ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Петербургский государственный университет путей сообщения
Императора Александра I»
(ФГБОУ ВО ПГУПС)
Кафедра «Информационные и вычислительные системы»

Отчет
по практике
«Учебная ознакомительная практика»
для направления
09.03.01 «Информатика и вычислительная техника»
по профилю
«Программное обеспечение средств вычислительной техники и
автоматизированных систем»
(программа подготовки – бакалавриат)
Форма обучения – очная

Выполнил:
студент группы ИВБ-211
Шефнер А.
Руководитель практики:
Доцент, к.т.н. Баталов Д.И.

Санкт-Петербург
2023 г.

Оценочный лист результатов практики
Ф.И.О. Шефнер Альберт
Группа ИВБ-211

№	Материалы необходимые для оценки знаний, умений и навыков	Показатель оценивания	Критерии Оценивания	Шкала оценивания	Оценка
1	Задания по УОП 1–250	Количество правильно выполненных заданий	221– 250 выполненных заданий	71 – 80 баллов	
			191 – 220 выполненных заданий	61 – 70 баллов	
			161 – 190 выполненных заданий	51 – 60 баллов	
			128 – 160 выполненных заданий	41 – 50 баллов	
			96 – 127 выполненных заданий	31 – 40 баллов	
			64 – 95 выполненных заданий	21 – 30 баллов	
			32 – 63 выполненных заданий	11 – 20 баллов	
			0 – 31 выполненное задание	0 – 10 баллов	
		Итого максимальное количество баллов за 250 заданий по УОП		80	
ИТОГО				**80**	

Вид контроля	**Материалы, необходимые для оценки индикатора достижения компетенции**	**Максимальное количество баллов в процессе оценивания**	**Шкала Оценивания**	**Оценка**
1. Текущий контроль	Задания по УОП 1-250	80	Количество баллов определяется в соответствии с таблицей 3 Допуск к зачету/экзамену ≥ 50 баллов	
2. Промежуточная Аттестация	Перечень вопросов к зачету	20	– получены полные ответы на вопросы – 17 – 20 баллов; – получены достаточно полные ответы на вопросы – 12 – 16 баллов; – получены неполные ответы на вопросы или часть вопросов – 9 – 11 баллов; – не получены ответы на вопросы или вопросы не раскрыты – 0 – 8 баллов.	
ИТОГО		**100**		
3. Итоговая оценка	«зачтено» - 50-100 баллов; «не зачтено» - 49 баллов и менее.			

_____ _____

«_____ » _____
202____ г.

Оглавление

Цель и задачи

Цель:
Приобрести и закрепить практические знания в области программирования на языке Rust

Задачи:
1. Изучить лекционный материал.
2. Закрепить полученные знания, выполняя индивидуальные задания.

Задание

1. Установка среды разработки (Visual Studio)
2. Ознакомление со средой программирования на Visual Basic.
Выполнение индивидуального задания:
3. Лабораторная работа по разработке структуры Следование;
4. Лабораторная работа по разработке структуры Развилка;
5. Лабораторная работа по разработке структуры Цикл.
6. Оформление отчета о выполнении задания практики.

Лабораторная работа № 2

Вариант 19

1. Предложенные формулы записать в виде операторов присваивания. Числа представить в виде констант языка программирования, переменные по необходимости переобозначить.

