#include "pch.h"

#include <gtest/gtest.h>

#include <iostream>

#include <assert.h>

#include <sstream>

#include "complex.h"

#include "matrix.h"

#include "polynomial.h"

#include <string>

#include "fraction.h"

namespace Fraction {

class FractionTest : public ::testing::Test {

protected:

};

TEST(Constructor, DefaultConstructor) {

fraction<int> f;

EXPECT\_EQ(f.getNumerator(), 0);

EXPECT\_EQ(f.getDenominator(), 1);

}

// Тест конструктора с одним аргументом

TEST(Constructor, SingleArgConstructor) {

fraction<int> f(5);

EXPECT\_EQ(f.getNumerator(), 5);

EXPECT\_EQ(f.getDenominator(), 1);

}

// Тест конструктора с двумя аргументами

TEST(Constructor, TwoArgConstructor) {

fraction<int> f(2, 4);

EXPECT\_EQ(f.getNumerator(), 1);

EXPECT\_EQ(f.getDenominator(), 2);

}

// Тест оператора сложения

TEST(Operator, AdditionOperator) {

fraction<int> f1(1, 2);

fraction<int> f2(1, 3);

fraction<int> result = f1 + f2;

EXPECT\_EQ(result.getNumerator(), 5);

EXPECT\_EQ(result.getDenominator(), 6);

}

// Тест оператора умножения

TEST(Operator, MultiplicationOperator) {

fraction<int> f1(2, 3);

fraction<int> f2(3, 4);

fraction<int> result = f1 \* f2;

EXPECT\_EQ(result.getNumerator(), 1);

EXPECT\_EQ(result.getDenominator(), 2);

}

// Тест оператора деления

TEST(Operator, DivisionOperator) {

fraction<int> f1(2, 3);

fraction<int> f2(3, 4);

fraction<int> result = f1 / f2;

EXPECT\_EQ(result.getNumerator(), 8);

EXPECT\_EQ(result.getDenominator(), 9);

}

// Тест оператора вычитания

TEST(Operator, SubtractionOperator) {

fraction<int> f1(5, 6);

fraction<int> f2(1, 6);

fraction<int> result = f1 - f2;

EXPECT\_EQ(result.getNumerator(), 2);

EXPECT\_EQ(result.getDenominator(), 3);

}

// Тест метода сокращения дроби

TEST(Metode, SimplifyFraction) {

fraction<int> f(8, 12);

f.reduce();

EXPECT\_EQ(f.getNumerator(), 2);

EXPECT\_EQ(f.getDenominator(), 3);

}

// Тест метода сравнения двух дробей

TEST(Equality\_method, CompareFractions) {

fraction<int> f1(1, 2);

fraction<int> f2(2, 4);

bool result = f1 == f2;

EXPECT\_TRUE(result);

}

}

namespace Polynomial {

TEST(PolynomialTest, AdditionOperatorTest) {

polynomial<int> p1; p1.newsize(2);

p1[0] = 1;

p1[1] = 2;

polynomial<int> p2; p2.newsize(2);

p2[0] = 3;

p2[1] = 4;

polynomial<int> sum = p1 + p2;

EXPECT\_EQ(sum[0], 4);

EXPECT\_EQ(sum[1], 6);

}

TEST(PolynomialTest, InputOperatorTest) {

std::stringstream ss("3\n1 2 3\n");

polynomial<int> p;

ss >> p;

EXPECT\_EQ(p.get\_deg(), 3);

EXPECT\_EQ(p[0], 1);

EXPECT\_EQ(p[1], 2);

EXPECT\_EQ(p[2], 3);

}

TEST(OutputOperatorTest, Polynomial\_OutputOperatorTest\_FULL) {

polynomial<int> p(3);

p.newsize(3);

p.output\_mode\_set(output\_mode::FULL);

p[0] = 1;

p[1] = 2;

p[2] = 3;

std::stringstream ss;

ss << p;

std::string expectedOutput = "Degree: 3, Coefficients: 1 + 2x + 3x^2";

EXPECT\_EQ(ss.str(), expectedOutput);

}

TEST(OutputOperatorTest, Polynomial\_OutputOperatorTest\_ABBREVIATED) {

polynomial<int> p(3);

p.newsize(3);

p.output\_mode\_set(1);

p[0] = 1;

p[1] = 2;

p[2] = 3;

std::stringstream ss;

ss << p;

std::string expectedOutput = "1+2x+3x^2";

