

20CYS312 -PRINCIPLE OF PROGRAMMING LANGUAGES

Date: 21-02-2025

Name: Naren S

Roll no.: CH.EN.U4CYS22034

Github: https://github.com/narens/22034_20CYS312

LAB-7 - Programming with RUST

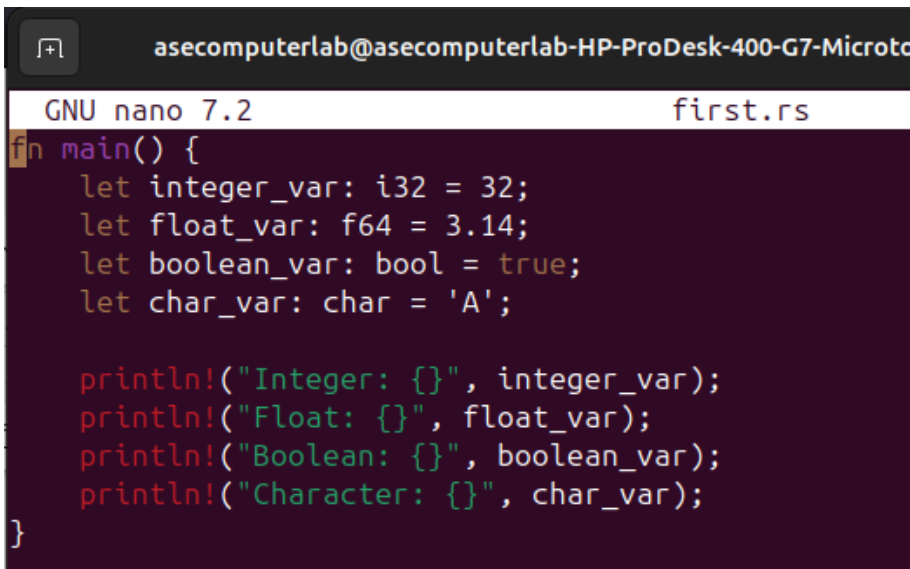
Task 1: Data Types and Variables

1. Declare variables of the following types: integer, floating-point, boolean, and character. Print the value of each variable.

Objective:

- Understand how to declare variables of different data types (integer, floating-point, boolean, and character) and print their values.

Code:

A screenshot of a terminal window with a dark background. The title bar at the top shows a terminal icon and the text 'asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microt'. The terminal content shows the GNU nano 7.2 editor editing a file named 'first.rs'. The code is written in Rust and declares four variables: an integer 'integer_var' with value 32, a float 'float_var' with value 3.14, a boolean 'boolean_var' with value true, and a character 'char_var' with value 'A'. Each variable is then printed to the console using println! with a formatted string showing the variable name and its value in curly braces.

```
GNU nano 7.2 first.rs
fn main() {
    let integer_var: i32 = 32;
    let float_var: f64 = 3.14;
    let boolean_var: bool = true;
    let char_var: char = 'A';

    println!("Integer: {}", integer_var);
    println!("Float: {}", float_var);
    println!("Boolean: {}", boolean_var);
    println!("Character: {}", char_var);
}
```

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ rustc first.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./first
Integer: 32
Float: 3.14
Boolean: true
Character: A
```

Explanation: In this task, we declare variables of four basic data types in Rust:

- **integer (i32):** Stores whole numbers.
- **floating-point (f64):** Stores decimal numbers.
- **boolean (bool):** Stores true or false values.
- **character (char):** Stores a single Unicode character.

Task 2: Simple Arithmetic Operations

2. Declare two integer variables and perform the following operations:
 - a. Addition
 - b. Subtraction
 - c. Multiplication
 - d. Division
 - e. Modulo
3. Print the result of each operation.

Objective:

- Perform basic arithmetic operations (addition, subtraction, multiplication, division, modulo) on two integers and display the results.

Code:

```
GNU nano 7.2 operator.rs
fn main() {
    let a: i32 = 23;
    let b: i32 = 12;

    println!("Addition: {}", a + b);
    println!("Subtraction: {}", a - b);
    println!("Multiplication: {}", a * b);
    println!("Division: {}", a / b);
    println!("Modulo: {}", a % b);
}
```

Output:

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./operator
Addition: 35
Subtraction: 11
Multiplication: 276
Division: 1
Modulo: 11
```

Explanation: This task demonstrates performing common arithmetic operations in Rust:

- **Addition (+):** Adds two numbers.
- **Subtraction (-):** Subtracts one number from another.
- **Multiplication (*):** Multiplies two numbers.
- **Division (/):** Divides one number by another (integer division).
- **Modulo (%):** Finds the remainder when one number is divided by another.

