Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по домашнему заданию

«Создание консольного приложения на языке Rust»

Выполнил:
студент группы РТ5-31Б
Серик И. Н.

Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.

Подпись и дата:

Москва, 2023 г.

Задание

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.

Основная часть

Rust — это компилируемый язык программирования общего назначения, объединяющий функциональное и процедурное программирование с объектной системой, основанной на типажах. Управление памятью осуществляется через механизм "владения" с использованием аффинных типов, что позволяет избежать системы сборки мусора во время выполнения программы. Одной из ключевых особенностей Rust является гарантированная безопасность работы с памятью благодаря встроенной в компилятор системе статической проверки ссылок, известной как "borrow checker". Язык также предоставляет средства для использования объектно-ориентированного программирования.

Основные приоритеты Rust включают безопасность, скорость и параллелизм, делая его подходящим для системного программирования, включая разработку операционных систем. Rust конкурентоспособен по скорости и возможностям с C++/C, но обеспечивает большую безопасность при работе с памятью за счет встроенных механизмов контроля ссылок. "Абстракции с нулевой стоимостью" способствуют повышению производительности программ на Rust.

Язык разрабатывается сообществом Mozilla Research с финансированием от Mozilla Foundation. В 2021 году была создана Rust Foundation, под управлением пятерых компаний-учредителей: AWS, Huawei, Google, Microsoft и Mozilla. Rust занимает первое место в рейтинге "Самые любимые языки программирования" Stack Overflow Developer Survey с 2016 по 2022 год.

Rust обладает несколькими преимуществами, которые делают его привлекательным для разработчиков:

Скорость и мощность:

Rust является высокопроизводительным языком, подходящим для создания сложных и высоконагруженных систем. Его возможность использования при написании операционных систем подчеркивает его мощь. Эффективное управление памятью делает его подходящим для встраиваемых систем и умных устройств.

Удобство использования:

Программисты, использующие Rust, часто выражают высокую удовлетворенность работой с этим языком. Создатели Rust учли ошибки предыдущих языков, стремясь сделать его удобным и приятным для разработчиков. Код на Rust компактный, легко читаемый, что облегчает жизнь программиста.

Безопасность:

Одной из ключевых особенностей Rust является внимание к безопасности кода. Rust предоставляет разработчикам "предохранитель" от ошибок, которые могут привести к неожиданным проблемам. Это снижает риск допущения ошибок и способствует более быстрому написанию кода.

Компактность и читаемость кода:

Код на Rust обычно является компактным и легко читаемым. Язык предоставляет много возможностей для улучшения опыта программирования и облегчения понимания кода.

Эти преимущества делают Rust привлекательным выбором для разработчиков, работающих над проектами с высокими требованиями к производительности, безопасности и удобству использования.

Универсальность:

Rust обеспечивает отличную работу с высокими уровнями абстракции, что позволяет разработчикам не беспокоиться о деталях низкоуровневого программирования, если это необходимо. В то же время язык предоставляет доступ к более низким уровням, таким как работа с "железом", процессами и памятью. Эта универсальность делает Rust подходящим для широкого спектра задач — как низкоуровневого, так и высокоуровневого программирования.

Практическая часть

В качестве примера кода рассмотрим реализацию консольной игры "Крестики-нолики" на языке программирования Rust.

Программа представляет собой простую реализацию классической игры "Крестики-нолики" для двух игроков. Игровое поле представлено в виде трех строк и трех столбцов. Каждый игрок в свой ход вводит координаты, указывающие, в какую ячейку он хочет поставить свой символ ('X' или 'O'). Игра продолжается до тех пор, пока один из игроков не выиграет или не наступит ничья.

