Rust Lab: Extended Spreadsheet program

Team Members

Shreya Bhaskar(2023CS10941)
Sweety Naveen Kumar (2023CS10172)
Sambhav Singhwi (2023CS10722)

Contents

1 Introduction

This report describes a spreadsheet program written in Rust. The program allows users to create, edit, and manage spreadsheets through a command-line interface or a web interface when extensions are enabled. It supports basic spreadsheet operations like entering formulas, copying and pasting cells, and visualizing data with graphs. The program is designed to be efficient and robust, handling dependencies and errors effectively.

2 Program Overview

The spreadsheet program is built in Rust, a programming language known for its safety and performance. The program is modular, with separate files for different functionalities:

- `main.rs`: Handles program startup, user input, and the web server (if enabled).
- `sheet.rs`: Manages the spreadsheet's structure, display, and commands like scrolling, copying, and pasting.
- `cell.rs`: Handles cell updates, formula evaluation, and dependency management.
- `dependencies.rs`: Manages cell dependencies and detects circular dependencies.
- `types.rs`: Defines data structures like cells, sheets, and dependency types.
- `utils.rs`: Contains helper functions for parsing, pattern detection, and calculations.

The program supports two modes:

- **Standard Mode**: A command-line interface for basic spreadsheet operations.
- **Extension Mode**: Adds advanced features like undo/redo, row/column deletion etc , and a web interface using the Rocket framework which run with the "–extension" flag (`./target/release/spreadsheet --extension 999 18278`)

3 Key Features

The program provides the following features:

3.1 Core Features

- **CLI Compatibility**: The application provides complete compatibility with the C version, ensuring autograder compatibility.
- **Spreadsheet Operations**: Supports standard spreadsheet functionality including cell references, arithmetic operations, and various formulae.
- **Navigation**: Cell navigation using `w/a/s/d` keys.
- **Viewport Control**: Users can enable or disable output features as needed.

- **Cell Management**: Users can enter values or formulas in cells. Formulas support arithmetic operations (e.g., `A1+B1`) and functions like `SUM`, `AVG`, `MIN`, `MAX`, and `STDEV`.
- **Dependency Tracking**: The program tracks cell dependencies and updates dependent cells when a cell's value changes. It also detects circular dependencies to prevent errors.
- **Scrolling and Navigation**: Users can scroll through the spreadsheet using `w`, `a`, `s`, `d` keys or jump to a specific cell with the `scroll_to` command.

3.2 Extended Features

- **Web Interface**: Extension mode includes a web interface for interacting with the spreadsheet using the Rocket framework.
- **Copy, Cut, and Paste**: Users can copy or cut a range of cells and paste them elsewhere, preserving formulas and values using commands like (e.g. `CUT A1:A1` for single cell, `CUT A1:A100` for range of cells , `COPY A1:A1`, `COPY A1:A100` and using the starting cell reference only to paste as in `PASTE A1`).
- **Formatting**: In extension mode, users can apply bold, italic, or underline formatting to cells using commands like `A1=BOLD(A1)`.
- **Formula Bar**: In extension mode, users can use the `FORMULA` command (e.g., `FORMULA A1`) to view the formula associated with a specific cell, if any.
- **Autofill**: The program detects patterns (e.g., Arithmetic, Geometric, Fibonacci,Triangular and Factorial sequences) and autofills cells based on these patterns using commands with the starting cell from which the pattern has to be filled as reference like `A5=AUTOFILL(A5:A999)`.
- **Graphs and Scatter Plots**: Users can create bar graphs or scatter plots for a range of cells in extension mode by using commands like (`GRAPH (BAR) A1:A10`) and (`GRAPH (SCATTER) A1:A10`)
- **Undo/Redo**: Extension mode supports undoing and redoing actions, storing up to 10 previous states.
- **File Import**: In extension mode, users can import data from CSV or Excel files. Loading is done while executing the program (e.g., `./target/release/spreadsheet --extension 999 18278 input.csv`)
- **Sorting**: In extension mode, supports one-dimensional and two-dimensional sorting in ascending (`SORTA`) and descending (`SORTD`) order for cell ranges, implemented in `sheet.rs`'s `process_command` function (e.g., `SORTA A1:A10`, `SORTD B1:C5`).
- **Row and Column Deletion**: In extension mode, users can delete rows or columns using commands like `ROWDEL 1` or `COLDEL A`, updating the spreadsheet structure and dependencies.

