std::string);

TFrac copy();

TFrac add(TFrac&);

TFrac multiply(TFrac&);

TFrac subtract(TFrac&);

TFrac divide(TFrac&);

TFrac square();

TFrac reverse();

TFrac operator-();

bool operator==(TFrac);

bool operator>(TFrac);

int getNumeratorNum();

int getDenominatorNum();

std::string getNumeratorString();

std::string getDenominatorString();

std::string getFractionString();

};

**TFrac.cpp:**

#include "pch.h"

#include "TFrac.h"

TFrac::TFrac()

{

num = 0;

den = 1;

}

TFrac::TFrac(int a, int b) : num(a), den(b)

{

if (den == 0)

{

std::cerr << "Denominator can't be 0!" << std::endl;

num = 0;

den = 1;

return;

}

int gcd = std::gcd(num, den);

num /= gcd;

den /= gcd;

if (den < 0)

{

num \*= -1;

den \*= -1;

}

}

TFrac::TFrac(std::string frac)

{

std::string s\_num, s\_den;

size\_t slash = frac.find('/');

s\_num = frac.substr(0, slash);

s\_den = frac.substr(slash + 1, frac.length());

num = std::stoi(s\_num);

den = std::stoi(s\_den);

if (den == 0)

{

std::cerr << "Denominator can't be 0!" << std::endl;

num = 0;

den = 1;

return;

}

int gcd = std::gcd(num, den);

num /= gcd;

den /= gcd;

if (den < 0)

{

num \*= -1;

den \*= -1;

}

}

TFrac TFrac::copy()

{

return {num, den};

}

TFrac TFrac::add(TFrac& frac)

{

TFrac res;

res.num = num \* frac.den + frac.num \* den;

res.den = den \* frac.den;

int gcd = std::gcd(res.num, res.den);

res.num /= gcd;

res.den /= gcd;

if (res.den < 0)

{

res.num \*= -1;

res.den \*= -1;

}

return res;

}

TFrac TFrac::multiply(TFrac& frac)

{

TFrac res;

res.num = num \* frac.num;

res.den = den \* frac.den;

int gcd = std::gcd(res.num, res.den);

res.num /= gcd;

res.den /= gcd;

if (res.den < 0)

{

res.num \*= -1;

res.den \*= -1;

}

return res;

}

TFrac TFrac::subtract(TFrac& frac)

{

TFrac res;

res.num = num \* frac.den - frac.num \* den;

res.den = frac.den \* den;

int gcd = std::gcd(res.num, res.den);

res.num /= gcd;

res.den /= gcd;

if (res.den < 0)

{

res.num \*= -1;

res.den \*= -1;

}

return res;

}

TFrac TFrac::divide(TFrac& frac)

{

if (frac.num == 0 || den == 0)

{

std::cout << "No divinition by 0!" << std::endl;

return { 0, 1 };

}

TFrac res;

res.num = num \* frac.den;

res.den = den \* frac.num;

int gcd = std::gcd(res.num, res.den);

res.num /= gcd;

res.den /= gcd;

if (res.den < 0)

{

res.num \*= -1;

res.den \*= -1;

}

return res;

}

TFrac TFrac::square()

{

return { num \* num, den \* den };

}

TFrac TFrac::reverse()

{

return { den, num };

}

TFrac TFrac::operator-()

{

return { -num, den };

}

bool TFrac::operator==(TFrac frac)

{

return num == frac.num && den == frac.den;

}

bool TFrac::operator>(TFrac frac)

{

return num \* frac.den > frac.num \* den;

}

int TFrac::getNumeratorNum()

{

return num;

}

int TFrac::getDenominatorNum()

{

return den;

}

std::string TFrac::getNumeratorString()

{

return std::to\_string(num);

}

std::string TFrac::getDenominatorString()

{

return std::to\_string(den);

}

std::string TFrac::getFractionString()

{

return std::to\_string(num) + '/' + std::to\_string(den);

}

**Lab1 Test.cpp:**

#include "pch.h"

#include "CppUnitTest.h"

#include "../Lab1/TFrac.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace Lab1Test

{

TEST\_CLASS(Lab1Test)

{

public:

TEST\_METHOD(Fraction\_Correct\_Defenition)

{

std::string expect = "2/5";

TFrac test(2, 5);

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Defenition\_2)

{

std::string expect = "2/5";

TFrac test("2/5");

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Defenition\_3)

{

std::string expect = "0/1";

TFrac test(1, 0);

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Defenition\_4)

{

std::string expect = "4/5";

TFrac test(000004, 00005);

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Defenition\_5)

{

std::string expect = "-1/2";

TFrac test(-50, 100);

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Defenition\_6)

{

std::string expect = "-4/5";

TFrac test(4, -5);

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Defenition\_7)

{

std::string expect = "1/4";

TFrac test(-8, -32);

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Defenition\_8)

{

std::string expect = "0/1";

TFrac test("5/0");

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Reduction\_Num)

{

std::string expect = "1/3";

TFrac test(2, 6);

