Software Design Report

Project Team

February 24, 2025

1 Introduction

This document provides an overview of the design decisions, challenges faced, program structure, edge cases, and error handling of our software system. The document includes a diagram illustrating key files, data structures, and functions. Additionally, the report contains links to the GitHub repository and a video demonstration.

2 Design Decisions

Our software is structured as a command-line spreadsheet application that supports arithmetic operations, functions like SUM, MIN, MAX, and dependency tracking. The key design decisions include:

- Using a graph-based structure to handle cell dependencies.
- Implementing an AVL tree to maintain dependencies efficiently.
- Enabling a user-friendly interface with interactive navigation.
- Using a modular approach with separate files for display, computation, and graph logic.
- Incorporating robust error handling mechanisms to handle invalid user input and system errors.
- Ensuring efficient memory management for large datasets.

3 Challenges Faced

During the development, we encountered several challenges:

- Implementing a cycle detection mechanism in formula dependencies.
- Managing memory efficiently to handle large spreadsheets.
- Ensuring correct parsing of user inputs with a robust error-handling mechanism.
- Optimizing performance while recalculating cell values upon dependency updates.
- Designing a user-friendly display and navigation system within the constraints of a terminal-based UI.

4 Program Structure

Our software consists of the following key files:

- **1.c** - The main entry point of the program.
- **display.c, display.h** - Handles rendering of the spreadsheet.
- **Functions.c, Functions.h** - Implements arithmetic and aggregate functions.
- **Graph.c, Graph.h** - Manages the dependency graph.
- **Parser.c, Parser.h** - Parses user commands.
- **Makefile** - Provides automated build commands.

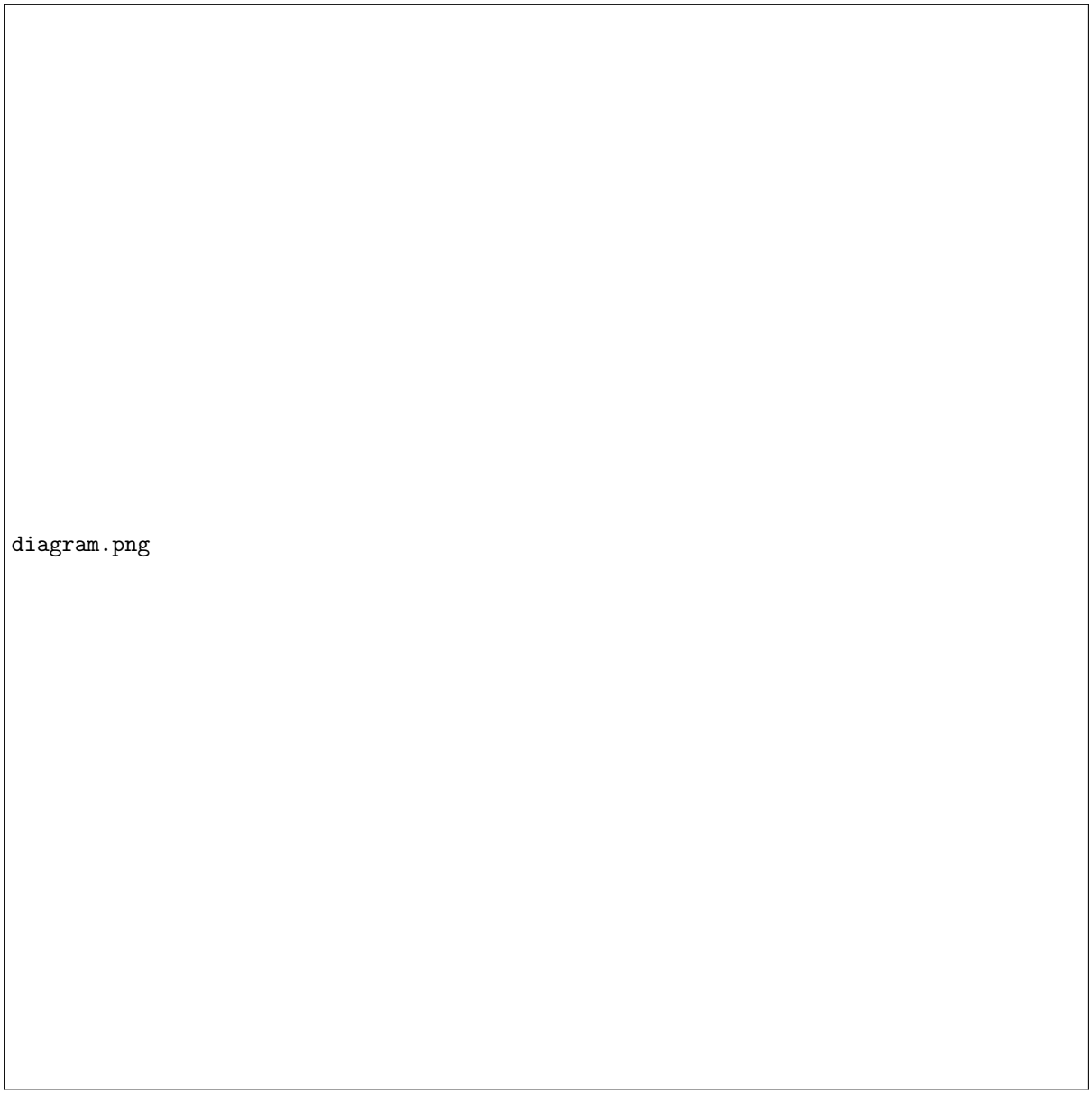

Figure 1: Software Structure

5 Edge Cases and Error Handling

Our test suite covers various edge cases, including:

- Handling division by zero.
- Detecting and preventing circular dependencies.
- Managing large spreadsheets efficiently.
- Correctly parsing invalid inputs and providing meaningful error messages.
- Handling negative numbers and large integer values.
- Ensuring correctness in dependency-based calculations with deeply nested formulas.
- Validating correct input formats and rejecting malformed expressions.

6 Performance Considerations

To ensure efficiency and responsiveness, we employed:

- AVL trees for efficient dependency tracking and updates.
- A queue-based topological sorting approach to avoid redundant calculations.
- Caching mechanisms to optimize repeated computations and minimize redundant work.
- Asynchronous computation for large formula evaluations to prevent UI lag.

7 Build and Execution

The software can be compiled and executed using the provided Makefile:

```
make        # Compiles and generates the executable
make test   # Runs all test cases
make report # Generates report.pdf from LaTeX source
```

To run the application:

```
./sheet 10 10  # Launches a 10x10 spreadsheet
```

8 Conclusion

Our software successfully implements a functional spreadsheet system with arithmetic operations, functions, and dependency management. The modular architecture ensures maintainability and scalability, while robust error handling enhances usability. Future improvements could include a GUI-based interface, additional functions, and integration with external data sources.

9 Links

Video Demo: Watch Here
GitHub Repository: View Here