Group Programming Assignment – Part B

Project Title: Safe Backup Utility in Rust

Group 10-
Abdullah Arefin Sinha -20250045
S M Ashiquzzaman Saikat-20240188
Md Asif IQBAL-20240077

GitHub Repository: https://github.com/saikat3424345/safe_backup_rust

Project Overview

The **Safe Backup Utility in Rust** is a command-line tool designed to provide reliable and secure file backups. Built using the Rust programming language, this project emphasizes performance, safety, and simplicity. In today's digital world, data loss can occur due to hardware failure, accidental deletion, or software bugs. This utility aims to mitigate such risks by offering a straightforward way to back up files and directories with minimal user effort.

Rust was chosen for its memory safety guarantees, concurrency support, and system-level control. The project also serves as a learning experience in using Git for version control and GitHub for collaboration and code hosting.

Objectives

The primary goals of this project are:

- Develop a fast and secure backup tool using Rust
- Explore Rust's file system and error-handling capabilities
- Practice software engineering principles such as modularity and documentation
- Learn Git and GitHub workflows for managing code and reports
- Create a professional PDF report to summarize the project

Technologies Used

Tool	Purpose
Rust	Core programming language
Cargo	Rust's package manager and build tool
Git	Version control
GitHub	Remote repository hosting
Word/Docs	Report writing and PDF generation

Rust's strong type system and compile-time checks help prevent common bugs, while GitHub provides a platform for sharing and versioning the codebase.

Project Structure

```
safe_backup_rust/
├── src/
│   └── main.rs
├── Cargo.toml
├── docs/
│   └── report.pdf
└── README.md
```

- **main.rs**: Contains the core logic for file backup
- **Cargo.toml**: Defines dependencies and project metadata
- **docs/report.pdf**: This report
- **README.md**: Project overview and usage instructions

How It Works

The backup utility operates through the terminal. The user provides a source path (file or folder) and a destination path. The program performs the following steps:

1. **Validation**: Checks if the source exists and is accessible
2. **Safety Check**: Verifies if the destination already contains a backup to avoid overwriting
3. **Copy Operation**: Uses Rust's standard library to copy files
4. **Feedback**: Prints success or error messages to the terminal

This approach ensures that backups are performed safely and transparently.

Example usage:

```
cargo run -- /path/to/source /path/to/backup
```

Code Snippet

Here's a simplified version of the backup logic:

```
use std::fs;
```

```
use std::path::Path;

fn backup_file(src: &str, dest: &str) {
    if Path::new(dest).exists() {
        println!("Backup already exists.");
    } else {
        match fs::copy(src, dest) {
            Ok(_) => println!("Backup successful."),
            Err(e) => println!("Error: {}", e),
        }
    }
}
```

This function checks if the destination file exists and performs the copy operation with error handling.

Results and Achievements

- Successfully backed up files without data loss
- Handled edge cases such as missing paths and duplicate backups
- Gained hands-on experience with Rust's file system APIs
- Practiced Git commands like `add`, `commit`, `push`, and `branch`
- Created a structured GitHub repository with documentation and report

The project demonstrates how Rust can be used for practical system utilities and how version control enhances software development.

Challenges Faced

- Understanding Rust's ownership and borrowing model
- Handling file system errors gracefully
- Configuring Git remotes and resolving push errors
- Formatting the report for clarity and professionalism

Each challenge was an opportunity to learn and improve coding and documentation skills.

Future Improvements

To make the utility more robust and user-friendly, the following enhancements are planned:

- Support for recursive folder backups
- Implement compression using libraries like `flate2` or `zip`

- Build a graphical interface using `egui` or `iced`
- Add detailed logging to a file for audit and debugging
- Enable cloud backup integration (e.g., Dropbox, Google Drive)
-

These features would expand the tool's usability and appeal to a wider audience.

Conclusion

The **Safe Backup Utility in Rust** is a practical and educational project that combines system programming with software engineering best practices. It showcases how Rust can be used to build reliable tools and how GitHub can serve as a platform for sharing and collaboration. The experience gained through this project lays the foundation for more advanced Rust development and open-source contributions.