Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э.Баумана (национальный исследовательский университет)»

Факультет «Информатика и системы управления»

Курс «Базовые компоненты интернет-технологий»

Лабораторная работа №6 «Разработка на языке программирования Rust»

Выполнил:

студент группы ИУ5-33Б Николай Горкунов

подпись: , дата:

Проверил:

преподаватель Юрий Гапанюк

подпись: , дата:

2022 г.

Задание.

1. Реализуйте любое из заданий курса на языке программирования Rust (биквадратное уравнение из первой лабораторной).
2. Разработайте хотя бы один макрос.
3. Разработайте модульные тесты (не менее 3 тестов).

Исходный код.

main.rs:

```
extern crate quartic_equation;
use quartic_equation::qe_struct::*;
use SquareRootResult::*;
use std::env;

fn main() {
    let mut v: Vec::<String> = env::args().collect();
    v.remove(0);
    let mut eq = Equation::new(v);
    eq.update_coefs();
    eq.calculate_roots();
    let res_str = match eq.res {
        Ok(res) => match 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!("Три корня: {}; {} и {}.", root1, root2,
root3),
            FourRoots {
                root1,
                root2,
                root3,
                root4
            } => format!("Четыре корня: {}; {}; {} и {}.", root1,
root2, root3, root4),
        },
        Err(err) => format!("Ошибка: {}.", err),
    };
    println!("{}", res_str);
}
```

qe_struct.rs:

```rust
use std::io;

#[derive(Debug, Copy, Clone)]
///Тип решения квадратного уравнения
pub enum SquareRootResult {
    /// Unit-тип
    NoRoots,
    /// Один корень - кортежная структура
    OneRoot(f64),
    /// C-подобная структура
    TwoRoots { root1: f64, root2: f64 },
    /// Один корень
    ThreeRoots { root1: f64, root2: f64, root3: f64 },
    /// Один корень
    FourRoots { root1: f64, root2: f64, root3: f64, root4: f64 },
}

#[derive(Debug, Clone)]
/// Структура, соответствующая уравнению
pub struct Equation {
    /// Коэффициент A
    pub c_a: f64,
    /// Коэффициент B
    pub c_b: f64,
    /// Коэффициент C
    pub c_c: f64,
    /// Дискриминант
    pub diskr: f64,
    /// Корни
    pub res: Result::<SquareRootResult, &'static str>,
    /// Коэффициенты на вход new
    pub coefs: Vec<String>,
}

#[macro_export]
/// При получении коэффициента был повторяющийся код
macro_rules! my_macro {
    ( $x:expr , $y:tt ) => {
        match $x.trim().parse() {
            Ok(val) => {
                $y val;
```

```
            }
            Err(_) => ()
        }
    };
}

#[macro_export]
/// Более удобный Equation new
macro_rules! qe_new {
    ( $a:expr , $b:expr , $c:expr ) => {
        Equation::new(vec![$a.to_string(), $b.to_string(),
$c.to_string()]);
    };
}

impl Equation {
    /// Функция создания
    pub fn new(coefs: Vec::<String>) -> Self {
        Self {
            c_a: 0.0,
            c_b: 0.0,
            c_c: 0.0,
            diskr: 0.0,
            res: Ok(SquareRootResult::NoRoots),
            coefs,
        }
    }

    fn big_d_to_roots(&mut self, roots: &mut Vec::<f64>, big_d:
f64) -> i8 {
        if big_d == 0.0 {
            let root = big_d.sqrt();
            roots.push(root);
            return 1;
        } else if big_d > 0.0 {