$$m_{2p} = 22 \cdot 10^2 \qquad y = \sqrt[5]{d}$$

$$h = -48{,}3 \qquad p = y^2 + \frac{4x}{3}$$

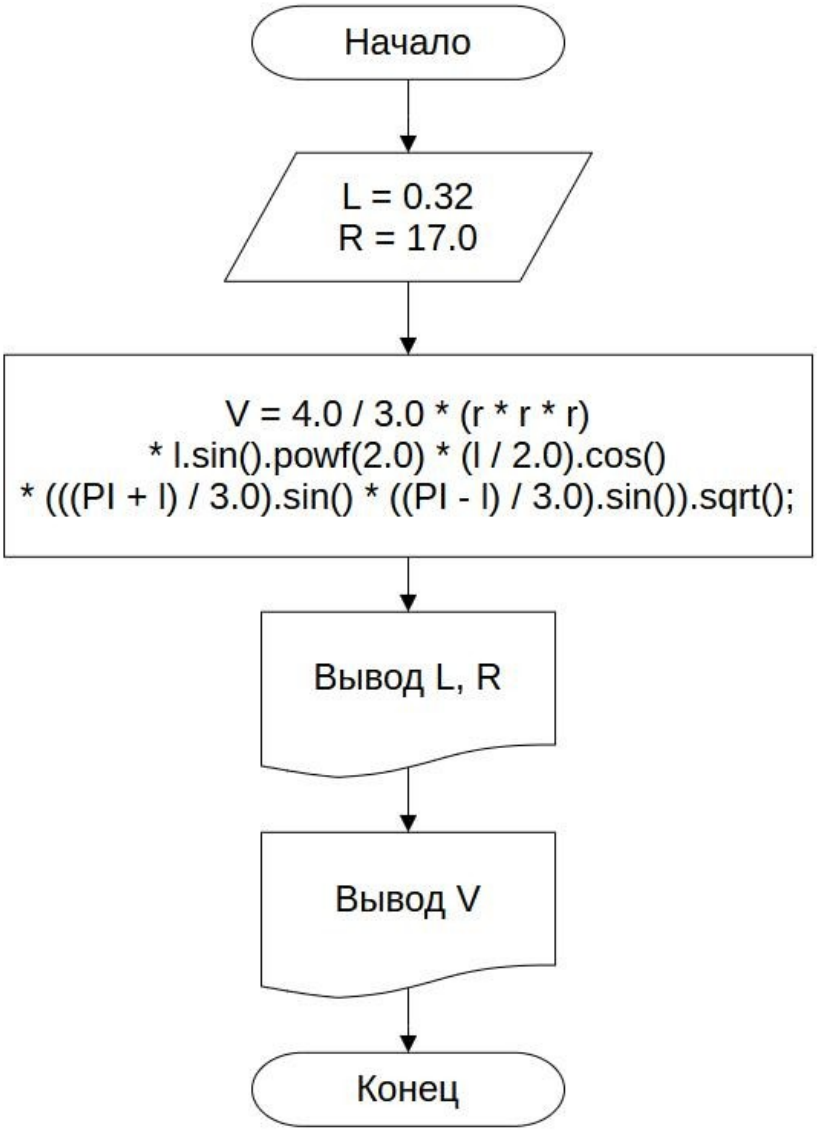
$$\sigma = 0{,}0005 \qquad t = |c + m| tgc^2$$

2. Подготовить задачу к решению на ЭВМ, выполнить постановку задачи, математическое описание, разработку алгоритма и программы. Рассчитать контрольный вариант по предложенным численным значениям входных данных и отладить программу.

$$V = \frac{4}{3} R^3 \sin^2 L \cdot Cos \frac{L}{2} \sqrt{Sin(\frac{\pi}{3} + \frac{L}{3}) \cdot Sin(\frac{\pi}{3} - \frac{L}{3}),}$$

Вычислить объем правильной треугольной пирамиды, зная, что плоский угол при вершине равен L, а радиус окружности, описанной около боковой грани, равен R.

