9. Basics of Rust:

A) Program to display statements:

```
fn main() {
   println!("Hello, Rust!");
}
```

B) Program to demonstrate basic data types in Rust:

```
fn main() {
   let integer_num: i32 = 42;
   let float_num: f64 = 3.14;
   let is_rust_fun: bool = true;
   let greeting: &str = "Hello, Rust!";

   println!("Integer: {}", integer_num);
   println!("Float: {}", float_num);
   println!("Boolean: {}", is_rust_fun);
   println!("String: {}", greeting);
}
```

C) Program to format strings and numbers:

```
fn main() {
   let price = 19.99;
   let quantity = 5;

   println!("Total: ${:.2}", price * quantity);
}
```

D) Program to compute arithmetic operations with user input:

```
use std::io;

fn main() {
   println!("Enter two numbers:");

   let mut input = String::new();
   io::stdin().read_line(&mut input).expect("Failed to read line");
   let num1: i32 = input.trim().parse().expect("Invalid input");

   let mut input = String::new();
   io::stdin().read_line(&mut input).expect("Failed to read line");
```

```
    let num2: i32 = input.trim().parse().expect("Invalid input");

    println!("Sum: {}", num1 + num2);
    println!("Difference: {}", num1 - num2);
    println!("Product: {}", num1 * num2);
    println!("Quotient: {}", num1 / num2);
}
```

E) Program to demonstrate bitwise and logical operators:

```
fn main() {
    let a = 0b1010; // 10 in binary
    let b = 0b1100; // 12 in binary

    let and_result = a & b; // Bitwise AND
    let or_result = a | b;  // Bitwise OR
    let xor_result = a ^ b; // Bitwise XOR

    let logical_and = true && false; // Logical AND
    let logical_or = true || false;  // Logical OR

    println!("AND: {:04b}", and_result);
    println!("OR: {:04b}", or_result);
    println!("XOR: {:04b}", xor_result);
    println!("Logical AND: {}", logical_and);
    println!("Logical OR: {}", logical_or);
}
```

F) Program to swap two numbers without a temporary variable:

```
fn main() {
    let mut num1 = 5;
    let mut num2 = 10;

    num1 = num1 + num2;
    num2 = num1 - num2;
    num1 = num1 - num2;

    println!("After swapping: num1 = {}, num2 = {}", num1, num2);
}
```

10. Compound Data Types (Arrays and Tuples):

Arrays:

```
fn main() {
   let numbers: [i32; 5] = [1, 2, 3, 4, 5];
   println!("Third element: {}", numbers[2]);
}
```

Tuples:

```
fn main() {
   let person: (String, i32, bool) = ("Alice".to_string(), 30, true);
   println!("Name: {}", person.0);
   println!("Age: {}", person.1);
   println!("Active: {}", person.2);
}
```

11. Loops and Conditional Loops:

```
fn main() {
   let mut count = 0;
   while count < 5 {
      println!("Count: {}", count);
      count += 1;
   }

   for i in 0..5 {
      println!("Iteration: {}", i);
   }
}
```

12. Assigning Value, Passing to Function, Returning from Function:

A) Assigning value of one variable to another:

```
fn main() {
   let x = 5;
   let y = x;
   println!("x: {}, y: {}", x, y);
}
```

B) Passing value to a function:

```
fn greet(name: &str) {
    println!("Hello, {}!", name);
}

fn main() {
    let user_name = "Alice";
    greet(user_name);
}
```

C) Returning value from a function:

```
fn square(x: i32) -> i32 {
    x * x
}

fn main() {
    let num = 5;
    let result = square(num);
    println!("Square of {} is {}", num, result);
}
```

13. Generating a Random Number:

```
use rand::Rng;

fn main() {
    let secret_number = rand::thread_rng().gen_range(1..101);
    println!("Secret number: {}", secret_number);
}
```

14. Comparing Guessed Number with Secret Number:

```
use std::io;
use rand::Rng;

fn main() {
    let secret_number = rand::thread_rng().gen_range(1..101);

    println!("Guess the secret number!");

    loop {
        let mut guess = String::new();
```

```
        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");

        let guess: i32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };

        if guess == secret_number {
            println!("Congratulations, you guessed right!");
            break;
        } else if guess < secret_number {
            println!("Too low! Try again.");
        } else {
            println!("Too high! Try again.");
        }
    }
}
```

15. Borrowing in Rust:

```
fn main() {
    let x = 5;
    let y = &x; // Borrowing x

    println!("x: {}, y: {}", x, y);
}
```