20CYS312 - PPL - Lab Exercise 4

Harishkar B
CH.EN.U4CYS22019

Github Link: github.com/itsharishkar/Rust3/upload

Objective:

The objective of this lab exercise is to develop a strong understanding of Rust programming fundamentals, including control flow (if-else, match), loops (for, while let), data structures (tuples, vectors, structs), iterators, and pattern matching. By implementing real-world scenarios, students gain hands-on experience in writing efficient and structured Rust code.

1. Nested Decision Making with if-else

o Write a Rust program that takes a person's age and income as input and determines their eligibility for a loan. The program should check: If the person is below 21, they are ineligible.
If between 21 and 60, they are eligible based on income (> ₹50,000).
If above 60, they need a guarantor

Code:

```rust
use std::io;
fn main() {
    let mut age = String::new();
    let mut income = String::new();
    println!("Enter your age:");
    io::stdin().read_line(&mut age).expect("Failed to read line");
    let age: u32 = age.trim().parse().expect("Please enter a valid number");
    println!("Enter your income:");
    io::stdin().read_line(&mut income).expect("Failed to read line");
    let income: u32 = income.trim().parse().expect("Please enter a valid number");
    if age < 21 {
        println!("You are ineligible for a loan.");
    } else if age >= 21 && age <= 60 {
        if income > 50000 {
            println!("You are eligible for a loan.");
        } else {
            println!("You are ineligible for a loan due to insufficient income.");
        }
    } else {
        println!("You need a guarantor to be eligible for a loan.");
    }
}
```

Explanation:

The program reads the user's age and income, then uses nested if-else conditions to determine loan eligibility. If under 21, the user is ineligible. If between 21-60, eligibility depends on having an income greater than ₹50,000. If above 60, a guarantor is required for loan approval.

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./vote
Enter your age:
20
Enter your income:
83000
You are ineligible for a loan.
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./vote
Enter your age:
28
Enter your income:
275000
You are eligible for a loan.
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./vote
Enter your age:
78
Enter your income:
120000
You need a guarantor to be eligible for a loan.
```

2. Using match with Complex Cases

o Implement a restaurant billing system where a user enters a menu item (e.g., "Burger", "Pizza", "Pasta"), and the program prints the price.

o Use a match expression with additional conditions to apply discounts based on the he user inputs a menu item (Burger, Pizza, or Pasta), and a match expression assigns a corresponding price. The program calculates the total cost based on quantity and applies a 10% discount if the order is more than 5 items quantity ordered.

Code:

```
use std::io;
fn main() {
    let mut item = String::new();
    let mut quantity = String::new();
    println!("Enter menu item (Burger, Pizza, Pasta):");
    io::stdin().read_line(&mut item).expect("Failed to read line");
    let item = item.trim().to_lowercase();
    println!("Enter quantity:");
    io::stdin().read_line(&mut quantity).expect("Failed to read line");
    let quantity: u32 = quantity.trim().parse().expect("Please enter a valid number");
    let price = match item.as_str() {
        "burger" => 200,
        "pizza" => 300,
        "pasta" => 150,
        _ => {
            println!("Item not found.");
            return;
        }
    };

    let total_price = price * quantity;
    let discount = if quantity > 5 { 0.1 } else { 0.0 }; // 10% discount for quantity > 5
    let discounted_price = total_price as f32 * (1.0 - discount);
    println!("Total price: ₹{:.2}", discounted_price);
}
```

Explanation:
This Rust program takes user input for a menu item (burger, pizza, or pasta) and quantity, calculates the total price based on predefined item prices, and applies a 10% discount if the quantity exceeds five. It reads input from the standard input, trims and converts the item name to lowercase, matches it with predefined prices, and calculates the total and discounted prices. If an invalid item is entered, it prints an error message and exits. The final price is displayed with two decimal places.

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./menu
Enter menu item (Burger, Pizza, Pasta):
Pizza
Enter quantity:
4
Total price: ₹1200.00
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./menu
Enter menu item (Burger, Pizza, Pasta):
pasta
Enter quantity:
8
Total price: ₹1080.00
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$
```

