ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ЛАБОРАТОРНОЙ РАБОТЕ №1

По дисциплине «ООП»

ВАРИАНТ 15

Выполнил: ст. гр. ТКИ-141

Буланый Сергей Евгеньевич

Проверил: к.т.н., доц. Васильева М. А.

(Проверил: к.т.н, доц. Балакина Е. П.)

Москва 2024

Оглавление

[1 Решение задачи 3](#_Toc151483877)

[1.1 Формулировка задачи 3](#_Toc151483878)

[1.2 UML диаграмма алгоритма задание 4](#_Toc151483879)

[1.3 Текст программы на языке С++ задание 5](#_Toc151483880)

[1.4 Результаты выполнения программы 12](#_Toc151483881)

[1.5 Выполнение тестовых примеров задание 13](#_Toc151483881)

[1.6 Отметка о выполнении задания в веб-хостинге системы контроля версий задание 17](#_Toc151483882)

1. Решение задачи

**1.1 Формулировка** **задачи**

Реализовать два класса-наследника. Один класс Simple solver находит решение, если дискриминант положительный, в противном случае выбрасывает исключение. Второй класс ‑ Complex solver. Данный решатель используется для отыскания корней, лежащих не на вещественной прямой, в а комплексной плоскости. Для этого следует использовать класс **Комплексное** **число**.

Для класса **Комплексное** **число** переопределить операторы сдвига, сложения (комплексных чисел и комплексных чисел, комплексных чисел и вещественных чисел), вычитания, деления и умножения, равенства и неравенства. Реализовать методы представления в виде строки в экспоненциальной, алгебраической и тригонометрической форме записи, метод нахождения комплексно-сопряженного числа.

1.2 UML диаграмма алгоритма задание

UML диаграмма представлена ниже (Рисунок 1).

Изображение выглядит как текст, диаграмма, План, линия

Автоматически созданное описание

Рисунок 1– UML диаграмма

1.3 Текст программы на языке С++ задание

// Файл BaseAnswer.h

#include <string>

class BaseAnswer

{

public:

virtual std::string ToString() const = 0;

~BaseAnswer() {} ;

};

// Файл BaseSolver.h

#pragma once

#include "BaseAnswer.h"

class BaseSolver

{

public:

virtual BaseAnswer\* Solve(

const double a,

const double b,

const double c) const = 0;

double GetDiscriminant(

const double a,

const double b,

const double c) const;

};

// Файл Complex.h

#pragma once

#include <string>

#include <iostream>

#include "BaseAnswer.h"

class Complex

{

double real;

double imaginary;

public:

Complex();

Complex(double real, double imaginary);

std::string ToString() const;

Complex(const Complex& other);

friend std::ostream& operator<<(std::ostream& output, const Complex& a)

{

output << a.ToString();

return output;

}

Complex operator+(Complex other);

Complex operator-(Complex other);

Complex operator\*(Complex other);

Complex operator/(Complex other);

Complex conjugate();

std::string stringAlgebra()

{

return "Алгебраическая форма: " + this->ToString();

}

std::string stringTrig()

{

double r = this->real; // модуль числа

double theta = this->imaginary; // аргумент числа

return "Тригонометрическая форма: " + std::to\_string(r) + " \* (cos(" + std::to\_string(theta) + ") + i \* sin(" + std::to\_string(theta) + "))";

}

std::string stringExp()

{

double r = this->real; // модуль числа

double theta = this->imaginary; // аргумент числа

return "Экспотенциальная форма: " + std::to\_string(r) + " \* exp(i \* " + std::to\_string(theta) + ")";

}

};

// Файл ComplexAnswer.h

#pragma once

#include "Complex.h"

class ComplexAnswer : public BaseAnswer

{

public:

ComplexAnswer(Complex first, Complex second);

std::string ToString() const;

Complex getFirst() const;

Complex getSecond() const;

void setFirst(Complex number);

void setSecond(Complex number);

private:

Complex first;

Complex second;

};

// Файл ComplexSolver.h

#pragma once

#include "BaseSolver.h"

#include "ComplexAnswer.h"

#include <cmath>

#include <utility>

class ComplexSolver : public BaseSolver

{

public:

// Конструктор класса

ComplexSolver() = default;

// Метод для нахождения комплексных корней

BaseAnswer\* Solve(double a, double b, double c) const override;

};

// Файл RealAnswer.h

#pragma once

#include "BaseAnswer.h"

class RealAnswer : public BaseAnswer

{

public:

RealAnswer(const double first, const double second);

std::string ToString() const override;

private:

double first;

double second;

};

// Файл RealSolver.h

#pragma once

#include "BaseSolver.h"

class RealSolver : public BaseSolver

{

public:

BaseAnswer\* Solve(

const double a,

const double b,

const double c) const override;

};

// Файл BaseSolver.cpp

#include "BaseSolver.h"

double BaseSolver::GetDiscriminant(

const double a,

const double b,

const double c) const

{

return b \* b - 4 \* a \* c;