Блок-схема алгоритма



Код программы

src > main.rs > ...

```rust
use chrono::Local;

▶ Run | Debug
fn main() {
    println!(
        "Albert Shefner Variant 19 {}\n",
        Local::now().format("%d.%m.%Y %H:%M:%S")
    );

    let m: f64 = 22e2;
    let h: f64 = -48.3;
    let sigma: f64 = 0.0005;
    let d: f64 = 1.0;
    let y: f64 = d.powf(1.0 / 5.0);
    let c: f64 = 2.0;
    let t: f64 = (c + m).abs() * (c * c).tan();

    println!("Constants:\nm = {m}\nh = {h}\nsigma = {sigma}\nd = {d}\nc = {c}\n");
    println!("Results:\ny = {y}\nt = {t}\n");
}
```

Листинг 2.1 — 1 пункт лабораторной работы №2.

src > main.rs > ...

```rust
use chrono::Local;
use std::f64::consts::PI;

▶ Run | Debug
fn main() {
    println!(
        "Albert Shefner Variant 19 {}\n",
        Local::now().format("%d.%m.%Y %H:%M:%S")
    );

    let l: f64 = 0.32;
    let r: f64 = 17.0;

    let v: f64 = 4.0 / 3.0
        * (r * r * r)
        * l.sin().powf(2.0)
        * (l / 2.0).cos()
        * (((PI + l) / 3.0).sin() * ((PI - l) / 3.0).sin()).sqrt();

    println!("Input:\nL = {l} rad\nR = {r} cm\n");
    println!("Result:\nV = {:.3} cm^3", v);
}
```

Листинг 2.2 — 2 пункт лабораторной работы №2.

На листинге 2.1 представлен код файла main.rs для первого пункта лабораторной работы №2. Для форматированного вывода текущей даты используется сторонняя библиотека (crate) chrono. Она же будет использована для всех последующих работ. Для всех переменных используется тип данных f64, являющийся реализацией стандарта IEEE 754 с размерностью 64 бит (аналог в Си — double). Для вычисления математических функций, таких как степень, модуль или тангенс, используется синтаксис методов (method syntax): функции powf, abs и tan являются методами встроенного типа f64. На листинге 2.2 представлен код файла main.rs для второго пункта. В качестве значения числа π используется встроенная константа типа f64.

```
Albert Shefner Variant 19 10.07.2023 14:13:41

Constants:
m = 2200
h = -48.3
sigma = 0.0005
d = 1
c = 2

Results:
y = 1
t = 2549.5224637337697
```

Листинг 2.3 — результат выполнения программы для первого пункта

```
Albert Shefner Variant 19 10.07.2023 14:28:18

Input:
L = 0.32 rad
R = 17 cm

Result:
V = 549.986 cm^3
```

Листинг 2.4 — результат выполнения программы для второго пункта

Лабораторная работа № 3

Вариант 10

Вычислить значение функции. Для вычисления значений функций необходимо определить требуемые входные и выходные данные, составить схемы алгоритмов и коды приложений. В первых примерах самостоятельно выбрать значение входных данных. Отладить коды приложений.

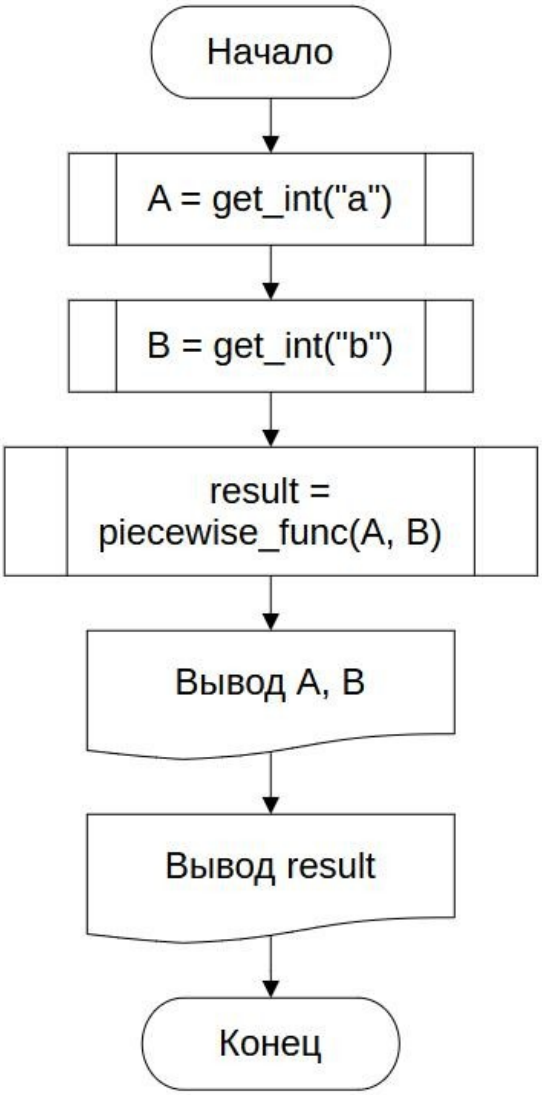
$$z = \begin{cases} (a+b)^{a-b}, & если \quad a > b \\ a^2 - b^a, & если \quad a = b \\ \dfrac{sin(\pi + a)}{b}, & если \quad a < b \end{cases}$$