Предлагаемое решение:

```rust
use std::io;
use std::process::Command;

// Функция для вывода игровой доски
fn print_board(board: &Vec<Vec<char>>) {
    // Очистка консоли
    if cfg!(windows) {
        let _ = Command::new("cmd").arg("/c").arg("cls").status();
    } else {
        let _ = Command::new("sh").arg("-c").arg("clear").status();
    }

    // Перерисовка обновленной игровой доски
    println!("  1 2 3");
    for (i, row) in board.iter().enumerate() {
        print!("{} ", i + 1);
        for &cell in row {
            print!("{} ", cell);
        }
        println!();
    }
}

// Проверка победителя
fn is_winner(board: &Vec<Vec<char>>, player: char) -> bool {
    // Проверка строк и столбцов
    for i in 0..3 {
        if (1..=3).all(|j| board[i][j - 1] == player) || (1..=3).all(|j| board[j - 1][i] ==
player) {
            return true;
        }
    }

    // Проверка диагоналей
    if (1..=3).all(|i| board[i - 1][i - 1] == player) || (1..=3).all(|i| board[i - 1][3 - i] ==
player) {
        return true;
    }

    false
}

// Проверка ничьей
```

```rust
fn is_draw(board: &Vec<Vec<char>>) -> bool {
    board.iter().all(|row| row.iter().all(|&cell| cell != ' '))
}

fn main() {
    loop {
        // Инициализация игровой доски
        let mut board = vec![
            vec![' ', ' ', ' '],
            vec![' ', ' ', ' '],
            vec![' ', ' ', ' '],
        ];

        let mut current_player = 'X';

        loop {
            print_board(&board);

            println!("Ход игрока {}", current_player);

            println!("Введите ход (например, 11 для строки 1, столбца 1): ");
            let mut input = String::new();
            io::stdin().read_line(&mut input).expect("Не удалось прочитать строку");

            // Обработка введенного хода
            let trimmed_input = input.trim();
            if trimmed_input.len() == 2 {
                if let (Some(row), Some(col)) =
(trimmed_input.chars().nth(0).unwrap().to_digit(10),
trimmed_input.chars().nth(1).unwrap().to_digit(10)) {
                    let (row, col) = (row as usize, col as usize);

                    // Проверка корректности введенных координат
                    if row >= 1 && row <= 3 && col >= 1 && col <= 3 {
                        // Проверка, что выбранная клетка свободна
                        if board[row - 1][col - 1] == ' ' {
                            // Установка символа игрока на доску
                            board[row - 1][col - 1] = current_player;

                            // Проверка победителя
                            if is_winner(&board, current_player) {
                                print_board(&board);
                                println!("Игрок {} выиграл!", current_player);
                                break;
                            }
```

```
                            // Проверка ничьей
                            if is_draw(&board) {
                                print_board(&board);
                                println!("Ничья!");
                                break;
                            }

                            // Смена хода игрока
                            current_player = if current_player == 'X' { 'O' } else { 'X' };
                        } else {
                            println!("Клетка уже занята. Попробуйте снова.");
                        }
                    } else {
                        println!("Неверный ввод. Пожалуйста, введите числа от 1 до 3.");
                    }
                }
            } else {
                println!("Неверный формат ввода. Пожалуйста, введите двузначное число.");
            }
        }

        // Вопрос о продолжении
        println!("Хотите сыграть еще раз? (y/n)");
        let mut choice = String::new();
        io::stdin().read_line(&mut choice).expect("Не удалось прочитать строку");
        if !choice.trim().eq_ignore_ascii_case("y") {
            break;
        }
    }

    println!("Нажмите Enter для выхода...");
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Не удалось прочитать строку");
}
```

1. *Вывод игровой доски (print_board)*

Функция использует стандартные библиотеки std::io и std::process::Command для вывода текущего состояния игровой доски в консоль. Она также осуществляет очистку консоли перед каждым выводом, обеспечивая наглядность игрового процесса.

2. *Проверка победы (is_winner)*

Функция определяет, есть ли победитель в текущей конфигурации доски. Проверяются все строки, столбцы и диагонали на наличие одинаковых символов ('X' или 'O').

3. *Проверка ничьей (is_draw)*

Эта функция проверяет, является ли текущее состояние доски ничейным, то есть заполненным без победителя.

4. *Основной игровой цикл (main)*

Основная функция программы содержит два вложенных цикла. Внешний цикл отвечает за повторение игры, а внутренний — за сам процесс ходов и ввода игроков. Ввод координат обрабатывается, проверяется на корректность и обновляется состояние доски. По окончании каждой партии программа предлагает пользователю сыграть еще раз. Пользователь может ввести 'y' для повторной игры или любой другой символ для завершения программы.

Таким образом, представленный код предоставляет полноценную консольную реализацию игры "Крестики-нолики" на языке программирования Rust.

Результаты

```
  1 2 3
1
2
3
Ход игрока X
Введите ход (например, 12 для строки 1, столбца 2):
11
```

```
  1 2 3
1 X
2
3
Ход игрока O
Введите ход (например, 12 для строки 1, столбца 2):
```

```
  1 2 3
1 X   O
2   X
3   O X
Игрок X выиграл!
Хотите сыграть еще раз? (y/n)
```

```
  1 2 3
1 O X X
2 X O O
3 X O X
Ничья!
Хотите сыграть еще раз? (y/n)
n
Нажмите Enter для выхода...
```