            let root = big_d.sqrt();
            roots.push(-root);
            roots.push(root);
            return 2;
        } else {
            return 0;
        }
```

```
    }

    /// Функция вычисления корней
    pub fn calculate_roots(&mut self) {
        self.diskr = self.c_b.powi(2) - 4.0 * self.c_a * self.c_c;
        let mut roots = Vec::<f64>::new();
        self.res = {
            if self.diskr < 0.0 {
                Ok(SquareRootResult::NoRoots)
            } else if self.diskr == 0.0 {
                let big_d = -self.c_b / (2.0 * self.c_a);
                let cnt = self.big_d_to_roots(&mut roots, big_d);
                match cnt {
                    0 => Ok(SquareRootResult::NoRoots),
                    1 => Ok(SquareRootResult::OneRoot(roots[0])),
                    2 => Ok(SquareRootResult::TwoRoots {
                        root1: roots[0],
                        root2: roots[1],
                    }),
                    _ => Err("невозможное количество корней"),
                }
            } else {
                let sq_d = self.diskr.sqrt();
                let mut big_d = (-self.c_b + sq_d) / (2.0 *
self.c_a);
                let mut cnt = self.big_d_to_roots(&mut roots,
big_d);
                big_d = (-self.c_b - sq_d) / (2.0 * self.c_a);
                cnt += self.big_d_to_roots(&mut roots, big_d);
                match cnt {
                    0 => Ok(SquareRootResult::NoRoots),
                    1 => Ok(SquareRootResult::OneRoot(roots[0])),
                    2 => Ok(SquareRootResult::TwoRoots {
                        root1: roots[0],
                        root2: roots[1],
                    }),
                    3 => Ok(SquareRootResult::ThreeRoots {
                        root1: roots[0],
                        root2: roots[1],
                        root3: roots[2],
                        }),
                    4 => Ok(SquareRootResult::FourRoots {
```

```
                        root1: roots[0],
                        root2: roots[1],
                        root3: roots[2],
                        root4: roots[3],
                    }),
                    _ => Err("невозможное количество корней"),
                }
            }
        };
    }

    /// Обновление одного коэффициента, требует ввод если дан не
был или некорректный
    fn get_coef(&mut self, index: i8, message: &str) -> f64 {
        let mut input = String::new();
        if self.coefs.len() > index as usize{
            input = self.coefs[index as usize].clone();
            my_macro!(input, return)
        }
        return loop {
            input.clear();
            println!("{}", message);
            io::stdin()
                .read_line(&mut input)
                .expect("Неверно введена строка");
            my_macro!(input, break)
        };
    }

    /// Обновление коэффициентов уравнения, требует ввод
недостающих и некорректных
    pub fn update_coefs(&mut self) -> () {
        self.c_a = self.get_coef(0, "Введите коэффициент A: ");
        self.c_b = self.get_coef(1, "Введите коэффициент B: ");
        self.c_c = self.get_coef(2, "Введите коэффициент C: ");
    }
}
```