где постоянная π = 3,14; a, b –переменные целого типа.

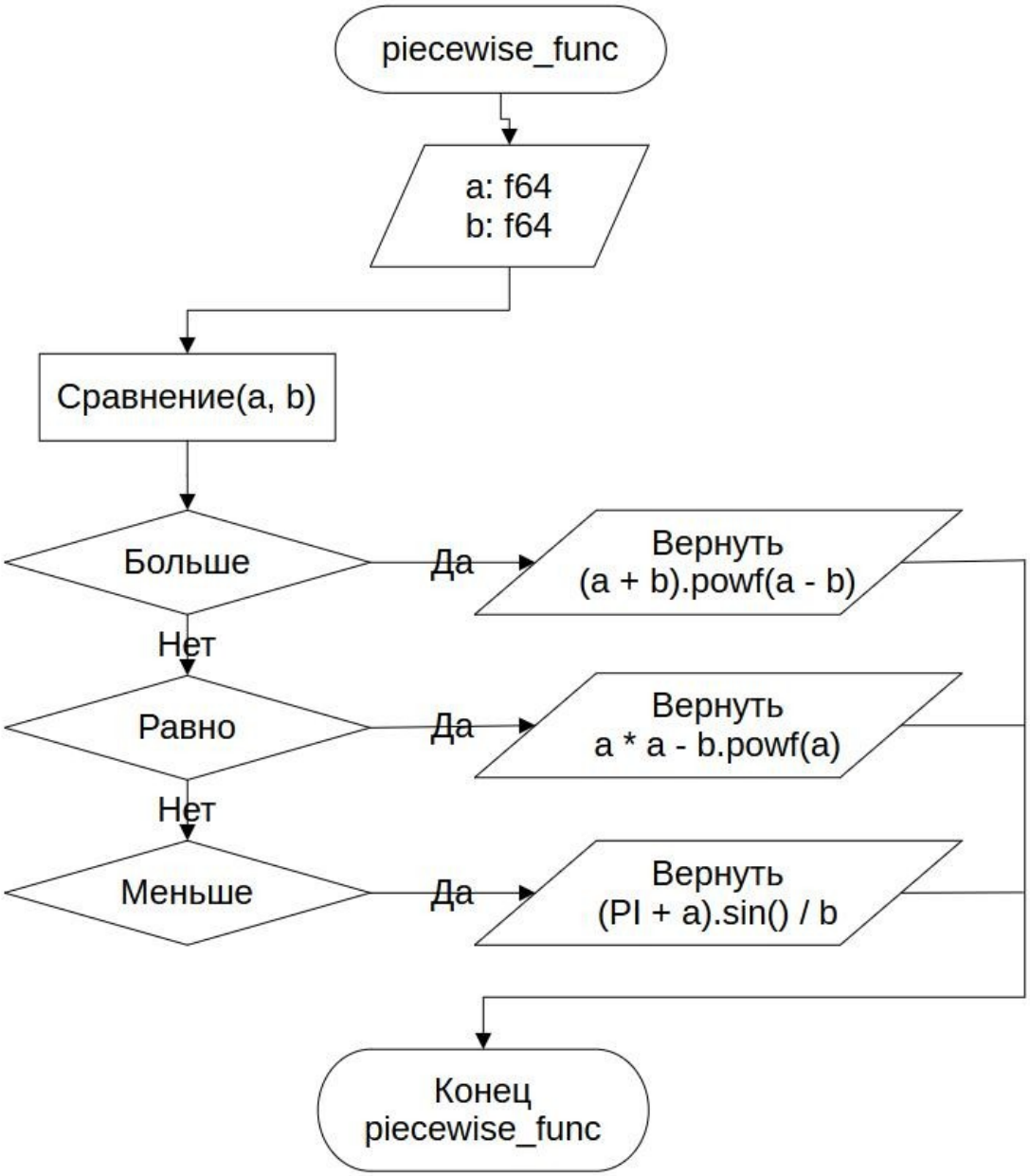
При решении контрольных примеров переменным присвоить значения:

1) a = 13; b = 2;
2) a = 3; b = 3;
3) a = 2; b = 5.

Блок-схема алгоритма




get_int
Name: &str
Int: String =
String::new()
Вывод "Enter {name}: "
Очистка буфера
Считывание
данных в
строку int
Преобразование
строки int в
число int
Возврат int
Конец get_int


piecewise_func
a: f64
b: f64
Сравнение(a, b)
Больше
Да
Вернуть
(a + b).powf(a - b)
Нет
Равно
Да
Вернуть
a * a - b.powf(a)
Нет
Меньше
Да
Вернуть
(PI + a).sin() / b
Конец
piecewise_func

Код программы

src > main.rs > ...

```rust
use chrono::Local;
use std::cmp::Ordering;
use std::io;
use std::io::Write;

const PI: f64 = 3.14;

fn get_int(name: &str) -> i32 {
    let mut int: String = String::new();
    print!("Enter {name}: ");
    io::stdout().flush().unwrap();
    io::stdin() Stdin
        .read_line(buf: &mut int) Result<usize, Error>
        .expect(msg: "Error reading string");
    let int: i32 = int.trim().parse().expect(msg: "Please enter a valid integer.");
    return int;
}

fn piecewise_func(a: f64, b: f64) -> f64 {
    match a.partial_cmp(&b).expect(msg: "Error comparing numbers") {
        Ordering::Greater => (a + b).powf(a - b),
        Ordering::Equal => a * a - b.powf(a),
        Ordering::Less => (PI + a).sin() / b,
    }
}

▶ Run | Debug
fn main() {
    println!(
        "Albert Shefner Variant 10 {}\n",
        Local::now().format("%d.%m.%Y %H:%M:%S")
    );

    let a: i32 = get_int(name: "a");
    let b: i32 = get_int(name: "b");
    let result: f64 = piecewise_func(a: f64::from(a), b: f64::from(b));

    println!("");
    println!("Input:\na = {a}\nb = {b}\n");
    println!("Result:\nf = {result}");
}
```

Листинг 3.1 — код программы для лабораторной работы №3.

На листинге 3.1 представлен код файла main.rs для лабораторной работы. Программа поделена на несколько логически полных процедур: основная процедура main, процедура для получения ввода с консоли get_int, процедура для вычисления функции piecewise_func. Программа получает на вход целые числа типа i32, однако для вычислений они конвертируются в вещественные числа типа f64. При этом значение функции powf для вычисления вещественной степени определено для отрицательных оснований, если показатель является целым числом.

```
Albert Shefner Variant 10 10.07.2023 16:14:49

Enter a: 13
Enter b: 2

Input:
a = 13
b = 2

Result:
f = 8649755859375
```

Листинг 3.2 — результат работы программы для чисел из примера 1)

```
Albert Shefner Variant 10 10.07.2023 16:16:46

Enter a: 3
Enter b: 3

Input:
a = 3
b = 3

Result:
f = -18
```

