Защищено:
Гапанюк Ю.Е.

"__"_____**2025** г.

Демонстрация:
Константинов А.А.

"__"_____**2025** г.

Отчет по домашней работе по курсу Парадигмы и конструкции языков программирования

Тема работы: " Ранее неизвестный язык программирования. "

10
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5Ц-53Б _____
(подпись)

Константинов А.А. "__"_____**2025** г.

Москва, МГТУ - **2025**

Содержание

1. Описание задания

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).

2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.

3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.

4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).

5. В случае создания проекта необходимо детально комментировать код.

6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.

7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

2. Описание выбранного языка программирования

Язык программирования "Rust" - это современный системный язык, созданный для безопасного, параллельного и эффективного программирования. Он уникален тем, что сочетает низкоуровневый контроль над памятью и производительность языков вроде C/C++ с высокой безопасностью и удобством высокоуровневых языков.

Язык был начат как личный проект Грейдона Хора в Mozilla Research в 2006 году, а его первая стабильная версия вышла в 2015 году. Основная цель создания Rust - решить ключевую проблему системного программирования: исключить ошибки работы с памятью (такие как переполнение буфера, висячие указатели) и гонки данных в многопоточном коде, не жертвуя скоростью и контролем.

Главная особенность Rust - его система владения с правилами заимствования и временем жизни. На этапе компиляции она гарантирует, что в коде не будет обращений к несуществующей памяти или конфликтов доступа к данным. Это избавляет от необходимости в сборщике мусора.

Rust удобен и нужен для системного программирования: ОС, драйверы, встраиваемые системы, для высоконагруженных сервисов: Веб-движки, базы данных, игровые движки, где критичны скорость и надежность, для параллельного и параллельного программирования: Безопасная работа с потоками "из коробки", для интеграции с другим кодом: Возможность создания эффективных библиотек для Python, JS и т.д.

В общем, Rust предоставляет разработчикам уникальное преимущество: возможность писать быстрые и ресурсоэффективные программы с уровнем безопасности, который раньше был доступен только в управляемых языках вроде Java или C#. Его растущая экосистема (пакетный менеджер Cargo) и активное сообщество делают его мощным инструментом для создания надежного программного обеспечения будущего.

3. Программы, написанные на Rust и экранные формы

В рамках изучения дисциплины, связанной с современными языками программирования, был рассмотрен язык Rust и выполнен ряд практических заданий. Основной целью работы являлось освоение базовых возможностей языка Rust, понимание его синтаксиса, принципов работы с функциями, макросами, а также знакомство с модульным тестированием и инструментами сборки.

Язык программирования Rust относится к системным языкам и ориентирован на разработку высокопроизводительных и безопасных приложений. Одной из ключевых особенностей Rust является строгая система типов и механизм управления памятью без использования сборщика мусора, что позволяет предотвращать многие распространённые ошибки на этапе компиляции. В процессе выполнения лабораторных заданий были разработаны несколько программ, демонстрирующих различные возможности языка.

1. Базовый синтаксис и вывод данных

Программа: hello_world/src/main.rs

```rust
fn main() {
    println!("Hello, world!");
    let x = 5 + /* 90 + */ 5;
    println!("Is `x` 10 or 100? x = {}", x);
    println!("{} дней", 31);
    println!("{0}, это {1}. {1}, это {0}", "Алиса", "Боб");
}
```

Результат программы:

```
PS C:\Users\anton\OneDrive\Рабочий стол\ПКЯП_ДЗ\hello_world> cargo run
   Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.01s
    Running `target\debug\hello_world.exe`
Hello, world!
Is `x` 10 or 100? x = 10
31 дней
Алиса, это Боб. Боб, это Алиса
```

Данная программа является классическим примером знакомства с языком и демонстрирует основные элементы синтаксиса Rust. Функция main() служит точкой входа в программу и обязательна для исполняемых проектов.

Изученные концепции:

1. Точка входа программы — выполнение любой программы на Rust начинается с функции main()
2. Макросы вывода — println! используется для форматированного вывода данных в консоль
3. Переменные — объявляются с помощью ключевого слова let и по умолчанию являются неизменяемыми
4. Комментарии — Rust поддерживает как однострочные (//), так и блочные (/* */) комментарии
5. Форматирование строк — используются фигурные скобки {} и позиционные аргументы {0}, {1}

В результате выполнения программы в консоль выводятся строки с текстом и значениями переменных, что позволяет убедиться в корректности работы форматирования.