EXPECT\_EQ(ss.str(), expectedOutput);

}

TEST(OutputOperatorTest,Polynomial\_OutputOperatorTest\_SHORT) {

polynomial<int> p(3);

p.newsize(3);

p.output\_mode\_set(2);

p[0] = 1;

p[1] = 2;

p[2] = 3;

std::stringstream ss;

ss << p;

std::string expectedOutput = "1 2x 3x^2";

EXPECT\_EQ(ss.str(), expectedOutput);

}

TEST(PolynomialTest, MultiplyTest) {

polynomial<int> p1;

p1.newsize(3);

p1.output\_mode\_set(0);

p1[0] = 2;

p1[1] = 3;

p1[2] = 1;

polynomial<int> p2;

p2.newsize(3);

p2[0] = 2;

p2[1] = 3;

p2[2] = 1;

polynomial<int> product; product = p1 \* p2;

std::stringstream ss;

ss << product;

std::string expectedOutput1 = "Degree: 5, Coefficients: 4 + 12x + 13x^2 + 6x^3 + 1x^4", expectedOutput2 = "4 12x 13x^2 6x^3 1x^4", expectedOutput3 = "4+12x+13x^2+6x^3+1x^4";

EXPECT\_TRUE(ss.str()== expectedOutput1|| ss.str() == expectedOutput2 || ss.str() == expectedOutput3);

}

TEST(PolynomialTest, DivideTest) {

polynomial<double> p1;

p1.newsize(3);

p1[0] = 4;

p1[1] = 8;

p1[2] = 12;

polynomial<double> p2; p2.newsize(1);

p2[0] = 2;

polynomial<double> quotient = p1 / p2;

EXPECT\_EQ(quotient.get\_deg(), 3);

EXPECT\_EQ(quotient[0], 2);

EXPECT\_EQ(quotient[1], 4);

EXPECT\_EQ(quotient[2], 6);

}

TEST(PolynomialTest, ModulusTest) {

polynomial<double> p1;

p1.newsize(4);

p1[0] = 5;

p1[1] = 10;

p1[2] = 15;

p1[3] = 20;

polynomial<double> p2;

p2.newsize(2);

p2[0] = 3;

p2[1] = 6;

polynomial<double> remainder = p1 % p2;

EXPECT\_EQ(remainder.get\_deg(), 1);

EXPECT\_EQ(remainder[0], 1.25);

}

}

namespace Matrix {

TEST(Constructor, ConstructorWithSize)

{

matrix<int> m(3, 3);

EXPECT\_EQ(m.getcol(), 3);

EXPECT\_EQ(m.getrow(), 3);

}

TEST(operators, MatrixMultiplication\_operator1)

{

matrix<int> m1(3, 3);

m1[0][0] = 1; m1[0][1] = 2; m1[0][2] = 3;

m1[1][0] = 4; m1[1][1] = 5; m1[1][2] = 6;

m1[2][0] = 7; m1[2][1] = 8; m1[2][2] = 9;

matrix<int> m2(3, 3);

m2[0][0] = 9; m2[0][1] = 8; m2[0][2] = 7;

m2[1][0] = 6; m2[1][1] = 5; m2[1][2] = 4;

m2[2][0] = 3; m2[2][1] = 2; m2[2][2] = 1;

matrix<int> result = m1 \* m2;

EXPECT\_EQ(result[0][0], 30); EXPECT\_EQ(result[0][1], 24); EXPECT\_EQ(result[0][2], 18);

EXPECT\_EQ(result[1][0], 84); EXPECT\_EQ(result[1][1], 69); EXPECT\_EQ(result[1][2], 54);

EXPECT\_EQ(result[2][0], 138); EXPECT\_EQ(result[2][1], 114); EXPECT\_EQ(result[2][2], 90);

}

TEST(operators, MatrixMultiplication\_operator2)

{

matrix<int> m1(3, 3);

m1[0][0] = 1; m1[0][1] = 2; m1[0][2] = 3;

m1[1][0] = 4; m1[1][1] = 5; m1[1][2] = 6;

m1[2][0] = 7; m1[2][1] = 8; m1[2][2] = 9;

matrix<int> m2(3, 2);

m2[0][0] = 9; m2[0][1] = 8;

m2[1][0] = 6; m2[1][1] = 5;

m2[2][0] = 3; m2[2][1] = 2;

matrix<int> result = m1 \* m2;

std::stringstream method\_ans;

method\_ans << result;