Листинг 3.3 — результат работы программы для чисел из примера 2)

```
Albert Shefner Variant 10 10.07.2023 16:17:30

Enter a: 2
Enter b: 5

Input:
a = 2
b = 5

Result:
f = -0.18199181021234206
```

Листинг 3.4 — результат работы программы для чисел из примера 3)

Лабораторная работа № 4

Вариант 3

Необходимо определить требуемые входные и выходные данные, для вычисления предложенных функций составить схемы алгоритмов и программы решения задач. Предусмотреть печать всех входных и выходных данных.Подготовить контрольные варианты (при необходимости самостоятельно выбрать значение входных данных), отладить программы.

Поезд, двигаясь под уклон, прошел за t секунд путь S и развил скорость V. Как изменяется ускорение поезда и какова была его скорость в начале уклона в зависимости от времени t?

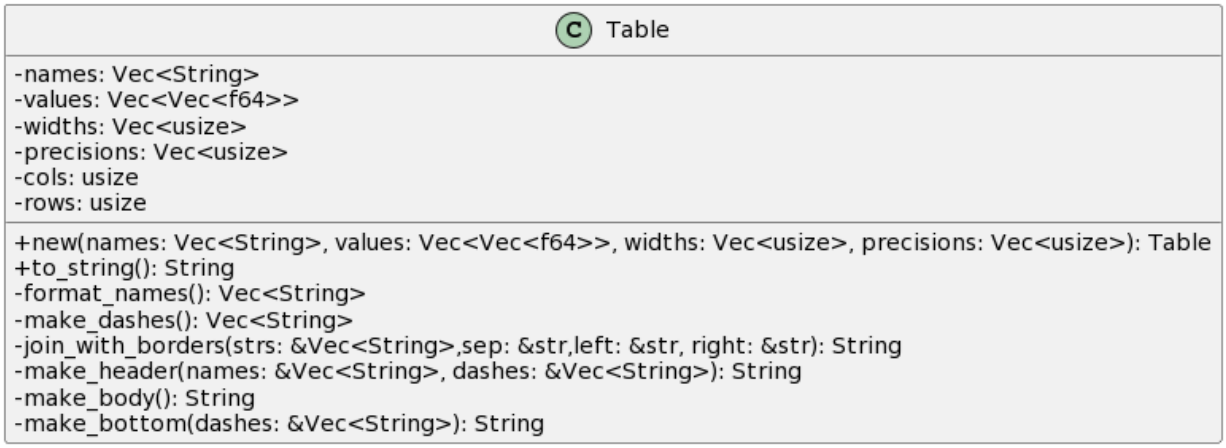
$$V_0 = \frac{2S}{t} - V \quad a = \frac{V - V_0}{t};$$

Где $S = 340\,м$; $V = 19\,м/с$; $15 \leqslant t \leqslant 25$; *с шагом* $1\,с$.

Блок-схема алгоритма

Основной алгоритм программы

UML-диаграмма структуры Table



Код программы

src > main.rs > ...

```rust
mod table;
use chrono::Local;
use table::Table;

const V: f64 = 19.0;
const S: f64 = 340.0;

fn v_t(t: f64) -> f64 {
    2.0 * S / t - V
}

fn a(t: f64, v_0: f64) -> f64 {
    (V - v_0) / t
}

▶ Run | Debug
fn main() {
    println!(
        "Albert Shefner Variant 3 {}\n",
        Local::now().format("%d.%m.%Y %H:%M:%S")
    );

    let t_list: Vec<f64> = (15..=25).map(|i: i32| f64::from(i)).collect();
    let v_t_list: Vec<f64> = t_list.iter().map(|t: &f64| v_t(*t)).collect();
    let a_list: Vec<f64> = t_list Vec<f64>
        .iter() Iter<'_, f64>
        .zip(v_t_list.iter()) impl Iterator<Item = (&f64, …)>
        .map(|(t: &f64, v_0: &f64)| a(*t, *v_0)) impl Iterator<Item = f64>
        .collect();

    let table: Table = Table::new(
        names: vec![
            String::from("t - время (c)"),
            String::from("V0 - нач. скорость (м/с)"),
            String::from("a - ускорение (м/с2)"),
        ],
        values: vec![t_list, v_t_list, a_list],
        widths: vec![13, 24, 20],
        precisions: vec![0, 10, 10],
    );

    let table_str: String = table.to_string();
    println!("{table_str}");
} fn main
```

