Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №1
«Основные конструкции языка Rust»

Выполнил:
студент группы ИУ5-24Б

Соколов Б. О.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5

Гапанюк Ю. Е.
Подпись и дата:

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

Код:

```
use std::io;

#[derive(Debug, Copy, Clone)]
enum TypeOfRes {
   Without,
   OneSolv(f64),
   TwoSolv{Root1: f64, Root2: f64},
   ThreeSolv{Root1: f64, Root2: f64, Root3: f64},
   FourSolv{Root1: f64, Root2: f64, Root3: f64, Root4: f64}
}

#[derive(Debug, Copy, Clone)]
struct Equation {
   coef_A: f64,
   coef_B: f64,
   coef_C: f64,
   Diskr: f64,
   result: TypeOfRes
}

impl Equation{
   fn calculation(&mut self){
      self.Diskr = self.coef_B.powi(2) - 4.0 * self.coef_A * self.coef_C;
      self.result = {
         if self.coef_A == 0.0 && self.coef_B == 0.0 {
            TypeOfRes::Without
         }else if self.coef_A == 0.0 && self.coef_B == 0.0 && self.coef_C == 0.0 {
            TypeOfRes::Without
         }else if self.coef_A > 0.0 && self.coef_B > 0.0 && self.coef_C > 0.0 {
            TypeOfRes::Without
         }else if self.coef_C == 0.0 && (self.coef_B == 0.0 || self.coef_B / self.coef_A > 0.0) {
            TypeOfRes::OneSolv(0.0)
         }else if self.coef_C == 0.0 && (self.coef_B / self.coef_A < 0.0) {
            let root2 = (self.coef_B / self.coef_A).abs().sqrt();
            let root3 = -(self.coef_B / self.coef_A).abs().sqrt();
            TypeOfRes::ThreeSolv { Root1: (0.0), Root2: (root2), Root3: (root3) }
         }else if self.coef_A == 0.0 && (-self.coef_C / self.coef_B > 0.0) {
            let root1 = (-self.coef_C / self.coef_B).abs().sqrt();
            let root2 = -(-self.coef_C / self.coef_B).abs().sqrt();
            TypeOfRes::TwoSolv { Root1: (root1), Root2: (root2) }
         }else if self.Diskr < 0.0 {
            TypeOfRes::Without
         }else if self.Diskr == 0.0 {
            let root: f64 = - self.coef_B / (2.0 * self.coef_A);
```

```
            if root > 0.0 {
                let root1 = root.abs().sqrt();
                let root2 = -root.abs().sqrt();
                TypeOfRes::TwoSolv { Root1: (root1), Root2: (root2) }
            }else if root == 0.0{
                TypeOfRes::OneSolv(0.0)
            }else{
                TypeOfRes::Without
            }
        }else {
            let root_one: f64 = (-self.coef_B - self.Diskr.sqrt()) / (2.0 * self.coef_A);
            let root_sec: f64 = (-self.coef_B + self.Diskr.sqrt()) / (2.0 * self.coef_A);
            if root_one > 0.0 && root_sec > 0.0 {
                let root1 = root_one.sqrt();
                let root2 = -root_one.sqrt();
                let root3 = root_sec.sqrt();
                let root4 = -root_sec.sqrt();
                TypeOfRes::FourSolv { Root1: (root1), Root2: (root2), Root3: (root3), Root4:
(root4) }
            }else if root_one > 0.0 && root_sec < 0.0 {
                let root1 = root_one.sqrt();
                let root2 = -root_one.sqrt();
                TypeOfRes::TwoSolv { Root1: (root1), Root2: (root2) }
            }else if root_one < 0.0 && root_sec > 0.0 {
                let root1 = root_sec.sqrt();
                let root2 = -root_sec.sqrt();
                TypeOfRes::TwoSolv { Root1: (root1), Root2: (root2) }
            }else{
                TypeOfRes::Without
            }
        }
    };
}

fn get_coef(line: &str) -> f64 {
    return loop {
        let mut input = String::new();
        println!("{}", line);
        io::stdin()
            .read_line(&mut input)
            .expect("Error!");
        match input.trim().parse() {
            Ok(res) => {
                break res;
            }
            Err(_) => {
                print!("Error!\n");
                continue;
            }
        }
    };
}
```

```
    fn CoefsInRaw(&mut self) -> () {
        self.coef_A = Equation::get_coef("Введите A: ");
        self.coef_B = Equation::get_coef("Введите B: ");
        self.coef_C = Equation::get_coef("Введите C: ");
    }
}

macro_rules! print_Equation {
    ($A:expr, $B:expr, $C:expr) => {
        let mut sign1 = String::new();
        let mut sign2 = String::new();