Task 3: If-Else Decision Making

1. Write a program that:
 - a. Takes a number as input.
 - b. Checks whether the number is positive, negative, or zero using an if-else statement.
 - c. Print a message based on the result.

Objective:

- Take user input, check whether the number is positive, negative, or zero, and print a corresponding message based on the result.

Code and Output Examples:

```

GNU nano 7.2                                if.rs
use std::io;
fn main() {
    let mut input = String::new();
    println!("Enter a number: ");
    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 {
        println!("The number is positive.");
    } else if number < 0 {
        println!("The number is negative.");
    } else {
        println!("The number is zero.");
    }
}

```

```

asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ rustc if.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./if
Enter a number:
-2
The number is negative.
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./if
Enter a number:
0
The number is zero.
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./if
Enter a number:
34
The number is positive.

```

Explanation: In this task, we demonstrate how to use an **if-else statement** to make decisions based on the input:

- If the number is greater than zero, we print "positive".
- If the number is less than zero, we print "negative".
- If the number is exactly zero, we print "zero". This decision-making structure helps us handle different scenarios based on input values.

Task 4: Checking for Even or Odd

1. Write a program that:
 - a. Takes an integer as input.
 - b. Uses an if-else statement to check if the number is even or odd.

- c. Print "Even" if the number is even and "Odd" if the number is odd.

Objective:

- Take an integer input and determine if it is even or odd using an if-else statement.

Code and Output Examples:

```
GNU nano 7.2                                odd.rs
use std::io;

fn main() {
    let mut input = String::new();
    println!("Enter an integer: ");
    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 % 2 == 0 {
        println!("Even");
    } else {
        println!("Odd");
    }
}
```

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ nano odd.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ rustc odd.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./odd
Enter an integer:
34
Even
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./odd
Enter an integer:
99
Odd
```

Explanation: This task checks if a number is **even** or **odd**. The rule for checking:

- A number is even if it divides by 2 without leaving a remainder ($\text{number} \% 2 == 0$).
- Otherwise, it is odd.

Task 5: Using a Loop to Print Numbers

2. Write a program that uses a for loop to print the even numbers from the range 1 to 20.

Objective:

- Use a **while loop** to print odd numbers between 1 and 20.

Code and Output Examples:

```
GNU nano 7.2                                loop.rs
fn main() {
    for num in 1..=20 {
        if num % 2 == 0 {
            println!("{}", num);
        }
    }
}
```

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ nano loop.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ rustc loop.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./loop
2
4
6
8
10
12
14
16
18
20
```

- **Explanation:** In this task, we demonstrate how to use a for loop in Rust. The loop iterates over a range of numbers and prints the even numbers. To achieve this:
- We use the **step_by(2)** method to increment by 2, starting from 2 and ending at 20.

Task 6: While Loop Example

3. Write a program that uses a while loop to print odd numbers from the range 1 to 20.

Objective:

To use a while loop to print odd numbers from the range (1 to 20).

Code and Output Examples:

```

GNU nano 7.2                               whileloop.rs
fn main() {
    let mut num = 1;
    while num <= 20 {
        if num % 2 != 0 {
            println!("{}", num);
        }
        num += 1;
    }
}

```

```

asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ rustc whileloop.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./whileloop
1
3
5
7
9
11
13
15
17
19

```

- **explanation:**

Here, we use a while loop to iterate through numbers from 1 to 20 and check if the number is odd. Odd numbers have a remainder of 1 when divided by 2 ($\text{num} \% 2 \neq 0$).

Task 7: Using a For Loop with a Range

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

Objective:

To print numbers from 10 to 1 in reverse order using a for loop in Rust.

Code and Output Examples:

```
GNU nano 7.2 rev.rs
fn main() {
    for num in (1..=10).rev() {
        println!("{}", num);
    }
}
```

```
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ nano rev.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ rustc rev.rs
asecomputerlab@asecomputerlab-HP-ProDesk-400-G7-Microtower-PC:~$ ./rev
10
9
8
7
6
5
4
3
2
1
```

Explanation:

We can use the for loop with the range syntax, where the step is negative. In Rust, we can specify a step by using `rev()` on a range.

Conclusion:

Got insights into the rust language and its data types, loops – for and while , basic inbuilt functions like `rev()` and on getting user input for the functions defined. Worked with odd and even numbers as well.