**Principles of Programming Language**

**LAB 10 -RUST**

**Name: Kiruthik Pranav P V Roll no:CH.EN.U4CYS22026**

**Github Link:** **https://github.com/kpsan12/Haskell-Programming/tree/main/lab10**

--------------------------------------------------------------------------------------------------------------------------------------

**Question:**

1.Write a Rust program that does the following:

1. Reads the contents of a file named "input.txt".

2. Handles possible errors (file not found, permission denied, etc.) using Result<T, E>.

3. Writes the content to a new file named "output.txt".

4. Uses Option<T> to check if the file is empty and prints an appropriate message.

**OBJECTIVE:**

* Understand and implement structured error handling using Result<T, E> in Rust.
* Learn how to read from and write to files using Rust's standard library (std::fs).
* Handle potential errors such as file not found, permission denied, etc.
* Use Option<T> to check if the file is empty and display an appropriate message.

**Code:**

use std::fs::File;

use std::io::{self, Read, Write};

use std::path::Path;

fn main() {

    let input\_file = "input.txt";

    let output\_file = "output.txt";

    match read\_file(input\_file) {

        Ok(content) => {

            if content.is\_empty() {

                println!("The file '{}' is empty.", input\_file);

            } else {

                println!("File read successfully. Writing to '{}'...", output\_file);

                if let Err(e) = write\_file(output\_file, &content) {

                    eprintln!("Error writing to file: {}", e);

                } else {

                    println!("File written successfully.");

                }

            }

        }

        Err(e) => eprintln!("Error reading file: {}", e),

    }

}

fn read\_file(filename: &str) -> Result<String, io::Error> {

    let path = Path::new(filename);

    if !path.exists() {

        return Err(io::Error::new(io::ErrorKind::NotFound, "File not found"));

    }

    let mut file = File::open(filename)?;

    let mut contents = String::new();

    file.read\_to\_string(&mut contents)?;

    Ok(contents)

}

fn write\_file(filename: &str, content: &str) -> Result<(), io::Error> {

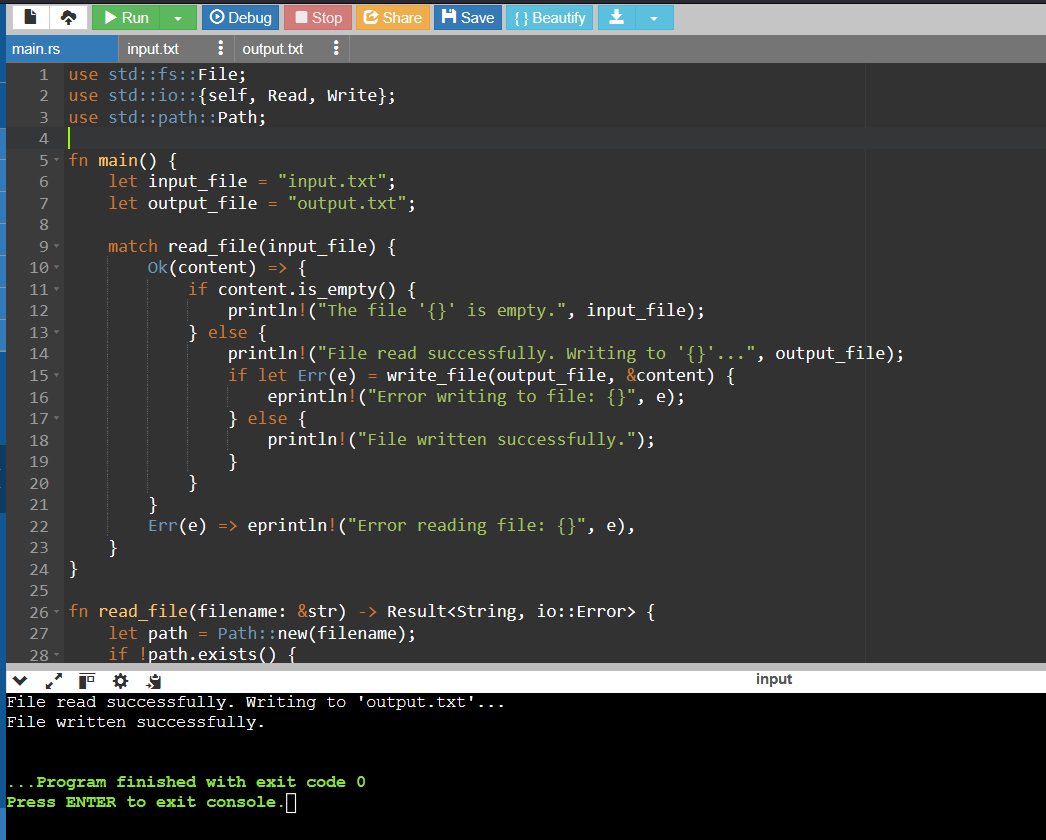
    let mut file = File::create(filename)?;

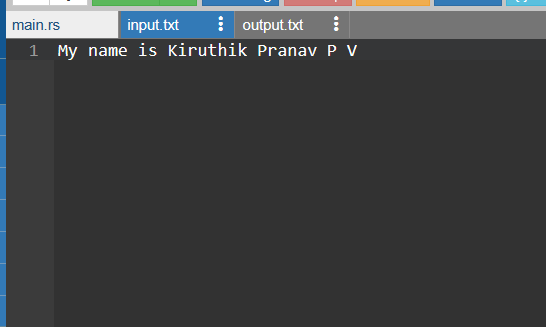
    file.write\_all(content.as\_bytes())?;

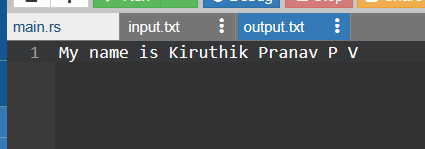
    Ok(())

}

**Output:**







**Explanation:**

* This program reads content from an input file ("input.txt") and writes it to an output file ("output.txt"), with comprehensive error handling using Rust's Result type for both operations.
* The read\_file() function first checks if the file exists, then opens it, reads its contents into a String, and returns either the content or an appropriate error.
* The write\_file() function creates (or overwrites) the output file, converts the string content to bytes, writes it to the file, and returns either success or an error, with the main function printing appropriate status messages throughout the process.

**CONCLUSION:**

This exercise improved error handling skills in Rust by implementing structured file I/O operations using Result<T, E> and Option<T>.