        if $B < 0.0 {
            sign1 = String::from(" - ");
        }else{
            sign1 = String::from(" + ");
        }
        if $C < 0.0 {
            sign2 = String::from(" - ");
        }else{
            sign2 = String::from(" + ");
        }
        if $A == 0.0 && $B == 0.0 && $C == 0.0{
            println!("Такого уравнения нет");
        }else if $A == 0.0{
            println!("Введённое уравнение: {}x^2 {} {} = 0", $B.abs(), sign2, $C.abs());
        }else if $B == 0.0{
            println!("Введённое уравнение: {}x^4 {} {} = 0", $A, sign2, $C.abs());
        }else if $C == 0.0{
            println!("Введённое уравнение: {}x^4 {} {}x^2 = 0", $A, sign1, $B.abs());
        }else{
            println!("Введённое уравнение: {}x^4 {} {}x^2 {} {} = 0", $A, sign1, $B.abs(), sign2,
$C.abs());
        }
    }
}

fn main() {
    use TypeOfRes::*;
    let mut equat = Equation {
        coef_A: 0.0,
        coef_B: 0.0,
        coef_C: 0.0,
        Diskr: 0.0,
        result: TypeOfRes::Without,
    };
    equat.CoefsInRaw();
    print_Equation!(equat.coef_A, equat.coef_B, equat.coef_C);
    equat.calculation();
    let final_res = match equat.result {
        Without => format!("Корней нет"),
        OneSolv (root) => format!("Один корень => {}", root),
```

```
        TwoSolv {Root1, Root2} => format!("Два корня => {} и {}", Root1, Root2),
        ThreeSolv{Root1, Root2, Root3} => format!("Три корня => {} и {} и {}", Root1, Root2,
Root3),
        FourSolv{Root1, Root2, Root3, Root4} => format!("Четыре корня => {} и {} и {} и {}",
Root1, Root2, Root3, Root4)
    };
    println!("{}", final_res);
}

#[test]
fn Diskr1() {
    use TypeOfRes::*;
    let mut equat = Equation {
        coef_A: -4.0,
        coef_B: 16.0,
        coef_C: 0.0,
        Diskr: 0.0,
        result: TypeOfRes::Without,
    };
    equat.calculation();
    let mut flag: bool;
    if equat.Diskr == 256.0 {
        flag = true;
    }else{
        flag = false;
    }
    assert!(flag)
}

#[test]
fn Wrong_Diskr2() {
    use TypeOfRes::*;
    let mut equat = Equation {
        coef_A: 4.0,
        coef_B: 15.0,
        coef_C: 3.0,
        Diskr: 0.0,
        result: TypeOfRes::Without,
    };
    equat.calculation();
    let mut flag: bool;
    if equat.Diskr == 256.0 {
        flag = true;
    }else{
        flag = false;
    }
    assert!(flag)
}

#[test]
fn Diskr3() {
    use TypeOfRes::*;
```

```
    let mut equat = Equation {
        coef_A: 18.0,
        coef_B: 100.0,
        coef_C: 1.0,
        Diskr: 0.0,
        result: TypeOfRes::Without,
    };
    equat.calculation();
    let mut flag: bool;
    if equat.Diskr == 9928.0 {
        flag = true;
    }else{
        flag = false;
    }
    assert!(flag)
}

    #[test]
    fn result(){
        let mut equat = Equation {
            coef_A: 0.0,
            coef_B: -20.0,
            coef_C: 5.0,
            Diskr: 0.0,
            result: TypeOfRes::Without,
        };
        equat.calculation();
        let flag: bool;
        if let Equation{coef_A: 0.0, coef_B: -20.0, coef_C: 5.0, Diskr: 400.0, result:
TypeOfRes::TwoSolv{ Root1:0.5, Root2:-0.5}} = equat{
            flag = true
        }else{
            flag = false
        }
        assert!(flag)
    }

    #[test]
    fn result2(){
        let mut equat = Equation {
            coef_A: -4.0,
            coef_B: 16.0,
            coef_C: 0.0,
            Diskr: 0.0,
            result: TypeOfRes::Without,
        };
        equat.calculation();
        let flag: bool;
        if let Equation{coef_A: -4.0, coef_B: 16.0, coef_C: 0.0, Diskr: 256.0, result:
TypeOfRes::ThreeSolv{ Root1:0.0, Root2: 2.0, Root3: -2.0}} = equat{
            flag = true
        }else{
```

```
        flag = false
    }
    assert!(flag)
}
```

Результат работы:

```
    Finished dev [unoptimized + debuginfo] target(s) in 1.44s
     Running `target\debug\first_rust_project.exe`
Введите A:
-4
Введите B:
16
Введите C:
0
Введённое уравнение: -4x^4  +  16x^2 = 0
Три корня => 0 и 2 и -2
```

```
running 5 tests
test Diskr3 ... ok
test result ... ok
test Wrong_Diskr2 ... FAILED
test Diskr1 ... ok
test result2 ... ok
```