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Reduction\_Num\_2)

{

std::string expect = "-5/1";

TFrac test(25, -5);

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Reduction\_String)

{

std::string expect = "3/7";

TFrac test("9/21");

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Reduction\_String\_2)

{

std::string expect = "-15/1";

TFrac test("30/-2");

Assert::AreEqual(expect, test.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Addition)

{

std::string expect = "4/5";

TFrac test("1/5");

TFrac test2("3/5");

Assert::AreEqual(expect, test.add(test2).getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Addition\_2)

{

std::string expect = "-1/4";

TFrac test("1/2");

TFrac test2(3, -4);

Assert::AreEqual(expect, test.add(test2).getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Multiplication)

{

std::string expect = "16/25";

TFrac test("1/5");

TFrac test2("16/5");

Assert::AreEqual(expect, test.multiply(test2).getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Multiplication\_2)

{

std::string expect = "-2/1";

TFrac test("-1/6");

TFrac test2("12/1");

Assert::AreEqual(expect, test.multiply(test2).getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Subtraction)

{

std::string expect = "1/5";

TFrac test("6/10");

TFrac test2("2/5");

Assert::AreEqual(expect, test.subtract(test2).getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Subtraction\_2)

{

std::string expect = "1/2";

TFrac test("3/5");

TFrac test2("1/10");

Assert::AreEqual(expect, test.subtract(test2).getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Division)

{

std::string expect = "25/9";

TFrac test("5/3");

TFrac test2("3/5");

Assert::AreEqual(expect, test.divide(test2).getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Division\_2)

{

std::string expect = "-25/9";

TFrac test("5/3");

TFrac test2("3/-5");

Assert::AreEqual(expect, test.divide(test2).getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Reverse)

{

std::string expect = "7/3";

TFrac test(3, 7);

Assert::AreEqual(expect, test.reverse().getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Reverse\_2)

{

std::string expect = "1/2";

TFrac test(10, 5);

Assert::AreEqual(expect, test.reverse().getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Square)

{

std::string expect = "1/4";

TFrac test(5, 10);

Assert::AreEqual(expect, test.square().getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Square\_2)

{

std::string expect = "9/49";

TFrac test(3, 7);

Assert::AreEqual(expect, test.square().getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Comparison)

{

TFrac test("3/5");

TFrac test2("3/5");

Assert::IsTrue(test == test2);

}

TEST\_METHOD(Fraction\_Correct\_Comparison\_2)

{

TFrac test("3/5");

TFrac test2("1/100");

Assert::IsTrue(test > test2);

}

TEST\_METHOD(Fraction\_Correct\_Comparison\_3)

{

TFrac test("3/5");

TFrac test2("6/10");

Assert::IsTrue(test == test2);

}

TEST\_METHOD(Fraction\_Correct\_Copy)

{

TFrac test(8, 11);

TFrac test2 = test.copy();

Assert::AreEqual(test.getFractionString(), test2.getFractionString());

}

TEST\_METHOD(Fraction\_Correct\_Copy\_2)

{

TFrac test("15/17");

TFrac test2 = test.copy();

Assert::AreEqual(test.getFractionString(), test2.getFractionString());

}

TEST\_METHOD(Fraction\_Get\_Numerator)

{

TFrac test(8, 11);

std::string expect = "8";

Assert::AreEqual(expect, test.getNumeratorString());

}

TEST\_METHOD(Fraction\_Get\_Numerator\_2)

{

TFrac test(8, 11);

int expect = 8;

Assert::AreEqual(expect, test.getNumeratorNum());

}

TEST\_METHOD(Fraction\_Get\_Numerator\_3)

{

TFrac test(-1, 3);

int expect = -1;

Assert::AreEqual(expect, test.getNumeratorNum());

}

TEST\_METHOD(Fraction\_Get\_Denominator)

{

TFrac test(8, 11);

std::string expect = "11";

Assert::AreEqual(expect, test.getDenominatorString());

}

TEST\_METHOD(Fraction\_Get\_Denominator\_2)

{

TFrac test(8, 11);

int expect = 11;

Assert::AreEqual(expect, test.getDenominatorNum());

}

TEST\_METHOD(Fraction\_Get\_Denominator\_3)

{

TFrac test(2, -13);

int expect = 13;

Assert::AreEqual(expect, test.getDenominatorNum());

}

};

}

**Lab1.cpp:**

#include <iostream>

#include "TFrac.h"

int main()

{

TFrac test("5/2");

TFrac test2("14/76");

std::cout << "First fraction: " << test.getFractionString() << std::endl;

std::cout << "Second fraction: " << test2.getFractionString() << std::endl;

std::cout << "Sum result: " << test.add(test2).getFractionString() << std::endl;

std::cout << "Subtract result: " << test.subtract(test2).getFractionString() << std::endl;

std::cout << "Multiply result: " << test.multiply(test2).getFractionString() << std::endl;

std::cout << "Divide result: " << test.divide(test2).getFractionString() << std::endl;

return 0;

}