EXPECT\_EQ(method\_ans.str(), "sizex:3sizey:2\n[0][0] = \t30\t | [0][1] = \t24\t | \n[1][0] = \t84\t | [1][1] = \t69\t | \n[2][0] = \t138\t | [2][1] = \t114\t | \n");

EXPECT\_EQ(result[0][0], 30); EXPECT\_EQ(result[0][1], 24);

EXPECT\_EQ(result[1][0], 84); EXPECT\_EQ(result[1][1], 69);

EXPECT\_EQ(result[2][0], 138); EXPECT\_EQ(result[2][1], 114);

}

TEST(methods, MatrixTranspose)//there were no problems in the rest of the program

{

matrix<double> m(2, 3);

m[0][0] = 1; m[0][1] = 2; m[0][2] = 3;

m[1][0] = 4; m[1][1] = 5; m[1][2] = 6;

std::cout << m;

matrix<double> transposed = m.transpose();

std::cout << transposed;

EXPECT\_EQ(transposed.getcol(), 3);

EXPECT\_EQ(transposed.getrow(), 2);

EXPECT\_EQ(transposed[0][0], 1);

EXPECT\_EQ(transposed[1][0], 2);

EXPECT\_EQ(transposed[2][0], 3);

EXPECT\_EQ(transposed[0][1], 4);

EXPECT\_EQ(transposed[1][1], 5);

EXPECT\_EQ(transposed[2][1], 6);

}

TEST(methods, determinant) {

std::stringstream ss("2\n2\n2\n1\n1\n1\n1\n2\n1\n2\n1\n1\n2\n1\n3\n1\n1\n4\n1\n2\n3\n4\n1\n1\n");

matrix<fraction<polynomial<int>>> mtrx;

ss >> mtrx;

std::stringstream method\_ans;

method\_ans << mtrx.determinant();

std::string true\_ans\_str1 = "(Degree: 5, Coefficients: 0 + (-2)x + (-1)x^2 + 7x^3 + 4x^4) / (Degree: 1, Coefficients: 1)",

true\_ans\_str2 = "(0+(-2)x+(-1)x^2+7x^3+4x^4) / (1)",

true\_ans\_str3 = "(0 -2x -1x^2 7x^3 4x^4) / (1)";

EXPECT\_TRUE(method\_ans.str() == true\_ans\_str1 || method\_ans.str() == true\_ans\_str2 || method\_ans.str() == true\_ans\_str3);

}

TEST(methods, inverse\_M\_int) {

std::stringstream ss("2\n2\n2\n1\n1\n1\n1\n2\n1\n2\n1\n1\n2\n1\n3\n1\n1\n4\n1\n2\n3\n4\n1\n1\n");

matrix<fraction<polynomial<int>>> mtrx;//there used to be a problem with %, and it only worked with int, now it seems to have disappeared

ss >> mtrx;

std::stringstream method\_ans;

method\_ans << mtrx.inverse\_M();

std::string true\_ans\_str1 = "sizex:2sizey:2\n[0][0] = \t(Degree: 6, Coefficients: 1 + 4x + 8x^2 + 12x^3 + 11x^4 + 4x^5) / (Degree: 7, Coefficients: 0 + (-2)x + (-5)x^2 + 3x^3 + 17x^4 + 15x^5 + 4x^6)\t | [0][1] = \tDegree: 3, Coefficients: (-1) + (-3)x + (-2)x^2 / (Degree: 6, Coefficients: 0 + (-2)x + (-3)x^2 + 6x^3 + 11x^4 + 4x^5)\t | \n[1][0] = \tDegree: 3, Coefficients: (-1) + (-4)x + (-3)x^2 / (Degree: 6, Coefficients: 0 + (-2)x + (-3)x^2 + 6x^3 + 11x^4 + 4x^5)\t | [1][1] = \tDegree: 2, Coefficients: 1 + 1x / (Degree: 5, Coefficients: 0 + (-2)x + (-1)x^2 + 7x^3 + 4x^4)\t | \n",

