### **Lab 4:** **Exception Handling**

**Exercise: Safe Division with Error Handling**

In this exercise, we will create a Rust program to perform division between two numbers and handle possible errors, such as division by zero, using the Result type.

1. Create a new Rust project using cargo:
2. Open your terminal/command prompt and run the following command:

cargo new safe\_division

cd safe\_division

1. Open the main.rs file in the src directory of your project. You can use any code editor for this.
2. Write a function called safe\_divide that performs division between two numbers and returns a Result:

fn safe\_divide(dividend: i32, divisor: i32) -> Result<i32, String> {

if divisor == 0 {

Err("Division by zero is not allowed.".to\_string())

} else {

Ok(dividend / divisor)

}

}

In the main function, call the safe\_divide function and handle possible errors:

fn main() {

let dividend = 10;

let divisor = 2;

match safe\_divide(dividend, divisor) {

Ok(result) => println!("Result: {}", result),

Err(error) => println!("Error: {}", error),

}

let dividend = 20;

let divisor = 0;

match safe\_divide(dividend, divisor) {

Ok(result) => println!("Result: {}", result),

Err(error) => println!("Error: {}", error),

}

}

1. Save the file and return to your terminal/command prompt.
2. Build and run your program using cargo run:

cargo run

The program will perform the division and display the result if successful or an error message if division by zero occurs.

Example Output:

Result: 5

Error: Division by zero is not allowed.

We have now successfully completed the lab exercise on exception handling in Rust, using Result to handle errors during division! Error handling in Rust emphasizes the use of Result to indicate possible failure and encourages explicit handling of errors in a way that ensures safer and more reliable code.

**Happy Coding!**