}

// Файл Complex.cpp

#include "Complex.h"

#include <format>

#include<limits>

Complex::Complex(double real, double imaginary)

:real(real), imaginary(imaginary)

{

}

std::string Complex::ToString() const

{

if (this->imaginary < 0)

{

return std::format("({} - {}i)", this->real, -this->imaginary);

}

return std::format("({} + {}i)", this->real, this->imaginary);

}

Complex Complex::operator+(Complex other)

{

return Complex(this->real + other.real, this->imaginary + other.imaginary);

}

Complex Complex::operator-(Complex other)

{

return Complex(this->real - other.real, this->imaginary - other.imaginary);

}

Complex Complex::operator\*(Complex other)

{

return Complex(this->real \* other.real - this->imaginary \* other.imaginary, this->real \* other.imaginary + this->imaginary \* other.real);

}

Complex Complex::operator/(Complex other)

{

auto a = this->real;

auto b = this->imaginary;

auto c = other.real;

auto d = other.imaginary;

if (a \* a + d \* d <= std::numeric\_limits<double>::epsilon()) throw;

return Complex((a \* c + b \* d) / (c \* c + d \* d), (b \* c - a \* d) / (c \* c + d \* d));

}

Complex Complex::conjugate()

{

return Complex(this->real, -this->imaginary);

}

Complex::Complex(const Complex& other)

{

this->real = other.real;

this->imaginary = other.imaginary;

}

Complex::Complex()

{

this->real = 0;

this->imaginary = 0;

}

// Файл ComplexAnswer.cpp

#include "ComplexAnswer.h"

ComplexAnswer::ComplexAnswer(Complex first, Complex second)

{

this->first = first;

this->second = second;

}

std::string ComplexAnswer::ToString() const

{

return first.ToString() + "\n" + second.ToString() + "\n";

}

Complex ComplexAnswer::getFirst() const

{

return first;

}

Complex ComplexAnswer::getSecond() const

{

return second;

}

void ComplexAnswer::setFirst(Complex number)

{

this->first = number;

}

void ComplexAnswer::setSecond(Complex number)

{

this->second = number;

}

// Файл ComplexSolver.cpp

#include "ComplexSolver.h"

#include "Complex.h"

BaseAnswer\* ComplexSolver::Solve(double a, double b, double c) const

{

ComplexAnswer \*roots = new ComplexAnswer(Complex(), Complex());

double discriminant = this->GetDiscriminant(a, b, c);

if (discriminant < 0)

{

// Если дискриминант отрицательный, тогда корни - комплексные числа

double real\_part = -b / (2 \* a);

double imaginary\_part = sqrt(-discriminant) / (2 \* a);

roots->setFirst(Complex(real\_part, imaginary\_part));

roots->setSecond(Complex(real\_part, -imaginary\_part));

}

else if (discriminant == std::numeric\_limits<double>::epsilon())

{

// Если дискриминант равен нулю, тогда корни - одинаковые действительные числа

double root = -b / (2 \* a);

roots->setFirst(Complex(root, 0));

}

else

{

// Если дискриминант положительный, тогда корни - два различных действительных числа

double root1 = (-b + sqrt(discriminant)) / (2 \* a);

double root2 = (-b - sqrt(discriminant)) / (2 \* a);

roots->setFirst(Complex(root1, 0));

roots->setSecond(Complex(root2, 0));

}

return roots;

}

// Файл RealAnswer.cpp

#include <sstream>

#include "RealAnswer.h"

RealAnswer::RealAnswer(const double first, const double second)

: first{ first }, second{ second }

{

}

std::string RealAnswer::ToString() const

{

std::stringstream buffer{};

buffer << "{ " << this->first << "; " << this->second << " }";

return buffer.str();

}

// Файл RealSolver.cpp

#include "RealSolver.h"

#include "RealAnswer.h"

#include <cmath>

#include <limits>

#include <stdexcept>

BaseAnswer\* RealSolver::Solve(

const double a,

const double b,

const double c) const

{

if (std::abs(a) <= std::numeric\_limits<double>::epsilon())

{

throw std::runtime\_error("No quadratic equation");

}

auto discriminant = this->GetDiscriminant(a, b, c);

if (discriminant < 0)

{

throw std::runtime\_error("No real roots");

}

const auto root = std::sqrt(discriminant);

const auto x1 = (-b + root) / (2 \* a);

const auto x2 = (-b - root) / (2 \* a);

return new RealAnswer(x1, x2);

}

// Файл main.cpp

#include <iostream>

#include <locale>

#include "../Solver/RealSolver.h"

#include "..\Solver\ComplexSolver.h"

#include "../Solver/ComplexAnswer.h"

#include "../Solver/BaseSolver.h"

/\*\*

\* @brief Точка входа в программу

\* @return 0 в случае успеха

\*/

int main()