true\_ans\_str2 = "sizex:2sizey:2\n[0][0] = \t(1 4x 8x^2 12x^3 11x^4 4x^5) / (0 -2x -5x^2 3x^3 17x^4 15x^5 4x^6)\t | [0][1] = \t(-1 -3x -2x^2) / (0 -2x -3x^2 6x^3 11x^4 4x^5)\t | \n[1][0] = \t(-1 -4x -3x^2) / (0 -2x -3x^2 6x^3 11x^4 4x^5)\t | [1][1] = \t(1 1x) / (0 -2x -1x^2 7x^3 4x^4)\t | \n",

true\_ans\_str3="sizex:2sizey:2\n[0][0] = \t1 4x 8x^2 12x^3 11x^4 4x^5 / (0 -2x -5x^2 3x^3 17x^4 15x^5 4x^6)\t | [0][1] = \t-1 -3x -2x^2 / (0 -2x -3x^2 6x^3 11x^4 4x^5)\t | \n[1][0] = \t-1 -4x -3x^2 / (0 -2x -3x^2 6x^3 11x^4 4x^5)\t | [1][1] = \t1 1x / (0 -2x -1x^2 7x^3 4x^4)\t | \n";

EXPECT\_EQ(method\_ans.str(), true\_ans\_str2);

EXPECT\_TRUE(method\_ans.str() == true\_ans\_str1 || method\_ans.str() == true\_ans\_str2|| method\_ans.str() == true\_ans\_str3);

}

TEST(methods, inverse\_M\_float) {

std::stringstream ss("2\n2\n2\n1\n1\n1\n1\n2\n1\n2\n1\n1\n2\n1\n3\n1\n1\n4\n1\n2\n3\n4\n1\n1\n");

matrix<fraction<polynomial<float>>> mtrx;//there used to be a problem with %, and it only worked with int, now it seems to have disappeared

ss >> mtrx;

std::stringstream method\_ans;

method\_ans << mtrx.inverse\_M();

std::string true\_ans\_str1 = "sizex:2sizey:2\n[0][0] = \t(-233827 -935306x -1.87061e+06x^2 -2.80592e+06x^3 -2.57209e+06x^4 -935306x^5) / (0 467653x 1.16913e+06x^2 -701480x^3 -3.97505e+06x^4 -3.5074e+06x^5 -935306x^6)\t | [0][1] = \t(36.125 108.375x 72.2499x^2) / (0 72.2499x 108.375x^2 -216.75x^3 -397.375x^4 -144.5x^5)\t | \n[1][0] = \t(-2.89286 -8.67857x) / (-0 -5.78571x -2.89286x^2 20.25x^3 11.5714x^4)\t | [1][1] = \t(-0.5 -0.5x) / (0 1x 0.5x^2 -3.5x^3 -2x^4)\t | \n";

EXPECT\_EQ(method\_ans.str(), true\_ans\_str1);

}

TEST(methods, inverse\_M\_double) {

std::stringstream ss("2\n2\n2\n1\n1\n1\n1\n2\n1\n2\n1\n1\n2\n1\n3\n1\n1\n4\n1\n2\n3\n4\n1\n1\n");

matrix<fraction<polynomial<double>>> mtrx;//there used to be a problem with %, and it only worked with int, now it seems to have disappeared

ss >> mtrx;

std::stringstream method\_ans;

method\_ans << mtrx.inverse\_M();

std::string true\_ans\_str1 ="sizex:2sizey:2\n[0][0] = \t(-36.125 -144.5x -289x^2 -433.5x^3 -397.375x^4 -144.5x^5) / (0 72.25x 180.625x^2 -108.375x^3 -614.125x^4 -541.875x^5 -144.5x^6)\t | [0][1] = \t(-2.82452e+14 -8.47357e+14x -5.64905e+14x^2) / (0 -5.64905e+14x -8.47357e+14x^2 1.69471e+15x^3 3.10698e+15x^4 1.12981e+15x^5)\t | \n[1][0] = \t(-4.5036e+15 -1.80144e+16x -1.35108e+16x^2) / (-0 -9.0072e+15x -1.35108e+16x^2 2.70216e+16x^3 4.95396e+16x^4 1.80144e+16x^5)\t | [1][1] = \t(-0.5 -0.5x) / (0 1x 0.5x^2 -3.5x^3 -2x^4)\t | \n";

std::cout << method\_ans.str();

EXPECT\_EQ(method\_ans.str(), true\_ans\_str1);

}

}

int main(int argc, char\*\* argv) {

::testing::GTEST\_FLAG(catch\_exceptions) = false;

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}