4 Implementation Details

The program uses a modular design with clear separation of concerns. Below are key implementation details:

4.1 Data Structures

The `types.rs` file defines the core data structures:

- `Cell`: Stores a cell's value, formula, and properties like error status and formatting (bold, italic, underline).
- `Sheet`: Represents the spreadsheet, containing a grid of cells, dependency graph, and state information like view position and command history.
- `DependencyType`: Represents cell dependencies, either single cells or ranges.
- `CellDependencies`: Tracks which cells a cell depends on and which cells depend on it.

4.2 Formula Evaluation

The `cell.rs` file handles formula evaluation. Formulas can include:

- Arithmetic operations (e.g., `A1+B1*2`).
- Range functions (e.g., `SUM(A1:A5)`).
- Special functions like `SLEEP` for pausing execution.

The program parses formulas, resolves cell references, and evaluates expressions. It also checks for errors like division by zero or invalid references.

4.3 Dependency Management

The `dependencies.rs` file manages cell dependencies using a graph stored in a `HashMap`. Key functions include:

- `has_circular_dependency`: Uses depth-first search to detect circular dependencies.
- `recalculate_dependents`: Uses breadth-first search and topological sorting to update dependent cells in the correct order.
- `remove_dependency`: Removes dependencies when cells are modified or deleted.

4.4 Command Processing

The `sheet.rs` file processes user commands. Commands include:

- Navigation: `w`, `a`, `s`, `d`, `scroll_to A1`.
- Cell updates: `A1=5`, `A1=SUM(B1:B5)`.

- Clipboard operations: `COPY A1:A5`, `CUT A1:A5`, `PASTE B1`.
- Formatting: `A1=BOLD(B1)`.
- Graphs: `GRAPH (BAR) A1:A5`.
- Row/Column deletion: `ROWDEL 1`, `COLDEL A`.

Commands are validated using `is_valid_command` in `utils.rs` before execution.

4.5 Pattern Detection and Autofill

The `utils.rs` file includes `detect_pattern` to identify sequences like constant, arithmetic, geometric, Fibonacci, factorial, or triangular. The `AUTOFILL` command uses this to fill cells based on detected patterns.

4.6 Web Interface

In extension mode, `main.rs` sets up a Rocket web server with routes like `index`, `command`, and `scroll` to manage user interactions. The web interface is defined in the `index.html.tera` template, located in the templates folder, which renders the spreadsheet as an interactive HTML table. Key features include:

- A responsive table displaying cell values with dynamic styling (e.g., bold, italic, underline) based on cell properties, using CSS variables for light and dark themes.
- A theme toggle switch implemented in JavaScript, allowing users to switch between light and dark modes with preferences saved in local storage.
- A status bar showing `(ok)` or `(err)` for circular dependency detection, styled with theme-aware colors.
- A message area for command outputs (e.g., `FORMULA A1` results), formatted with a monospace font and responsive layout.
- A command input form for entering commands (e.g., `A1=5`, `w`), with automatic focus on page load and mobile-friendly design.

The Tera template uses templating syntax to populate data (e.g., cell values, column headers) from the `index` route in `main.rs`. CSS ensures a modern, accessible interface with smooth theme transitions and responsive layouts for desktop and mobile devices.

5 Approaches for Encapsulation

The program uses encapsulation to manage complexity and ensure data integrity through Rust's features:

- **Struct Encapsulation**: Structs like `Cell` and `Sheet` in `types.rs` bundle data (e.g., cell value, formula) and expose it via public methods, controlling access and modifications.

- **Module Boundaries**: Modules like `cell.rs` and `dependencies.rs` encapsulate specific functionality, exposing only necessary functions (e.g., `update_cell`) to prevent direct access to internal logic.
- **Global State**: Global variables (`SHEET`, `CLIPBOARD`) in `types.rs` are wrapped in `Mutex` for thread-safe access, encapsulating shared state.
- **Ownership Model**: Rust's ownership rules ensure functions like `update_cell` in `cell.rs` use mutable references (`&mut Sheet`) to safely modify data.

These techniques keep components isolated, improving maintainability and reducing errors.

