# 20CYS312 - Principles of Programing Languages - Lab Exercise 7

## Roll number: CH.EN.U4CYS22002

## Name: S. ASWIN SREE RAM

6th-sem-labs/PPL at main · lvexol/6th-sem-labs

Contribute to lvexol/6th-sem-labs development by creating an account on GitHub.

https://github.com/lvexol/6th-sem-labs/tree/main/PPL

lvexol/**Labs**

1 Contributor 0 Issues 0 Stars 0 Forks

# Ex01. Declare variables of the following types: integer, floating-point, boolean, and character. Print the value of each variable.

### Program Code:

```
fn main() {
    let integer: i32 = 42;          // Integer
    let float: f64 = 3.14;          // Floating-point
    let boolean: bool = true;       // Boolean
    let character: char = 'R';      // Character

    println!("Integer: {}", integer);
    println!("Float: {}", float);
    println!("Boolean: {}", boolean);
```

```
    println!("Character: {}", character);
}
```

---

## Explanation of the Code:

- `integer: i32 = 42;` → Declares an integer variable of type `i32`.
- `float: f64 = 3.14;` → Declares a floating-point variable of type `f64`.
- `boolean: bool = true;` → Declares a boolean variable with value `true`.
- `character: char = 'R';` → Declares a character variable using single quotes.
- The `println!` macro is used to print the values of these variables.

---

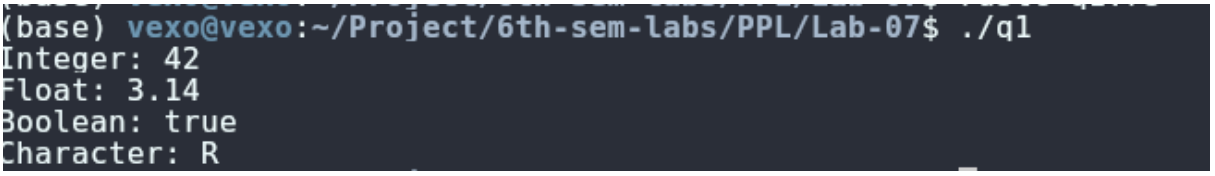
## Input/Output Example:

**Expected Output:**

```
Integer: 42
Float: 3.14
Boolean: true
Character: R
```

---

## Screenshots:




---

## Conclusion:

This program demonstrates how to declare and initialize **integer, floating-point, boolean, and character** variables in Rust. The `println!` macro is used to display their values, introducing basic variable declaration and printing in Rust.

# Ex02. Simple Arithmetic Operations

## Program Code:

```
fn main() {
    let a: i32 = 15;
    let b: i32 = 4;

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

---

## Explanation of the Code:

- `let a: i32 = 15;` → Declares an integer variable `a` with value `15`.
- `let b: i32 = 4;` → Declares an integer variable `b` with value `4`.
- The program performs the following arithmetic operations:
  - **Addition (`+`)** → `a + b` gives `19`.
  - **Subtraction (`-`)** → `a - b` gives `11`.
  - **Multiplication (`*`)** → `a * b` gives `60`.
  - **Division (`/`)** → `a / b` gives `3` (integer division).
  - **Modulo (`%`)** → `a % b` gives `3` (remainder of division).

---

## Input/Output Example:

### Expected Output:

```
Addition: 19
Subtraction: 11
Multiplication: 60
Division: 3
Modulo: 3
```

---

## Screenshots:

```
(base) vexo@vexo:~/Project/6th-sem-labs/PPL/Lab-07$ nvim q2.rs
(base) vexo@vexo:~/Project/6th-sem-labs/PPL/Lab-07$ rustc q2.rs
(base) vexo@vexo:~/Project/6th-sem-labs/PPL/Lab-07$ ./q2
Addition: 19
Subtraction: 11
Multiplication: 60
Division: 3
Modulo: 3
```

---

### Conclusion:

This program demonstrates **basic arithmetic operations** in Rust using two integer variables. It performs **addition, subtraction, multiplication, division, and modulo** operations, printing the results using the `println!` macro.

# Ex03. If-Else Decision Making

---

### Program Code:

```
use std::io;

fn main() {
    let mut input = String::new();
    println!("Enter a number:");

    io::stdin().read_line(&mut input).expect("Failed to read input");
    let number: i32 = input.trim().parse().expect("Please enter a valid intege
r");

    if number > 0 {
        println!("The number is Positive.");
    } else if number < 0 {
        println!("The number is Negative.");
    } else {
        println!("The number is Zero.");
    }
}
```

---

### Explanation of the Code:

- `use std::io;` → Imports the input/output module for user input.
- `let mut input = String::new();` → Declares a mutable string to store user input.
- `io::stdin().read_line(&mut input)` → Reads input from the user.
- `let number: i32 = input.trim().parse()` → Converts the input into an integer.
- **If-Else Conditions:**
  - If `number > 0`, print `"The number is Positive."`
  - If `number < 0`, print `"The number is Negative."`
  - Otherwise, print `"The number is Zero."`

---

## Input/Output Example:

### Input:

```
Enter a number: 7
```

### Output:

```
The number is Positive.
```

### Input:

```
Enter a number: -5
```

### Output:

```
The number is Negative.
```

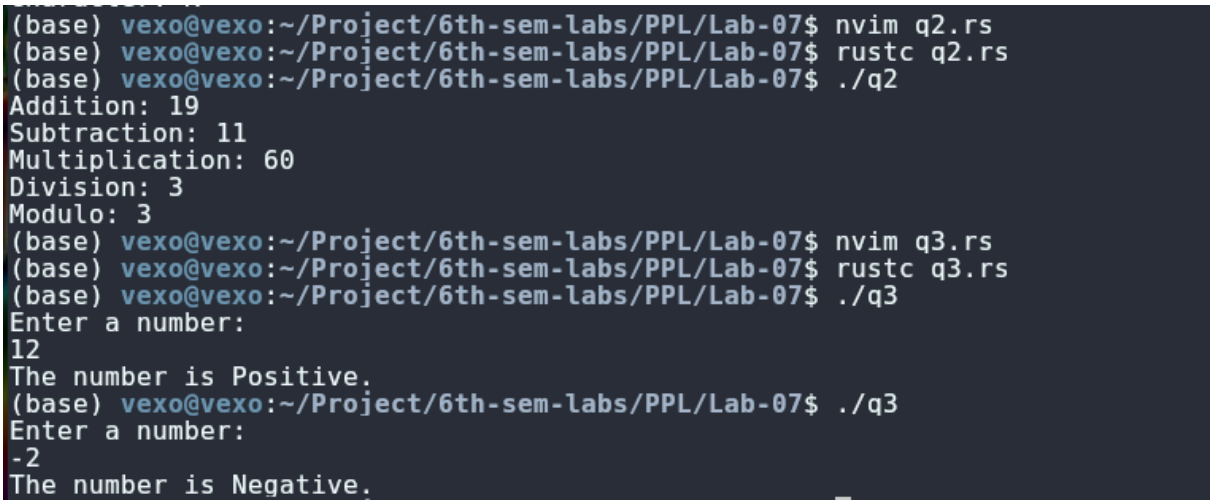
### Input:

```
Enter a number: 0
```

### Output:

```
The number is Zero.
```

### Screenshots:




### Conclusion:

This program demonstrates **if-else decision-making** in Rust. It takes a user-inputted number, checks whether it's **positive, negative, or zero**, and prints the appropriate message accordingly.

# Ex04. Checking for Even or Odd

### Program Code:

```
use std::io;

fn main() {
    let mut input = String::new();
    println!("Enter an integer:");

    io::stdin().read_line(&mut input).expect("Failed to read input");
    let number: i32 = input.trim().parse().expect("Please enter a valid intege
r");
```

```
    if number % 2 == 0 {
        println!("Even");
    } else {
        println!("Odd");
    }
}
```

---

## Explanation of the Code:

- `use std::io;` → Imports the input/output module for user input.
- `let mut input = String::new();` → Declares a mutable string to store user input.
- `io::stdin().read_line(&mut input)` → Reads input from the user.
- `let number: i32 = input.trim().parse()` → Converts the input into an integer.
- **If-Else Condition:**
  - If `number % 2 == 0`, print `"Even"`.
  - Otherwise, print `"Odd"`.

---

## Input/Output Example:

## Input:

```
Enter an integer: 8
```

## Output:

```
Even
```

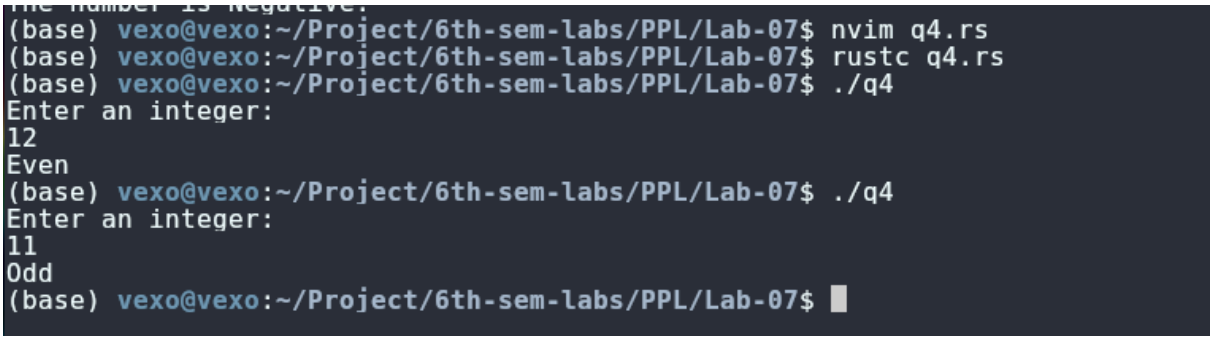
## Input:

```
Enter an integer: 13
```

## Output:

```
Odd
```

### Screenshots:




### Conclusion:

This program demonstrates **if-else decision-making** in Rust. It takes a user-inputted integer, checks whether it's **even or odd**, and prints the corresponding result.

# Ex05. Using a Loop to Print Even Numbers

### Program Code:

```
fn main() {
    for number in 1.. {
        if number % 2 == 0 {
            println!("{}", number);
        }
    }
}
```

### Explanation of the Code:

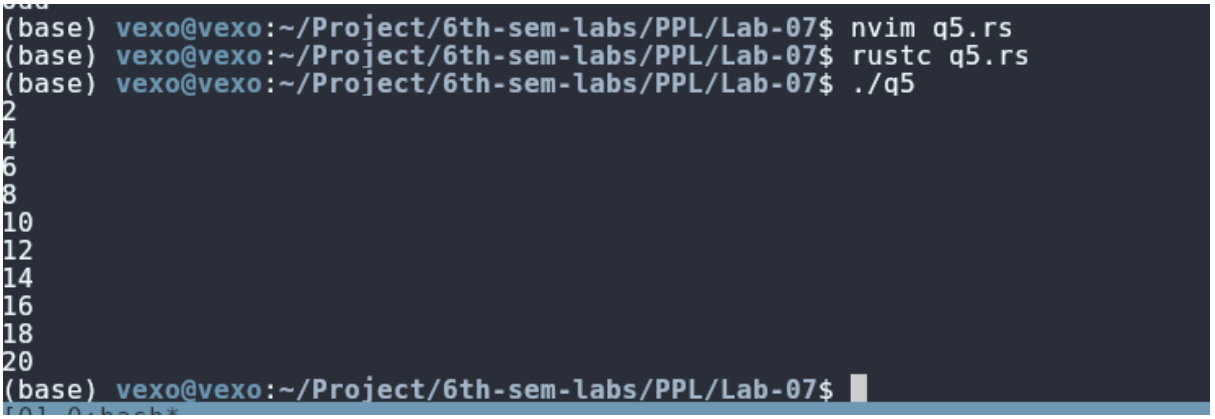
- The `for` loop iterates through numbers from `1` to `20` using the range `1..=20`.
- The `if number % 2 == 0` condition checks if the number is even.
- If true, the number is printed using `println!()`.

### Input/Output Example:

### Output:

```
2
4
6
8
10
12
14
16
18
20
```

### Screenshots:




### Conclusion:

This program demonstrates **looping in Rust** using a `for` loop. It efficiently prints **even numbers** in the range `1 to 20` by checking divisibility with `2`.

# Ex06. While Loop Example

### Program Code:

```
fn main() {
    let mut number = 1;

    while number <= 20 {
        if number % 2 != 0 {
            println!("{}", number);
        }
        number += 1;
    }
}
```

---

## Explanation of the Code:

- `let mut number = 1;` → Initializes the variable `number` to `1`.
- `while number <= 20` → Runs the loop while `number` is less than or equal to `20`.
- `if number % 2 != 0` → Checks if the number is odd.
- If true, `println!("{}", number);` prints the number.
- `number += 1;` increments `number` by `1` to move to the next value.

---

## Input/Output Example:

## Output:

```
1
3
5
7
9
11
13
15
17
19
```

---

## Screenshots:

```
(base) vexo@vexo:~/Project/6th-sem-labs/PPL/Lab-07$ nvim q6.rs
(base) vexo@vexo:~/Project/6th-sem-labs/PPL/Lab-07$ rustc q6.rs
(base) vexo@vexo:~/Project/6th-sem-labs/PPL/Lab-07$ ./q6
1
3
5
7
9
11
13
15
17
19
```

---

### Conclusion:

This program demonstrates the **while loop** in Rust. It prints **odd numbers** in the range `1 to 20` by checking divisibility with `2` and incrementing the counter in each iteration.

# Ex07. Using a For Loop with a Range (Reverse Order)

---

### Program Code:

```
fn main() {
    for number in (1..=10).rev() {
        println!("{}", number);
    }
}
```

---

### Explanation of the Code:

- The `for` loop iterates over the range `(1..=10).rev()`, which generates numbers from `10` to `1` in reverse order.
- The `rev()` function reverses the range, making it count **backward**.
- Each number is printed using `println!()`.

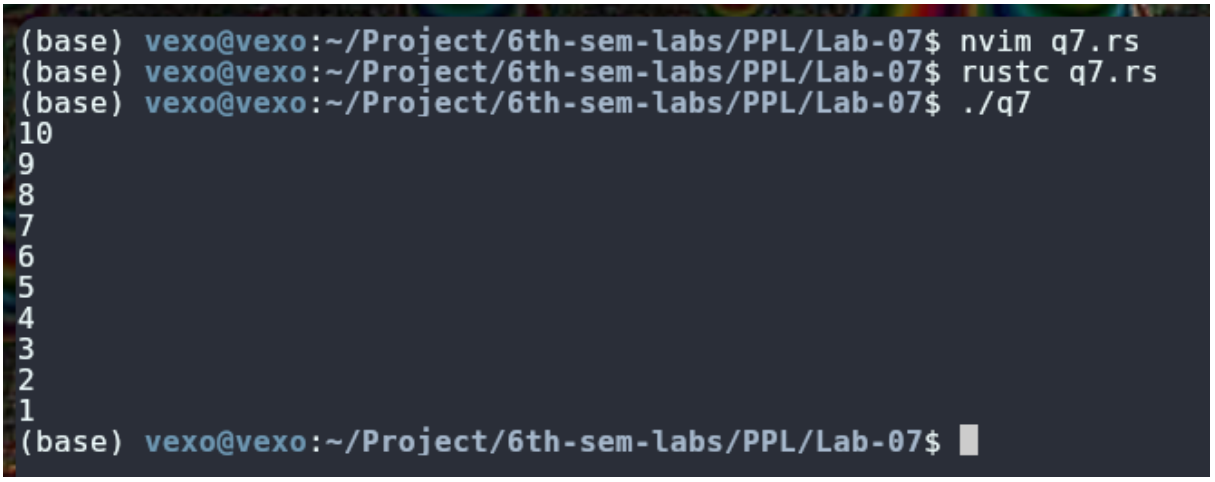
---

### Input/Output Example:

### Output:

```
10
9
8
7
6
5
4
3
2
1
```

## Screenshots:




## Conclusion:

This program demonstrates **using a for loop with a reversed range** in Rust. It efficiently prints numbers from **10 to 1 in reverse order** using the `.rev()` function.