Листинг 4.1 — файл main.rs для лабораторной работы №4.

```rust
pub struct Table {
    names: Vec<String>,
    values: Vec<Vec<f64>>,
    widths: Vec<usize>,
    precisions: Vec<usize>,
    cols: usize,
    rows: usize,
}
```

Листинг 4.2 — Объявление публичной структуры table в файле table.rs

```rust
impl Table {
    pub fn new(…
    ) -> Self {…

    pub fn to_string(&self) -> String {…

    fn format_names(&self) -> Vec<String> {…

    fn make_dashes(&self) -> Vec<String> {…

    fn join_with_borders(strs: &Vec<String>, sep: &str, left: &str, right: &str) -> String {…

    fn make_header(names: &Vec<String>, dashes: &Vec<String>) -> String {…

    fn make_body(&self) -> String {…

    fn make_bottom(dashes: &Vec<String>) -> String {…
} impl Table
```

Листинг 4.3 — Список методов имплементации структуры table в файле table.rs для лабораторной работы №4.

```rust
pub fn new(
    names: Vec<String>,
    values: Vec<Vec<f64>>,
    widths: Vec<usize>,
    precisions: Vec<usize>,
) -> Self {
    if names.is_empty() {
        panic!("Name vector should not be empty.");
    }

    if names.len() != values.len() {
        panic!("Name and value vectors should be the same len");
    }

    let cols: usize = names.len();
    let rows: usize = values[0].len();
```

```
    for val_col: &Vec<f64> in &values {
        if val_col.len() != rows {
            panic!("Value columns should be the same len.");
        }
    }

    Self {
        names,
        values,
        widths,
        precisions,
        cols,
        rows,
    }
} fn new
```

Листинг 4.4 — публичный метод new структуры table, выполняющий роль конструктора.

```
pub fn to_string(&self) -> String {
    let mut result: String = String::new();

    let names: Vec<String> = self.format_names();
    let dashes: Vec<String> = self.make_dashes();

    result.push_str(string: &Self::make_header(&names, &dashes));
    result.push_str(string: &Self::make_body(&self));
    result.push_str(string: &Self::make_bottom(&dashes));

    result
}
```

Листинг 4.5 — публичный метод to_string структуры table, преобразующий данные таблицы в одну строку.

```
fn format_names(&self) -> Vec<String> {
    (0..self.cols) Range<usize>
        .map(|i: usize| format!("{:>width$}", self.names[i], width = self.widths[i]))
        .collect()
}
```

Листинг 4.6 — закрытый метод format_names.

```
fn make_dashes(&self) -> Vec<String> {
    (0..self.cols) Range<usize>
        .map(|i: usize| format!("{:=>width$}", "", width = self.widths[i]))
        .collect()
}
```

Листинг 4.7 — закрытый метод make_dashes.

```rust
fn join_with_borders(strs: &Vec<String>, sep: &str, left: &str, right: &str) -> String {
    let mut result: String = String::from(left);
    result.push_str(string: &strs.join(sep));
    result.push_str(string: right);
    result
}
```

Листинг 4.8 — закрытый метод join_with_borders.

```rust
fn make_header(names: &Vec<String>, dashes: &Vec<String>) -> String {
    let mut header: String = Self::join_with_borders(strs: dashes, sep: "╤", left: "┌", right: "┐\n");
    header.push_str(string: &Self::join_with_borders(strs: names, sep: " ‖ ", left: "‖ ", right: " ‖\n"));
    header.push_str(string: &Self::join_with_borders(strs: dashes, sep: "╪", left: "╞", right: "╡\n"));
    header
}
```

