Rust - Лабораторная работа № 1

Задание 1

Постановка задачи:

Напишите программу, которая запрашивает у пользователя имя и выводит на экран приветственное сообщение с использованием этого имени.

Математическая модель:

-

Список идентификаторов:

Имя	**Тип**	**Назначение**
name	Mut String	Вводимое имя.

Код программы:

```rust
use std::io;

fn main()
{
    println!("Enter your name:");
    let mut name = String::new();
    io::stdin()
        .read_line(&mut name)
        .expect("Failded to read line");
    println!("Hello, {}!", name);
}
```

Результат выполненной работы:

```
Enter your name:
Maxwel
Hello, Maxwel!
```

Задание 2

Постановка задачи:

Создайте переменную типа целое беззнаковое число и выведите ее значение на экран. Явно укажите тип переменной. Затем измените значение переменной и снова выведите его.

Математическая модель:

-

Список идентификаторов:

Имя	Тип	Назначение
x	Mut u32	Переменная с типом целое беззнаковое число.

Код программы:

```
fn
main()
{
    let mut x : u32 = 3;
    println!("{}", x);
    x +=x;
    println!("{}", x);
}
```

Результат выполненной работы:

```
3
6
```

Задание 3

Постановка задачи:

Напишите функцию, которая принимает строку и возвращает ее длину (количество символов). Затем вызовите эту функцию с различными строками.

Математическая модель:

-

Список идентификаторов:

Имя	Тип	Назначение
s1	String	Строка для поверки функции 1.
s2	String	Строка для поверки функции 2.
s3	String	Строка для поверки функции 3.
lenght	Mut u64	Длинна строки.
_	char	Аргумент цилка.

Код программы:

```rust
fn main()
{
    let s1 = "Hello world!";
    let s2 = "Goodbye world!";
    let s3 = "rust";
    println!("len of {} = {}", s1, lenght(s1));
    println!("len of {} = {}", s2, lenght(s2));
    println!("len of {} = {}", s3, lenght(s3));
}

fn lenght(s: &str) -> usize
{
    let mut length = 0;
    for _ in s.chars() {
        length += 1;
    }
    length
}
```

Результат работы:

```
len of Hello world! = 12
len of Goodbye world! = 14
len of rust = 4
```

Задание 4

Постановка задачи:

Задайте структуру Car с полями brand, model и year, и создайте несколько экземпляров этой структуры. Выведите информацию о каждой машине на экран.

Математическая модель:

-

Список идентификаторов:

Имя	**Тип**	**Назначение**
car1	Car	Объект структуры Car для вывода информации о нем.
car2	Car	Объект структуры Car для вывода информации о нем.
car3	Car	Объект структуры Car для вывода информации о нем.

Код программы:

```rust
struct Car { brand:String, model:String, year:u16}

fn main()
{
    let car1 = Car{
        model: String::from("Model S"),
        brand: String::from("Tesla "),
        year: 2020
    };
    let car2 = Car{
        model: String::from("Model 3"),
        brand: String::from("Tesla"),
        year: 2017
    };
    let car3 = Car{
        model: String::from("Model X"),
        brand: String::from("Tesla"),
        year: 2015
    };
    print_car_info(car1);
    print_car_info(car2);
    print_car_info(car3);
}

fn print_car_info(car:Car)
{
    println!("{} {} made in {}.", car.brand, car.model, car.year);
}
```

Результат работы:

```
Tesla Model S made in 2020.
Tesla Model 3 made in 2017.
Tesla Model X made in 2015.
```

Задание 5

Постановка задачи:

Напишите программу, которая запрашивает у пользователя число N и выводит на экран N ное число Фибоначчи. Используйте рекурсию для решения этой задачи.

Математическая модель:

-

Список идентификаторов:

Имя	Тип	Назначение
curr_n	u64	i
prev_n	u64	i-1
recur	u64	Счётчик вызовов функуции.
end	u64	N полученное от пользоваеля.

Код программы:

```rust
use std::io;

fn main(){
    let mut Ns = String::new();
    io::stdin()
        .read_line(&mut Ns)
        .expect("Failded to read line");
    Ns.pop();
    let N = Ns.parse::<u64>().unwrap();
    println!("{}", fib(1, 0, 1, N));
}

fn fib(curr_n:u64, prev_n:u64, recur:u64, end:u64) -> u64{
    if recur == end
    {
    return curr_n;
    }
    fib(curr_n+prev_n, curr_n, recur+1, end)
}
```

Результат работы:

```
8
21
```

Задание 6

Постановка задачи:

Реализуйте перечисление DayOfWeek для дней недели. Напишите функцию, которая принимает день недели и возвращает следующий день. Обработайте случаи перехода на следующий день недели, если текущий день – воскресенье.

Математическая модель:

-

Список идентификаторов:

Имя	Тип	Назначение
today	DayOfWeek	«Сегодня» для нахождения «завтра» с помощью функции.
tomorrow	DayOfWeek	Найденное с помощью функции «завтра».