2. Функции и математические вычисления

Программа: lab/src/main.rs

```
use std::f64;

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

    // Решаем квадратное уравнение относительно y = x^2
    let discriminant = b * b - 4.0 * a * c;

    if discriminant < 0.0 {
        return solutions; // Нет действительных решений
    }

    // Находим корни уравнения для y = x^2
    let y1 = (-b + discriminant.sqrt()) / (2.0 * a);
    let y2 = (-b - discriminant.sqrt()) / (2.0 * a);

    // Находим x, если y >= 0
    if y1 >= 0.0 {
        solutions.push(y1.sqrt());
        solutions.push(-y1.sqrt());
    }
```

```rust
    if y2 >= 0.0 && (y2 - y1).abs() > f64::EPSILON {
        solutions.push(y2.sqrt());
        solutions.push(-y2.sqrt());
    }

    solutions
}

fn main() {
    let a = 1.0;
    let b = -5.0;
    let c = 4.0;

    let solutions = solve_biquadratic(a, b, c);
    if solutions.is_empty() {
        println!("Нет действительных решений");
    } else {
        println!("Решения биквадратного уравнения: {:?}", solutions);
    }
}
```

Результат программы:

```
PS C:\Users\anton\OneDrive\Рабочий стол\ПКЯП ДЗ\lab> cargo run
   Compiling lab v0.1.0 (C:\Users\anton\OneDrive\Рабочий стол\ПКЯП_ДЗ\lab)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.42s
     Running `target\debug\lab.exe`
Решения биквадратного уравнения: [2.0, -2.0, 1.0, -1.0]
PS C:\Users\anton\OneDrive\Рабочий стол\ПКЯП_ДЗ\lab>
⚠ 0   rust-analyzer
```

В данной программе реализована функция для решения биквадратного уравнения. Она демонстрирует работу с пользовательскими функциями, числами с плавающей точкой и динамическими структурами данных.

Изученные концепции:

1. Импорт стандартной библиотеки — использование use std::f64
2. Объявление функций — задание параметров и возвращаемого типа
3. Векторы (Vec) — динамические массивы для хранения результатов
4. Условные конструкции — if/else для обработки различных случаев
5. Математические функции — вычисление квадратного корня
6. Работа с числами с плавающей точкой — сравнение с использованием EPSILON

3. Макросы в Rust

Программа: lab_macros/src/main.rs

```
macro_rules! func
{
    ($a:expr)=>{
    $a*$a}
}


fn main() {
   let a=4;
   let sq = func!(a);
    println!("Квадрат {} равен {}",a,sq);
}
```

Результат программы:

```
PS C:\Users\anton\OneDrive\Рабочий стол\ПКЯП_ДЗ\lab_macros> cargo run
     Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.01s
      Running `target\debug\lab_macros.exe`
  Квадрат 4 равен 16
PS C:\Users\anton\OneDrive\Рабочий стол\ПКЯП_ДЗ\lab_macros>
```

Данный пример демонстрирует механизм макросов, который позволяет выполнять метапрограммирование на этапе компиляции.

Изученные концепции:

1. Создание макросов — использование macro_rules!
2. Шаблоны макросов — захват выражений с помощью expr
3. Подстановка кода — макрос разворачивается в обычное выражение
4. Повторное использование кода — упрощение типовых операций

4. Модульное тестирование

Программа: unit_test/src/main.rs

```
pub fn add(a: i32, b: i32) -> i32 {
    a + b
}

fn main() {
    let result = add(5, 3);
    println!("Результат: {}", result);
}

#[cfg(test)]
mod tests {
```

```rust
    use super::*;

    #[test]
    fn test_add_positive_numbers() {
        assert_eq!(add(2, 3), 5);
    }

    #[test]
    fn test_add_negative_numbers() {
        assert_eq!(add(-2, -3), -5);
    }

    #[test]
    fn test_add_mixed_numbers() {
        assert_eq!(add(-2, 3), 1); // ИСПРАВЛЕНО: -2 + 3 = 1 (было 2)
    }

    #[test]
    fn test_add_zero() {
        assert_eq!(add(0, 5), 5);
        assert_eq!(add(5, 0), 5);
    }
}
```

Результат программы:

```
PS C:\Users\anton\OneDrive\Рабочий стол\ПКЯП_ДЗ\unit_test> cargo test
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.01s
     Running unittests src\main.rs (target\debug\deps\unit_test-08ccd205f37bacbe.exe)

running 4 tests
test tests::test_add_negative_numbers ... ok
test tests::test_add_positive_numbers ... ok
test tests::test_add_zero ... ok
test tests::test_add_mixed_numbers ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s
```

В данном разделе изучена встроенная система модульного тестирования Rust, позволяющая автоматически проверять корректность работы функций.

5. Процесс компиляции и запуска

Для сборки и запуска всех проектов использовался стандартный инструмент Rust — Cargo, который автоматизирует компиляцию, управление зависимостями и запуск тестов.

cargo build (компилирует проект и проверяет код на ошибки, создавая исполняемый файл, но не запускает программу.)

cargo run (компилирует проект (если он ещё не собран) и сразу запускает получившуюся программу.)

cargo test запускает все модульные тесты проекта и выводит результаты их выполнения (успешные и с ошибками).

Выводы и заключение

В ходе выполнения практической работы был изучен язык программирования Rust и его основные возможности. Были освоены базовые конструкции языка, работа с функциями, макросами, векторами и системой модульного тестирования. Также получены навыки работы с инструментом Cargo.

Язык Rust продемонстрировал высокую строгость и надёжность, что делает его особенно подходящим для разработки безопасных и производительных приложений. Практические примеры показали, что Rust сочетает в себе современный синтаксис, мощную систему типов и развитую инфраструктуру, что подтверждает его актуальность и перспективность в сфере программной разработки.