{

std::setlocale(NULL, "ru\_RU.UTF-8");

double a = 2.0, b = 7.0, c = 3.0;

auto solver = RealSolver();

if (solver.GetDiscriminant(a, b, c) > 0) std::cout << solver.Solve(a, b, c)->ToString() << std::endl;

Complex a1(1, -1);

Complex a2(3, 6);

Complex b1 = a1 \* a2;

Complex b2 = a1 / a2;

std::cout << b1 << std::endl << b2 << std::endl << b1.stringExp() << std::endl << b1.stringTrig() << std::endl;

auto comSolver = ComplexSolver();

auto roots = comSolver.Solve(10, 5, 9);

std::cout << "Комплексные корни:\n";

std::cout << roots->ToString() << std::endl;

return 0;

}

1.4 Результаты выполнения программы

Результаты выполнения программы в C++ представлена ниже (Рисунок 2).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Результаты выполнения программы

1.5 Выполнение тестовых примеров задание

Выполнены тестовые примеры. Результаты их выполнения представлены ниже (Рисунок 3–9).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 3 – Результаты выполнения тестов

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 4 – Результаты выполнения тестов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 5 – Результаты выполнения тестов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 6 – Результаты выполнения тестов

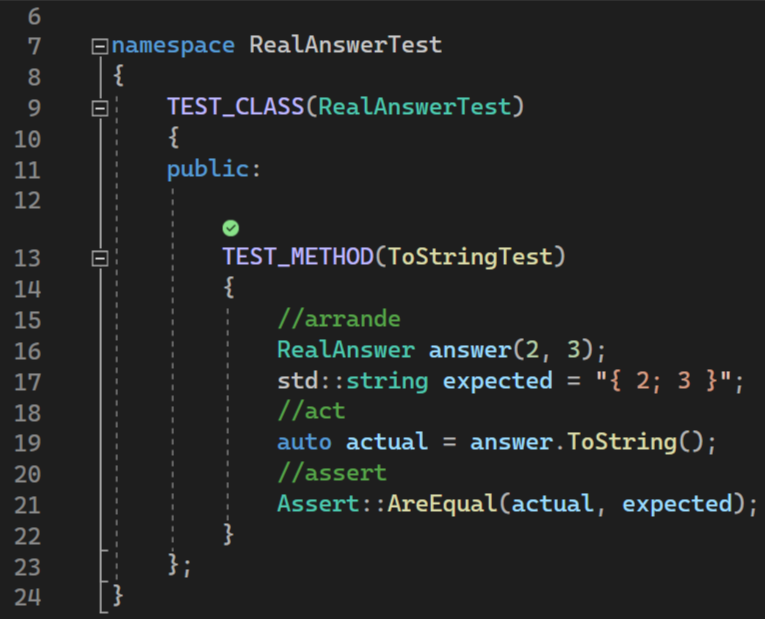


Рисунок 7 – Результаты выполнения тестов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 8 – Результаты выполнения тестов

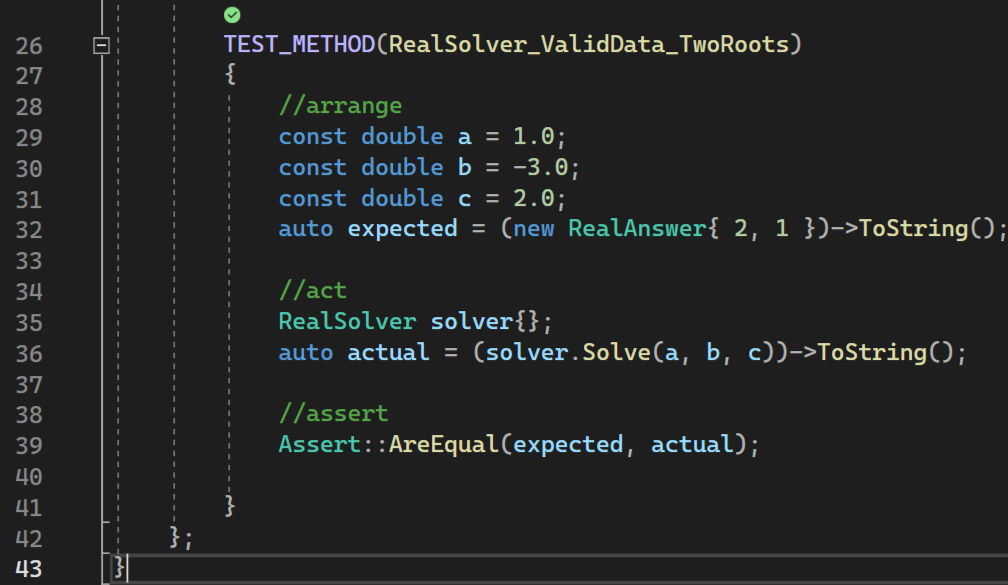


Рисунок 9 – Результаты выполнения тестов

1.6 Отметка о выполнении задания в веб-хостинге системы контроля версий задание

Ниже представлено доказательство того, что задание было принято. (Рисунок 10)

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 10 – Approve задачи