Листинг 4.9 — закрытый метод make_header.

```rust
fn make_body(&self) -> String {
    let mut values: String = String::new();

    for i: usize in 0..self.rows {
        values.push_str(string: "‖ ");
        for j: usize in 0..self.cols {
            values.push_str(string: &format!(
                "{:width$.prec$} ‖ ",
                self.values[j][i],
                width = self.widths[j],
                prec = self.precisions[j]
            ));
        }
        values.pop();
        values.push_str(string: "\n");
    }

    values
}
```

Листинг 4.10 — закрытый метод make_body.

```rust
fn make_bottom(dashes: &Vec<String>) -> String {
    Self::join_with_borders(strs: &dashes, sep: "╧", left: "└", right: "┘\n")
}
```

Листинг 4.11 — закрытый метод make_bottom.

На листинге 4.1 представлен файл main.rs. Вычисление функций для разных значений t происходит посредством лямбда-исчисления. Результаты собираются в переменные типа Vec<T> - аналога контейнера std::vector<T> из языка C++. Далее по данным из этих контейнеров составляется таблица, представляющая из себя строку, и выводится на экран. Для реализации табличного форматирования создана структура Table с соответствующими методами, представленными в листингах 4.2 - 4.11.

Отладка приложения

Albert Shefner Variant 3 10.07.2023 17:07:01

t - время (с)	V0 - нач. скорость (м/с)	a - ускорение (м/с2)
15	26.3333333333	-0.4888888889
16	23.5000000000	-0.2812500000
17	21.0000000000	-0.1176470588
18	18.7777777778	0.0123456790
19	16.7894736842	0.1163434903
20	15.0000000000	0.2000000000
21	13.3809523810	0.2675736961
22	11.9090909091	0.3223140496
23	10.5652173913	0.3667296786
24	9.3333333333	0.4027777778
25	8.2000000000	0.4320000000

Листинг 4.12 — результат работы программы для лабораторной работы №4

Заключение

В ходе работы я приобрёл и закрепил практические знания в области программирования на языке Rust. Были выполнены задачи по изучению лекционного материала и закреплению полученных знаний, на основе выполнения индивидуальных заданий.

Рабочий график, содержание и планируемые результаты практики

Кафедра Информационные и вычислительные системы
Ф.И.О. студента
Шефнер Альберт
Группа ИВБ-211
Факультет Автоматизация и интеллектуальные технологии
Сроки практики по календарному учебному графику
с 29.06.2023 по 21.07.2023
Рабочий график и содержание практики

Содержание практики	Рабочий график
1. Установка среды разработки Visual Studio	с 29.06.2023 по 03.07.2023
2. Ознакомление со средой программирования на Visual Basic	с 04.07.2023 по 07.07.2023
3. Лабораторная работа по разработке структуры Следование	с 08.07.2023 по 11.07.2023
4. Лабораторная работа по разработке структуры Развилка	с 12.07.2023 по 15.07.2023
5. Лабораторная работа по разработке структуры Цикл	с 16.07.2023 по 18.07.2023
6. Оформление отчета о выполнении задания практики	с 19.07.2023 по 21.07.2023

Планируемые результаты	Отметка о полученных результатах
Установить среду разработки Visual Studio	
Ознакомиться со средой программирования на Visual Basic	
Выполнить индивидуальное задание: Лабораторная работа по разработке структуры Следование	
Выполнить индивидуальное задание: Лабораторная работа по разработке структуры Развилка	
Выполнить индивидуальное задание: Лабораторная работа по разработке структуры Цикл	
Оформить отчет о выполнении задания практики	

Руководитель от кафедры

_____

Обучающийся

_____