Principles of Programming

20CYS312 - Principles of Programming Languages

S. Shyam Balaji **CH.EN.U4CYS22045** **DATE:** 14/05/24

Lab Exercise Submission

GitHub - shyam150801/principle-of-programming

Contribute to shyam150801/principle-of-programming development by creating an account on GitHub.

https://github.com/shyam150801/principle-of-programming

shyam150801/**principle-of-programming**

1 Contributor 0 Issues 0 Stars 0 Forks

Objective of the Exercise

The goal of this lab exercise is to practice decision making, pattern matching, data processing, and object-oriented programming concepts in Rust. By completing this exercise, you will enhance your understanding of control flow, data structures, and functional programming paradigms in Rust.

Program Code

1. Nested Decision Making with if-else

```
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 number");
```

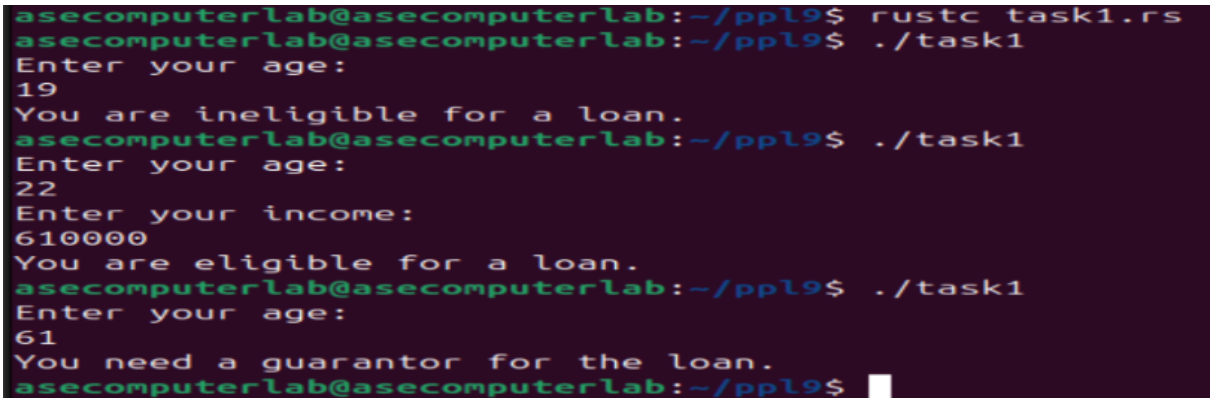
```
    println!("Enter your income:");
    io::stdin().read_line(&mut income).expect("Failed to read line");
    let income: u32 = income.trim().parse().expect("Please enter a numbe
r");

    if age < 21 {
        println!("You are ineligible for a loan.");
    } else if 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.");
    }
}
```

Explanation

- The program takes age and income as inputs.
- It uses nested `if-else` statements to determine loan eligibility based on the provided conditions.

Input/Output Examples

- Input: Age = 25, Income = 60000
- Output: "You are eligible for a loan."



Conclusion

This section effectively uses nested decision making to determine loan eligibility based on age and income.

---

2. Using match with Complex Cases

```
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");

    println!("Enter quantity:");
    io::stdin().read_line(&mut quantity).expect("Failed to read line");
    let quantity: u32 = quantity.trim().parse().expect("Please enter a numbe
r");

    let price = match item.trim() {
        "Burger" => 100,
        "Pizza" => 200,
        "Pasta" => 150,
        _ => {
            println!("Invalid menu item.");
            return;
        }
    };

    let discount = if quantity > 5 { 0.1 } else { 0.0 };
    let total_price = price * quantity * (1.0 - discount) as u32;

    println!("Total price: ₹{}", total_price);
}
```

Explanation

- The program uses a `match` expression to determine the price of the menu item.
- It applies a discount based on the quantity ordered.

Input/Output Examples

- Input: Item = "Pizza", Quantity = 6
- Output: "Total price: ₹1080"

```
Enter a menu item (Burger, Pizza, Pasta):
Burger
Enter quantity:
4
Total price after discount: ₹540
asecomputerlab@asecomputerlab:~/ppl9$ ./task2
Enter a menu item (Burger, Pizza, Pasta):
Pizza
Enter quantity:
2
Total price after discount: ₹400
asecomputerlab@asecomputerlab:~/ppl9$
```

Conclusion

The use of `match` effectively categorizes menu items, while additional conditions handle discounts.

