Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №1

Выполнил:
студент группы ИУ5-35Б
Талаев А.П.
Подпись и дата:

Проверил:
Подпись и дата:

Москва, 2024 г

Задание:
Разработать программу для решения биквадратного уравнения.

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 (вариант задания параметров приведен в конце файла с примером кода). Если они не заданы, то вводятся с клавиатуры в соответствии с пунктом 2. Описание работы с параметрами командной строки.
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.

Выполнение:
```
use std::env;
use std::io;

fn get_from_terminal(msg: &str) -> f64 {
    loop {
        println!("{}", msg);
        let mut coefficient = String::new();
        io::stdin().read_line(&mut coefficient).expect("Failed to read line");

        match coefficient.trim().parse::<f64>() {
            Ok(value) => return value,
            Err(_) => {println!("Incorrect input.")}
        }
    }
}

fn get_from_parameters(index: usize, msg: &str, args: &Vec<String>) -> f64 {
    return match args[index].parse() {
        Ok(value) => value,
        Err(_) => {
            println!("Incorrect value of the coefficient. Enter it again.");
            get_from_terminal(msg)
        }
    };
}
```

```
fn get_coefficient(a: &mut f64, b: &mut f64, c: &mut f64) {
    let args: Vec<String> = env::args().collect();
    if args.len() == 4 {
        *a = get_from_parameters(1, "Enter the coefficient A.", &args);
        *b = get_from_parameters(2, "Enter the coefficient B.", &args);
        *c = get_from_parameters(3, "Enter the coefficient C.", &args);
    }
    else {
        *a = get_from_terminal("Enter the coefficient A.");
        *b = get_from_terminal("Enter the coefficient B.");
        *c = get_from_terminal("Enter the coefficient C.");
    }
}

fn calculating_roots(a: f64, b: f64, c: f64) {
    if a == 0.0 {
        panic!("Coefficient A cannot be equal to 0.");
    }

    let d:f64  = b *  b - 4.0 * a * c;
    println!("Discriminant D = {}", d);

    let mut roots: Vec<f64> = Vec::new();
    if d > 0.0 {
        let x1: f64 = (-b + d.sqrt()) / (2.0 * a);
        let x2: f64 = (-b - d.sqrt()) / (2.0 * a);
        if x1 >= 0.0 {
            roots.push(x1.sqrt());
            roots.push(-x1.sqrt());
        }
        if x2 >= 0.0 {
            roots.push(x2.sqrt());
            roots.push(-x2.sqrt());
        }
    } else if d == 0.0 {
        let mut roots: Vec<f64> = Vec::new();
        let x1: f64 = (-b + d.sqrt()) / (2.0 * a);
        if x1 >= 0.0 {
            roots.push(x1.sqrt());
            roots.push(-x1.sqrt());
        }
    }

    if !roots.is_empty() {
        println!("Valid roots:");
```

```
        for root in roots {
            println!("-> {}", root);
        }
    } else {
        println!("Invalid roots.");
    }

}

fn main() {
    let mut a: f64 = 0.0;
    let mut b: f64 = 0.0;
    let mut c: f64 = 0.0;
    get_coefficient(&mut a, &mut b, &mut c);
    println!("A = {}, B = {}, C = {}", a, b, c);
    calculating_roots(a, b, c);
}
```

Результаты:

cargo run 1 2 3

A = 1, B = 2, C = 3

Discriminant D = -8

Invalid roots.