6 Interfaces Between Software Modules

The program's modules communicate through well-defined interfaces, ensuring modularity and clear data flow:

- `main.rs` and `sheet.rs`: `main.rs` calls `sheet.rs` functions like `process_command` to handle user inputs, passing commands as strings and receiving status messages or errors.
- `sheet.rs` and `cell.rs`: `sheet.rs` uses `update_cell` in `cell.rs` to modify cell values or formulas, providing a mutable `Sheet` reference and cell coordinates.
- `cell.rs` and `dependencies.rs`: `cell.rs` calls `recalculate_dependents` and other check functions such as `has_circular_dependency` in `dependencies.rs` to update dependent cells and check for circular references, passing cell IDs and dependency graphs.
- `types.rs` as Shared Interface: All modules use structs like `Cell`, `Sheet`, and `CellDependencies` from `types.rs`, ensuring consistent data representation across the program.
- `utils.rs` Support: Modules like `cell.rs` and `sheet.rs` use `utils.rs` functions (e.g., `parse_cell`, `detect_pattern`) for parsing and calculations, passing input data and receiving processed results.
- `main.rs` and `index.html.tera`: In extension mode, `main.rs` provides data (e.g., cell values, styles) to the `index.html.tera` template via the `index` route, which renders the web interface and returns user commands to `main.rs` via the `command` route.

7 Extensions and Future Work

In addition to the proposed extensions for the Rust lab , we could also implement the following extensions :-

- **Web Interface:** The web interface renders the spreadsheet as an interactive HTML table with command input and dynamic styling.

- **Support for .csv and .xlsx files:** In extension mode, the program supports loading data from .csv and .xlsx files, enabling users to import spreadsheet data seamlessly.

8 Why This Design is Effective

The program's design is effective for the following reasons:

- **Modularity**: The clear separation of concerns across modules (e.g., `cell.rs` for cell updates, `sheet.rs` for user interactions) simplifies maintenance and feature additions.
- **Encapsulation**: Encapsulated data and interfaces (e.g., `Cell` structs, public functions) prevent unintended modifications, enhancing reliability.
- **Rust's Safety**: Rust's ownership and type system ensure safe data access and thread-safe operations, reducing runtime errors in both command-line and web modes.

9 Modifications in Design During Implementation

During development, the dependency management system evolved to improve efficiency and support additional functionalities:

- Initially, a linked list was used to store cell dependencies, which was simple but slow for frequent traversals and updates.
- This was replaced with a `Vec<DependencyType>` to allow easier iteration and support for range-based dependencies (e.g., `A1:A5`), improving flexibility for functions like `SUM`.
- The final design adopted the current `CellDependencies` structure in `types.rs`, using a HashMap to map cell IDs to their dependencies and dependents. This enabled faster lookups, efficient circular dependency detection via `has_circular_dependency`, and reliable updates through `recalculate_dependents`, supporting features like autofill and row/column deletion.

These changes enhanced performance and scalability for dependency checks and other operations.

10 Testing Approach

1. **Unit Tests**: Testing individual components (formula parsing, evaluation, dependency tracking)
2. **Integration Tests**: Testing interactions between components
3. **Regression Tests**: Ensuring compatibility with the C implementation for autograder
4. **Manual Testing**: UI and usability testing with sample spreadsheets

11 Conclusion

The Rust spreadsheet program is a robust and feature-rich application for managing spreadsheets. Its modular design makes it easy to maintain and extend. The program efficiently handles cell dependencies, formula evaluation, and user commands while providing advanced features like autofill, graphs, and a web interface in extension mode. Future improvements could include support for more file formats, additional formula functions, or enhanced graph customization.

12 References

The following resources were consulted during the development of the spreadsheet program:

1. The Rust Programming Language Book. `https://doc.rust-lang.org/book/`.
2. Rocket Web Framework Documentation. `https://rocket.rs/v0.5-rc/guide/`.
3. Tera Templating Engine Documentation. `https://tera.netlify.app/docs/`.
4. csv Crate Documentation. Andrew Gallant. Available: `https://docs.rs/csv/latest/csv/`.
5. calamine Crate Documentation.`https://docs.rs/calamine/latest/calamine/`.

13 Project link

13.1 Github Repository

https://github.com/sweek10/COP290-Rust-project.git