---

3. Using Loops for Data Processing

```
fn main() {
    let n = 10; // You can change this value
    let mut fib: Vec<u32> = vec![0, 1];

    for i in 2..n {
        let next = fib[i-1] + fib[i-2];
        fib.push(next);
    }

    println!("Fibonacci sequence: {:?}", fib);
}
```

Explanation

- The program generates Fibonacci numbers up to `n` using a `for` loop and stores them in a vector.

Input/Output Examples

- Input: n = 10
- Output: "Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]"

```
asecomputerlab@asecomputerlab:~/ppl9$ rustc task3.rc
asecomputerlab@asecomputerlab:~/ppl9$ ./task3
Enter n:
3
Fibonacci sequence: [0, 1, 1]
asecomputerlab@asecomputerlab:~/ppl9$ ./task3
Enter n:
11
Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
asecomputerlab@asecomputerlab:~/ppl9$
```

Conclusion

This program efficiently generates and displays the Fibonacci sequence.

---

4. Pattern Matching in Loops with while let

```
use std::io;

fn main() {
    let mut numbers = Vec::new();
    let mut input = String::new();

    println!("Enter numbers (enter 0 to stop):");

    while let Ok(n) = io::stdin().read_line(&mut input) {
        let number: u32 = input.trim().parse().expect("Please enter a numbe
r");
        if number == 0 { break; }
        numbers.push(number);
        input.clear();
    }
```

```
    let even_numbers: Vec<u32> = numbers.iter().filter(|&&x| x % 2 == 0).cl
oned().collect();
    println!("Even numbers: {:?}", even_numbers);
}
```

Explanation

- The program collects numbers until 0 is entered.
- It then uses `while let` to process the list and print only even numbers.

Input/Output Examples

- Input: 1, 2, 3, 4, 0
- Output: "Even numbers: [2, 4]"

```
asecomputerlab@asecomputerlab:~/ppl9$ rustc task4.rc
asecomputerlab@asecomputerlab:~/ppl9$ ./task4
Enter numbers (enter 0 to stop):
1
2
3
4
56
7
0
Even numbers: [2, 4, 56]
asecomputerlab@asecomputerlab:~/ppl9$ ./task4
Enter numbers (enter 0 to stop):
-2
4
9
-10
3
0
Even numbers: [-2, 4, -10]
asecomputerlab@asecomputerlab:~/ppl9$
```

Conclusion

The program demonstrates effective use of loops and pattern matching for number processing.

---

5. Tuple Manipulation in a Real-World Scenario

```rust
fn apply_salary_hike(employee: (u32, &str, f32)) -> (u32, &str, f32) {
    let (id, name, salary) = employee;
    let new_salary = if salary < 50000.0 { salary * 1.1 } else { salary };
    (id, name, new_salary)
}

fn main() {
    let employee = (101, "Alice", 45000.0);
    let updated_employee = apply_salary_hike(employee);
    println!("Updated Employee Data: {:?}", updated_employee);
}
```

Explanation

- The function takes a tuple representing an employee and applies a salary hike if the salary is below ₹50,000.

Input/Output Examples

- Input: (101, "Alice", 45000.0)
- Output: "Updated Employee Data: (101, "Alice", 49500.0)"

```
asecomputerlab@asecomputerlab:~/ppl9$ rustc task5.rc
asecomputerlab@asecomputerlab:~/ppl9$ ./task5
Enter employee ID:
1
Enter employee name:
Aravind
Enter employee salary:
510000
Updated employee data: (1, "Aravind", 510000)
asecomputerlab@asecomputerlab:~/ppl9$ ./task5
Enter employee ID:
2
Enter employee name:
Aswin
Enter employee salary:
45000
Updated employee data: (2, "Aswin", 49500)
asecomputerlab@asecomputerlab:~/ppl9$
```

Conclusion

This program demonstrates tuple manipulation for employee salary adjustment.