Код программы:

```
enum DayOfWeek {
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday,
}

fn main()
{
    let today:DayOfWeek = DayOfWeek::Monday;

    let tomorrow:DayOfWeek = get_next_day(today);
}

fn get_next_day(curr_day:DayOfWeek) -> DayOfWeek
{
    match curr_day{
        DayOfWeek::Monday => return DayOfWeek::Thursday,
        DayOfWeek::Tuesday=> return DayOfWeek::Wednesday,
        DayOfWeek::Wednesday=> return DayOfWeek::Thursday,
        DayOfWeek::Thursday=> return DayOfWeek::Friday,
        DayOfWeek::Friday=> return DayOfWeek::Saturday,
        DayOfWeek::Saturday=> return DayOfWeek::Sunday,
        DayOfWeek::Sunday=> return DayOfWeek::Monday,
    };
}
```

Результат работы:

-

Задание 7

Постановка задачи:

Создайте структуру Product с полями name, price и category, а также перечисление (enum) Category для категорий товаров. Напишите метод для вывода информации о продукте и ассоциированную функцию для подсчета общей суммы товаров в заданной категории из

массива продуктов.

Математическая модель:

-

Список идентификаторов:

Имя	**Тип**	**Назначение**

Код программы:

```rust
#[derive(PartialEq)]
enum Category{
    Cpu,
    Gpu,
    Psu,
    Case,
    Ram,
    Storage,
    Generic
}

impl Category{
    fn get_string_category(&self) -> &str
    {
        match self{
           Category::Cpu => return "cpu",
           Category::Gpu => return "gpu",
           Category::Psu => return "psu",
           Category::Case => return "case",
           Category::Ram => return "ram",
           Category::Storage => return "storage",
           Category::Generic=> return "unknown",
        }
    }
    fn total_price(products:&[Product], category:Category) -> f64
    {
        let mut total_price:f64 = 0.0;
       for item in products{
           if item.category == category{
               total_price+=item.price;
           }
       }
        total_price
    }
}
```

```rust
struct Product{
    name:String,
    price:f64,
    category:Category
}

impl Product{
    fn print_info(&self)
    {
        println!("Product: {}\n\tPrice: {}\n\tCategory{}", self.name, self.price, self.category.get_string_category());
    }
}


fn main()
{
    let products = vec![
        Product {
            name: String::from("CPU"),
            price: 100.0,
            category: Category::Cpu,
        },
        Product {
            name: String::from("GPU"),
            price: 200.0,
            category: Category::Gpu,
        },
    ];
    let cpu_price = Category::total_price(&products, Category::Cpu);
    println!("Total price for CPU: {}", cpu_price);

    let gpu_price = Category::total_price(&products, Category::Gpu);
    println!("Total price for GPU: {}", gpu_price);
}
```

Результат работы:

```
Total price for CPU: 100
Total price for GPU: 200
```

Задание 8

Постановка задачи:

Создайте структуру Employee с полями name, position, salary, а также перечисление Position для должностей сотрудников. Напишите функцию, которая принимает вектор сотрудников и возвращает вектор сотрудников заданной должности.

Математическая модель:

-

Список идентификаторов:

Имя	**Тип**	**Назначение**
emplouees	vec	
managers	vec	
developers	vec	

Код программы:

```rust
#[derive(PartialEq)]
enum Position{
    Manager,
    Developer,
    Designer,
}
impl Clone for Position{
    fn clone(&self) -> Self
    {
        match self{
            Position::Manager => Position::Manager,
            Position::Developer => Position::Developer,
            Position::Designer => Position::Designer,
        }
    }
}
impl Position{
    fn position_to_string(&self ) -> String
    {
    match self{
            Position::Manager => "manager".to_string(),
            Position::Developer => "developer".to_string(),
            Position::Designer => "developer".to_string(),
        }
    }
}
struct Employee{
    name:String,
    position:Position,
    salary:f64
}

impl Employee{
    fn clone(&self) -> Self{
    Employee {
        name: self.name.clone(),
        position: self.position.clone(),
        salary: self.salary

        }
    }
}
```

```rust
fn employed_on_position(employees:&Vec<Employee>, position:Position) -> Vec<Employee>
    {
        let mut on_picked_position:Vec<Employee> = Vec::new();

        for employee in employees{
            if employee.position == position{
                on_picked_position.push(employee.clone());
            }
        }
        on_picked_position
    }

fn main()
{
       let employees = vec![
        Employee {
            name: String::from("Alice"),
            position: Position::Manager,
            salary: 1200.0,
        },
        Employee {
            name: String::from("Bob"),
            position: Position::Developer,
            salary: 1200.0,
        },
        Employee {
            name: String::from("Charlie"),
            position: Position::Designer,
            salary: 1200.0,
        }];
        let managers = employed_on_position(&employees, Position::Manager);
        let developers= employed_on_position(&employees, Position::Developer);

        println!("Managers:");
        for employee in managers {
            println!("\t{} - {} - {}", employee.name, employee.position.position_to_string(), employee.salary);
        }
        println!("Developers:");
        for employee in developers{
            println!("\t{} - {} - {}", employee.name, employee.position.position_to_string(), employee.salary);
        }
}
```

Результат работы:

```
Managers:
        Alice - manager - 1200
Developers:
        Bob - developer - 1200
```