3. Using Loops for Data Processing
o Write a Rust program to generate Fibonacci numbers up to a given n using a for loop.
o Store the sequence in a list (vector) and print the values.
Code:

```rust
use std::io;
fn main() {
    let mut n = String::new();
    io::stdin().read_line(&mut n).expect("Failed to read line");
    let n: usize = n.trim().parse().expect("Please enter a valid number");

    let mut fibonacci = vec![0, 1];

    if n == 0 {
        println!("[]");
        return;
    } else if n == 1 {
        println!("[0]");
        return;
    }

    for i in 2..n {
        let next_fib = fibonacci[i - 1] + fibonacci[i - 2];
        fibonacci.push(next_fib);
    }
    println!("{:?}", fibonacci);
}
```

Explanation:
Using a for loop, the program generates Fibonacci numbers up to n terms, storing them in a vector. It starts with [0,1] and computes each new term by summing the previous two, printing the final sequence.

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./fibo
5
[0, 1, 1, 2, 3]
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./fibo
11
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./fibo
2
[0, 1]
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$
```

4. Pattern Matching in Loops with while let
o Implement a program where a user enters multiple numbers, and the program keeps adding them to a list until they enter 0.
o Use while let to process the list and print only the even numbers.

Code:

```
use std::io;
fn main() {
    let mut numbers = Vec::new();
    loop {
        let mut input = String::new();
        println!("Enter a number (0 to stop):");
        io::stdin().read_line(&mut input).expect("Failed to read line");

        let number: i32 = input.trim().parse().expect("Please enter a valid number");
        if number == 0 {
            break;
        }
        numbers.push(number);
    }

    while let Some(num) = numbers.pop() {
        if num % 2 == 0 {
            println!("Even number: {}", num);
        }
    }
}
```

Explanation:
The user enters numbers into a list until they input 0. The program then processes the list using while let, removing and printing only even numbers. This approach efficiently handles the list using

pattern matching.

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./let
Enter a number (0 to stop):
17
Enter a number (0 to stop):
12
Enter a number (0 to stop):
5
Enter a number (0 to stop):
43
Enter a number (0 to stop):
6
Enter a number (0 to stop):
0
Even number: 6
Even number: 12
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$
```

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./let
Enter a number (0 to stop):
1
Enter a number (0 to stop):
2
Enter a number (0 to stop):
3
Enter a number (0 to stop):
8
Enter a number (0 to stop):
7
Enter a number (0 to stop):
6
Enter a number (0 to stop):
12
Enter a number (0 to stop):
34
Enter a number (0 to stop):
0
Even number: 34
Even number: 12
Even number: 6
Even number: 8
Even number: 2
```

5. Tuple Manipulation in a Real-World Scenario
o Create a tuple representing an employee's data (ID, Name, Salary).
o Write a function that takes this tuple as input, applies a 10% salary hike if salary < ₹50,000, and returns an updated tuple.

Code:

```
fn apply_salary_hike(employee: (u32, String, f32)) -> (u32, String, f32) {
    let (id, name, salary) = employee;
    let updated_salary = if salary < 50000.0 {
        salary * 1.10 // 10% hike
    } else {
        salary
    };
    (id, name, updated_salary)
}

fn main() {
    let employee = (1, String::from("ashwin s"), 45000.0);
    let updated_employee = apply_salary_hike(employee);
    println!("Updated employee data: {:?}", updated_employee);
  let employee2= (2, String::from("aravind"), 55000.0);
    let updated_employee2 = apply_salary_hike(employee2);
    println!("Updated employee data2: {:?}", updated_employee2);
}
```

Explanation:

The program defines an employee's details as a tuple (ID, Name, Salary). A function checks the salary, applies a 10% raise if it is below ₹50,000, and returns an updated tuple with the revised tuple.

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./hike
Updated employee data: (1, "ashwin s", 49500.0)
Updated employee data2: (2, "aravind", 55000.0)
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$
```

6. Vector (List) Operations with Iterators
o Write a Rust program that maintains a list of temperatures recorded in a city for a week. Implement:
§ A function that finds the average temperature.
§ Another function that finds the highest and lowest temperature using iterators.