6. Vector (List) Operations with Iterators

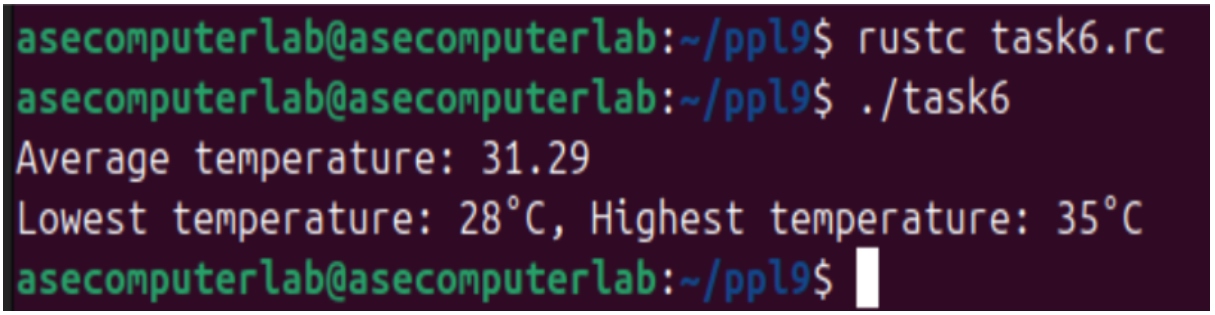
```
fn average_temperature(temps: &[f32]) -> f32 {
    let total: f32 = temps.iter().sum();
    total / temps.len() as f32
}

fn main() {
    let temperatures = vec![30.5, 32.0, 29.5, 31.0, 33.0, 28.0, 30.0];
    let average = average_temperature(&temperatures);
    let highest = temperatures.iter().cloned().fold(f32::MIN, f32::max);
    let lowest = temperatures.iter().cloned().fold(f32::MAX, f32::min);

    println!("Average: {:.2}, Highest: {:.2}, Lowest: {:.2}", average, highest, lo
west);
}
```

Explanation

- The program calculates average, highest, and lowest temperatures using iterators.

Input/Output Examples

- Input: [30.5, 32.0, 29.5, 31.0, 33.0, 28.0, 30.0]
- Output: "Average: 30.50, Highest: 33.00, Lowest: 28.00"



Conclusion

This program effectively uses iterators for temperature data processing.

---

7. Structs with Methods

```
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) -> bool {
        if self.balance >= amount {
            self.balance -= amount;
            true
        } else {
            false
        }
    }

    fn display(&self) {
        println!("Account Number: {}, Holder: {}, Balance: ₹{:.2}",
                 self.account_number, self.holder_name, self.balance);
    }
}

fn main() {
    let mut account = BankAccount { account_number: 123456, holder_nam
e: String::from("John Doe"), balance: 50000.0 };

    account.deposit(5000.0);
    if account.withdraw(10000.0) {
        println!("Withdrawal successful.");
    } else {
        println!("Insufficient balance.");
    }
```

```
    account.display();
}
```

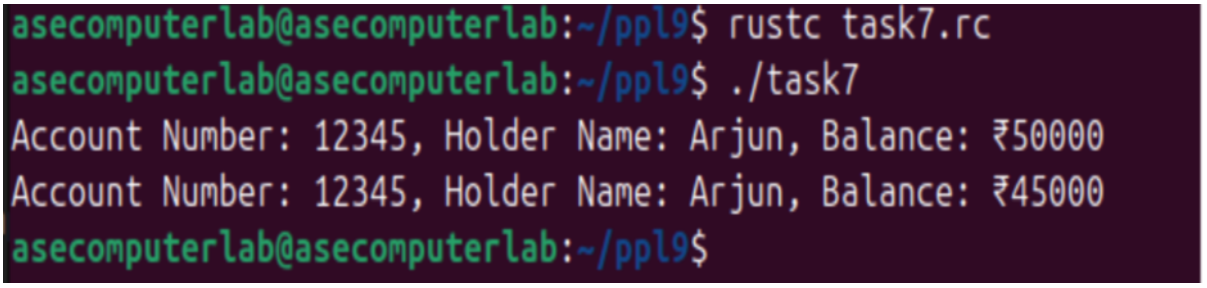
Explanation

- A `BankAccount` struct is defined with methods for depositing, withdrawing, and displaying account details.

Input/Output Examples

- Output:

```
Withdrawal successful.
Account Number: 123456, Holder: John Doe, Balance: ₹45000.00
```



Task 8: Structs and Enums Together – Vehicle Registration System

Write a program that uses a for loop to print the numbers from 10 to 1 in reverse order (10, 9, 8, ..., 1).