МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе №6
“Разработка на языке программирования Rust”

Выполнил:
Студент группы ИУ5-35Б
Рябов М.А.
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

Задание

1. Реализуйте любое из заданий курса на языке программирования Rust.

   Разработать программу для решения биквадратного уравнения. (Лабораторная работа №1)

   - Программа должна быть разработана в виде консольного приложения на языке Rust.
   - Программа осуществляет ввод с клавиатуры коэффициентов A, B, C, вычисляет дискриминант и ДЕЙСТВИТЕЛЬНЫЕ корни уравнения (в зависимости от дискриминанта).
   - Коэффициенты A, B, C могут быть заданы в виде параметров командной строки. Если они не заданы, то вводятся с клавиатуры в соответствии с пунктом 2.
   - Если коэффициент A, B, C введен или задан в командной строке некорректно, то необходимо проигнорировать некорректное значение и вводить коэффициент повторно пока коэффициент не будет введен корректно. Корректно заданный коэффициент - это коэффициент, значение которого может быть без ошибок преобразовано в действительное число.

2. Разработайте хотя бы один макрос.
3. Разработайте модульные тесты (не менее 3 тестов).

Листинг кода

```rust
use std::env;
use std::io::{self, Write};

fn get_coefficient(args: &Vec<String>, index: usize, name: &str) -> f64 {
    match args.get(index) {
        Some(coefficient_string) => {
            match coefficient_string.parse::<f64>() {
                Ok(coefficient) => return coefficient,
                Err(_) => {}
            }
        },
        None => {}
    }

    loop {
        let mut coefficient_string = String::new();

        print!("Введите коэффициент {}: ", name);
        std::io::stdout().flush().unwrap();
        match io::stdin().read_line(&mut coefficient_string) {
```

```
            Ok(_) => {},
            Err(error) => println!("Ошибка: {}", error)
        }

        match coefficient_string.trim().parse::<f64>() {
            Ok(coefficient) => return coefficient,
            Err(_) => println!("Введите действительное число!")
        }
    }
}

macro_rules! calculate_discriminant {
    ($a:expr, $b:expr, $c:expr) => {
        ($b as f64) * ($b as f64) - 4.0 * ($a as f64) * ($c as f64)
    };
}

fn get_roots(a: f64, b: f64, c: f64) -> Vec<f64> {
    let mut roots: Vec<f64> = Vec::new();

    let discriminant: f64 = calculate_discriminant!(a, b, c);

    if discriminant == 0.0 {
        let root = -b / (2.0 * a);

        if root >= 0.0 {
            let squared_root = root.sqrt();

            roots.push(squared_root);
            roots.push(-1.0 * squared_root);
        }
    } else if discriminant > 0.0 {
        let squared_discriminant: f64 = discriminant.sqrt();

        let first_root = (-b - squared_discriminant) / (2.0 * a);
        let second_root = (-b + squared_discriminant) / (2.0 * a);

        if first_root >= 0.0 {
            let squared_first_root = first_root.sqrt();

            roots.push(squared_first_root);
            roots.push(-1.0 * squared_first_root);
        }

        if second_root >= 0.0 {
            let squared_second_root = second_root.sqrt();

            roots.push(squared_second_root);
            roots.push(-1.0 * squared_second_root);
        }
    }
```

```
    roots.sort_by(f64::total_cmp);

    roots.dedup();

    return roots;
}

fn main() {
    let args: Vec<_> = env::args().collect();

    let a: f64 = get_coefficient(&args, 1, "A");
    let b: f64 = get_coefficient(&args, 2, "B");
    let c: f64 = get_coefficient(&args, 3, "C");

    let roots = get_roots(a, b, c);

    let roots_amount: usize = roots.len();

    match roots_amount {
        0 => println!("Нет корней"),
        1 => println!("Корень уравнения: {}", roots[0]),
        _ => {
            print!("Корни уравнения: ");

            for i in roots {
                print!("{} ", i);
            }

            println!("");
        }
    }
}

#[cfg(test)]
mod tests {
    use crate::get_coefficient;
    use crate::get_roots;

    #[test]
    fn test_get_coefficient() {
        let args: Vec<_> = vec!["./target".to_string(), "1".to_string(), "-
4".to_string(), "5".to_string()];

        let a = get_coefficient(&args, 1, "A");
        assert_eq!(a, 1.0);

        let b = get_coefficient(&args, 2, "B");
        assert_eq!(b, -4.0);

        let c = get_coefficient(&args, 3, "C");
```

```
        assert_eq!(c, 5.0);
    }

    #[test]
    fn test_calculate_discriminant() {
        let discriminant = calculate_discriminant!(1, -2, 1);
        assert_eq!(discriminant, 0.0);
    }

    #[test]
    fn test_get_roots() {
        let roots = get_roots(1.0, -2.0, 1.0);
        assert_eq!(roots, vec![-1.0, 1.0]);
    }
}
```

Выполнение программы

```
mmmmm@LAPTOPIC:~/projects/pcpl_2025/lab_6$ cargo run
   Compiling lab_6 v0.1.0 (/home/mmmmm/projects/pcpl_2025/lab_6)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.28s
     Running `target/debug/lab_6`
Введите коэффициент A: 1
Введите коэффициент B: -3
Введите коэффициент C: 2
Корни уравнения: -1.4142135623730951 -1 1 1.4142135623730951
```

```
mmmmm@LAPTOPIC:~/projects/pcpl_2025/lab_6$ cargo test
   Compiling lab_6 v0.1.0 (/home/mmmmm/projects/pcpl_2025/lab_6)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.26s
     Running unittests src/main.rs (target/debug/deps/lab_6-abc4e423c3cf8a5b)

running 3 tests
test tests::test_calculate_discriminant ... ok
test tests::test_get_coefficient ... ok
test tests::test_get_roots ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```