lib.rs:

```
// Здесь не на чего смотреть. Таков путь раста.
pub mod qe_struct;
```

test_calculate_roots.rs:

```rust
#[macro_use]
extern crate quartic_equation;
use quartic_equation::qe_struct::*;
use SquareRootResult::*;

#[test]
/// Проверка биквадратного уравнения без корней
fn zero_roots() {
    let mut eq = qe_new!(1, 1, 1);
    eq.update_coefs();
    eq.calculate_roots();
    match eq.res {
        Ok(res) => match res {
            NoRoots => (),
            _ => assert!(false, "Корней не должно было быть!"),
        }
        Err(err) => assert!(false, "Ошибка: {}", err),
    }
}

#[test]
/// Проверка биквадратного уравнения с одним корнем
fn one_root() {
    const EXPECTED: f64 = 0.0;
    let mut eq = qe_new!(1, 0, 0);
    eq.update_coefs();
    eq.calculate_roots();
    match eq.res {
        Ok(res) => match res {
            OneRoot(root) => assert!(root == EXPECTED, "Корень не
верный: {} != {}", root, EXPECTED),
            _ => assert!(false, "Корень должен был быть один!"),
        }
        Err(err) => assert!(false, "Ошибка: {}", err),
    }
}

#[test]
/// Проверка биквадратного уравнения с двумя корнями
fn two_roots() {
```

```
    const EXPECTED_1: f64 = -1.0;
    const EXPECTED_2: f64 = 1.0;
    let mut eq = qe_new!(1, -2, 1);
    eq.update_coefs();
    eq.calculate_roots();
    match eq.res {
        Ok(res) => match res {
            TwoRoots { root1, root2 } => {
                assert!(root1 == EXPECTED_1, "Первый корень не
верный: {} != {}", root1, EXPECTED_1);
                assert!(root2 == EXPECTED_2, "Второй корень не
верный: {} != {}", root2, EXPECTED_2);
            },
            _ => assert!(false, "Корня должно было быть два!"),
        }
        Err(err) => assert!(false, "Ошибка: {}", err),
    }
}

#[test]
/// Проверка биквадратного уравнения с тремя корнями
fn three_roots() {
    const EXPECTED_1: f64 = -1.0;
    const EXPECTED_2: f64 = 1.0;
    const EXPECTED_3: f64 = 0.0;
    let mut eq = qe_new!(2, -2, 0);
    eq.update_coefs();
    eq.calculate_roots();
    match eq.res {
        Ok(res) => match res {
            ThreeRoots { root1, root2, root3 } => {
                assert!(root1 == EXPECTED_1, "Первый корень не
верный: {} != {}", root1, EXPECTED_1);
                assert!(root2 == EXPECTED_2, "Второй корень не
верный: {} != {}", root2, EXPECTED_2);
                assert!(root3 == EXPECTED_3, "Третий корень не
верный: {} != {}", root3, EXPECTED_3);
            },
            _ => assert!(false, "Корня должно было быть три!"),
        }
        Err(err) => assert!(false, "Ошибка: {}", err),
    }
```

```
}

#[test]
/// Проверка биквадратного уравнения с четырьмя корнями
fn four_roots() {
    const EXPECTED_1: f64 = -1.0;
    const EXPECTED_2: f64 = 1.0;
    const EXPECTED_3: f64 = -0.5;
    const EXPECTED_4: f64 = 0.5;
    let mut eq = qe_new!(4, -5, 1);
    eq.update_coefs();
    eq.calculate_roots();
    match eq.res {
        Ok(res) => match res {
            FourRoots { root1, root2, root3, root4 } => {
                assert!(root1 == EXPECTED_1, "Первый корень не
верный: {} != {}"   , root1, EXPECTED_1);
                assert!(root2 == EXPECTED_2, "Второй корень не
верный: {} != {}"   , root2, EXPECTED_2);
                assert!(root3 == EXPECTED_3, "Третий корень не
верный: {} != {}"   , root3, EXPECTED_3);
                assert!(root4 == EXPECTED_4, "Четвёртый корень не
верный: {} != {}", root4, EXPECTED_4);
            },
            _ => assert!(false, "Корня должно было быть
четыре!"),
        }
        Err(err) => assert!(false, "Ошибка: {}", err),
    }
}
```

test_update_coefs.rs:

