** Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления» Кафедра «Системы обработки информации и управления»**

Лабораторная работа №6

по предмету

«Разработка комплексного приложения»

Выполнил:

студент группы № ИУ5-33Б

Комаров Дмитрий

Проверил:

Преподаватель кафедры ИУ-5

Гапанюк Юрий

2022 г.

**Задание.**

Задание: Реализуйте любое из заданий курса на языке программирования Rust. Разработайте хотя бы один макрос. Разработайте модульные тесты (не менее 3 тестов).

**Код программы:**

use std::io;

#[derive(Debug, Copy, Clone)]

///Тип решения квадратного уравнения

enum SquareRootResult {

    /// Unit-тип

    NoRoots,

    /// Один корень - кортежная структура

    OneRoot(f64),

    /// С-подобная структура 2 корня

    TwoRoots {root1: f64, root2: f64},

    ThreeRoots {root1: f64, root2: f64, root3: f64},

    /// 4 - корня

    FourRoots {root1: f64, root2: f64, root3: f64, root4: f64},

}

#[derive(Debug, Copy, Clone)]

/// Структура, соответствующая уравнению

struct Equation {

    /// Коэффициент A

    c\_a: f64,

    /// Коэффициент B

    c\_b: f64,

    /// Коэффициент C

    c\_c: f64,

    /// Дискриминант

    diskr: f64,

    /// Корни

    res: SquareRootResult,

}

impl Equation {

    /// Функция вычисления корней

    fn calculate\_roots(&mut self)->Vec<f64> {

        self.diskr = self.c\_b.powi(2) - 4.0 \* self.c\_a \* self.c\_c;

        let mut values:Vec<f64>=Vec::new();

        if self.diskr == 0.0 {

            let rt = -self.c\_b / (2.0 \* self.c\_a);

            if rt > 0.0 {

                values.push(rt.sqrt());

                values.push(-rt.sqrt());

            } else if rt == 0.0 {

                values.push(rt);

            }

        }

        else if self.diskr > 0.0 {

            let mut rt1 = (-self.c\_b - self.diskr.sqrt()) / (2.0 \* self.c\_a);

            let mut rt3 = (-self.c\_b + self.diskr.sqrt()) / (2.0 \* self.c\_a);

            if rt1 > 0.0 {

                rt1 = rt1.sqrt();

                values.push(rt1);

                values.push(-rt1);

            }else if rt1 == 0.0 {

                values.push(rt1);

            }

            if rt3 > 0.0 {

                rt3 = rt3.sqrt();

                values.push(rt3);

                values.push(-rt3);

            }else if rt3 == 0.0 {

                values.push(rt3)

            }

        }

        return values

    }

    /// Ввод одного коэффициента

    fn get\_coef(message: &str) -> f64 {

        return loop {

            let mut input = String::new();

            println!("{}", message);

            io::stdin()

                .read\_line(&mut input)

                .expect("Неверно введена строка");

            match input.trim().parse() {

                Ok(val) => {

                    break val;

                }

                Err(\_) => {

                    continue;

                }

            }

        };

    }

    fn get\_coefs(&mut self) -> () {

        self.c\_a = Equation::get\_coef("Введите коэффициент A: ");

        self.c\_b = Equation::get\_coef("Введите коэффициент B: ");

        self.c\_c = Equation::get\_coef("Введите коэффициент C: ");

    }

}

#[cfg(test)]

mod tests{

    use super::\*;

    #[test]

    fn test1(){

        let mut eq = Equation {

            c\_a: 1.0,

            c\_b: -5.0,

            c\_c: -36.0,

            diskr: 0.0,

            res: SquareRootResult::NoRoots,

        };

        eq.diskr = eq.c\_b.powi(2) - 4.0 \* eq.c\_a \* eq.c\_c;

        let mut need:Vec<f64>=Vec::new();

        need.push(3.0);

        need.push(-3.0);

        assert\_eq!(eq.calculate\_roots(), need);

    }

    #[test]

    fn test2() {

        let mut eq = Equation {

            c\_a: 1.0,

            c\_b: -5.0,

            c\_c: 4.0,

            diskr: 0.0,

            res: SquareRootResult::NoRoots,

        };

        eq.diskr = eq.c\_b.powi(2) - 4.0 \* eq.c\_a \* eq.c\_c;

        let mut need:Vec<f64>=Vec::new();

        need.push(1.0);

        need.push(-1.0);

        need.push(2.0);

        need.push(-2.0);

        assert\_eq!(eq.calculate\_roots(), need);

    }

    #[test]

    fn test3(){

        let mut eq = Equation {

            c\_a: -4.0,

            c\_b: 16.0,

            c\_c: 0.0,

            diskr: 0.0,

            res: SquareRootResult::NoRoots,

        };

        eq.diskr = eq.c\_b.powi(2) - 4.0 \* eq.c\_a \* eq.c\_c;

        let mut need:Vec<f64>=Vec::new();

        need.push(2.0);

        need.push(-2.0);

        need.push(-0.0);

        assert\_eq!(eq.calculate\_roots(), need);

    }

}

fn main() {

    use SquareRootResult::\*;

    let mut eq = Equation {

        c\_a: 0.0,

        c\_b: 0.0,

        c\_c: 0.0,

        diskr: 0.0,

        res: SquareRootResult::NoRoots,

    };

    eq.get\_coefs();

    let values\_f:Vec<f64>=eq.calculate\_roots();

    eq.res = {

        if values\_f.len() == 2 {

            TwoRoots {

                root1: values\_f[0],

                root2: values\_f[1],

            }

        } else if values\_f.len() == 1 {

            OneRoot(values\_f[0])

        } else if values\_f.len() == 3 {

            ThreeRoots {

                root1: values\_f[0],

                root2: values\_f[1],

                root3: values\_f[2],

            }

        } else if values\_f.len() == 4 {

            FourRoots {

                root1: values\_f[0],

                root2: values\_f[1],

                root3: values\_f[2],

                root4: values\_f[3],

            }

        } else {

            NoRoots

        }

    };

    let text\_res = match eq.res {

        NoRoots => format!("Корней нет"),

        OneRoot(rt) => format!("Один корень => {}", rt),

        TwoRoots { root1, root2 } => format!("Два корня => {} и {}", root1, root2),

        ThreeRoots { root1, root2, root3} => format!("Два корня => {}, {} и {}", root1, root2, root3),

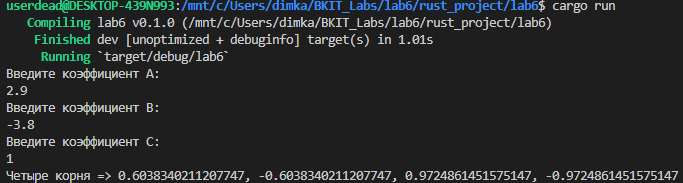
        FourRoots { root1, root2, root3, root4 } => format!("Четыре корня => {}, {}, {}, {}", root1, root2, root3, root4),

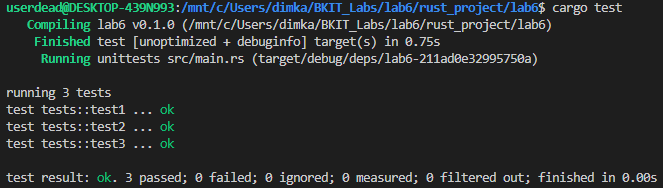
    };

    println!("{}", text\_res);

}

**Результат выполнения программы:**

****

****