Code

```rust
fn average_temperature(temps: &[f32]) -> f32 {
    let sum: f32 = temps.iter().sum();
    sum / temps.len() as f32
}

fn find_high_low(temps: &[f32]) -> (f32, f32) {
    let (min, max) = temps.iter().fold((f32::MAX, f32::MIN), |(min, max), &x| {
        (min.min(x), max.max(x))
    });
    (min, max)
}

fn main() {
    let temperatures = vec![30.5, 32.0, 33.5, 31.0, 29.5, 34.0, 28.5];

    println!("Average temperature: {:.2}", average_temperature(&temperatures));

    let (min, max) = find_high_low(&temperatures);
    println!("Lowest temperature: {:.2}, Highest temperature: {:.2}", min, max);
}
```

Explanation:

A list of weekly temperatures is stored in a vector. One function calculates the average temperature using iter().sum(), while another finds the highest and lowest values using fold(), ensuring efficient data processing

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./temp
Average temperature: 31.29
Lowest temperature: 28.50, Highest temperature: 34.00
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$
```

7. Structs with Methods

o Define a struct named BankAccount with fields account_number, holder_name, and balance.

o Implement methods for:

§ Depositing money.

§ Withdrawing money (with balance check).

§ Displaying account details

Code:

```rust
struct BankAccount {
    account_number: u32,
    holder_name: String,
    balance: f32,
}
impl BankAccount {
    fn deposit(&mut self, amount: f32) {
        self.balance += amount;
    }

    fn withdraw(&mut self, amount: f32) {
        if self.balance >= amount {
            self.balance -= amount;
        } else {
            println!("Insufficient funds.");
        }
    }
    fn display(&self) {
        println!("Account Number: {}", self.account_number);
        println!("Account Holder: {}", self.holder_name);
        println!("Balance: ₹{:.2}", self.balance);
    }
}
fn main() {
    let mut account = BankAccount {
        account_number: 123456,
        holder_name: String::from("guru"),
        balance: 1000.0,
    };
   account.display();
    account.deposit(500.0);
    account.withdraw(200.0);
    account.display();
}
```

Explanation:

The BankAccount struct contains account details and balance. Methods allow depositing money, withdrawing (with a balance check), and displaying account information, ensuring proper account management.

Output:

```
Account Number: 123456  Account Holder: guru
Balance: ₹1000.00

Account Number: 123456
Account Holder: guru
Balance: ₹1500.00

Account Number: 123456
Account Holder: guru
Balance: ₹1300.00
```

8. Structs and Enums Together – Vehicle Registration System
o Define an enum named FuelType with variants Petrol, Diesel, and Electric.
o Define a struct named Vehicle with fields brand, model, and fuel_type.
o Implement a function that takes a list of vehicles and filters only electric vehicles for display

Code:

```rust
enum FuelType {
    Petrol,
    Diesel,
    Electric,
}
struct Vehicle {
    brand: String,
    model: String,
    fuel_type: FuelType,
}

fn filter_electric_vehicles(vehicles: Vec<Vehicle>) -> Vec<Vehicle> {
    vehicles.into_iter().filter(|v| matches!(v.fuel_type, FuelType::Electric)).collect()
}
fn main() {
    let vehicles = vec![
        Vehicle { brand: String::from("Tesla"), model: String::from("Model S"), fuel_type: FuelType::Electric },
        Vehicle { brand: String::from("Toyota"), model: String::from("Camry"), fuel_type: FuelType::Petrol },
         Vehicle { brand: String::from("BMW"), model: String::from("E39"), fuel_type: FuelType::Diesel },
           Vehicle { brand: String::from("MG"), model: String::from("Comet EV"), fuel_type: FuelType::Electric },
    ];


    let electric_vehicles = filter_electric_vehicles(vehicles);
    for vehicle in electric_vehicles {
        println!("Electric Vehicle: {} {}", vehicle.brand, vehicle.model);
    }
}
```

Explanation:

The FuelType enum defines different fuel types (Petrol, Diesel, Electric). The Vehicle struct stores vehicle details, and a function filters and prints only electric vehicles using Rust's filter() and matches! expressions.

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./ev
Electric Vehicle: Tesla Model S
Electric Vehicle: MG Comet EV
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$
```

Conclusion:

This lab provided hands-on experience in writing structured and efficient Rust programs. It reinforced key programming paradigms, including decision-making, iteration, and object-oriented approaches using structs and enums. By completing these exercises, students gained practical knowledge of Rust's syntax and functionality, preparing them for more advanced applications in system and application development.