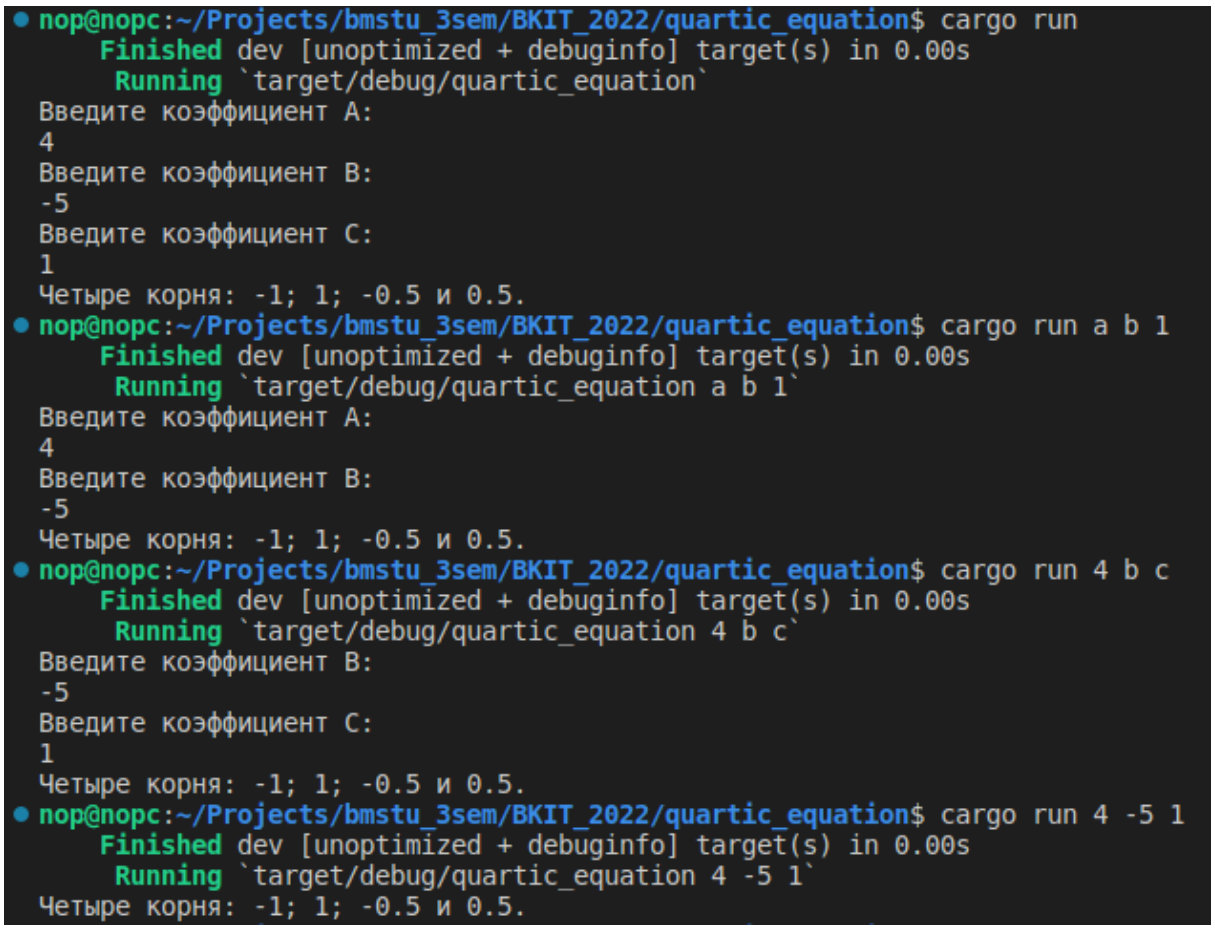
```
use std::process::{Command, Stdio};

#[test]
/// Проверка ввода трёх коэффициентов (происходит расчёт ответа)
fn three_coefs_input() {
    const EXPECTED: &str = "Корней нет\n";
    match
Command::new("cargo").arg("build").stdout(Stdio::null()).spawn() {
        Ok(..) => {
            let output = Command::new("cargo").args(["run", "1",
```

```
"1", "1"]).output().expect("command cargo run failed");
            assert!(output.status.success());
            let str = String::from_utf8_lossy(&output.stdout);
            assert!(str == EXPECTED, "Неверный ответ: \"{}\" !=
\"{}\"", str, EXPECTED);
        },
        Err(err) => assert!(false, "Ошибка: {}", err),
    }
}
```

Пример выполнения.

Тестирование вручную:



Автоматическое тестирование:

```
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022/quartic_equation$ cargo test --test '*'
    Finished test [unoptimized + debuginfo] target(s) in 0.03s
     Running tests/test_calculate_roots.rs (target/debug/deps/test_calculate_roots-b0a8d1e74cea1aca)

running 5 tests
test four_roots ... ok
test one_root ... ok
test three_roots ... ok
test two_roots ... ok
test zero_roots ... ok

test result: ok. 5 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

     Running tests/test_update_coefs.rs (target/debug/deps/test_update_coefs-4da307177549d821)

running 1 test
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
test three_coefs_input ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.07s

nop@nopc:~/Projects/bmstu_3sem/BKIT_2022/quartic_equation$ cargo build
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022/quartic_equation$ cargo run 4 -5 1
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/quartic_equation 4 -5 1`
Четыре корня: -1; 1; -0.5 и 0.5.
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022/quartic_equation$ cargo test --test '*'
    Finished test [unoptimized + debuginfo] target(s) in 0.00s
     Running tests/test_calculate_roots.rs (target/debug/deps/test_calculate_roots-b0a8d1e74cea1aca)

running 5 tests
test one_root ... ok
test four_roots ... ok
test three_roots ... ok
test two_roots ... ok
test zero_roots ... ok

test result: ok. 5 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

     Running tests/test_update_coefs.rs (target/debug/deps/test_update_coefs-4da307177549d821)

running 1 test
test three_coefs_input ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.06s

    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
nop@nopc:~/Projects/bmstu_3sem/BKIT_2022/quartic_equation$
```