Gebze Technical University

Department of Computer Engineering
CSE 241/501

Object Oriented Programming / Programming

Fall 2024
Homework 1 Documentation

Project Title: ANSI Terminal-Based Spreadsheet Program

Objective: Create a terminal-based spreadsheet application inspired by VisiCalc. The application allows users to manage data in cells, use formulas, and save/load data in CSV format.

Author: Kerim USLU

Student No: 235008003049

Contents

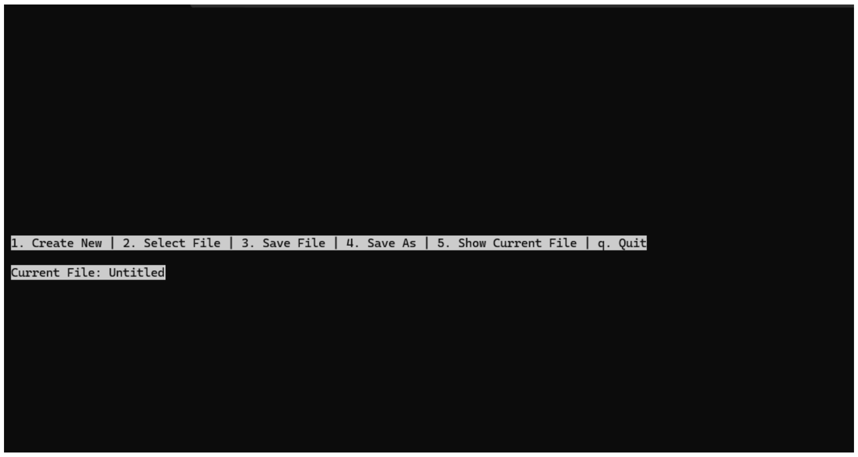
1. Introduction 3
2. Class Structure and UML Diagram 4
   2.1 Spreadsheet 4
   2.2 CellMatrix 5
   2.3 Tokenizer 6
   2.4 LexicalAnalysis 7
   2.5 Main Flow 8
3. Implementation Features 9
4. Missing Features 9
5. AI Assistance 10

Figure 1 Initial Teminal Page 3
Figure 2SpreadSheed Terminal 3
Figure 3 SpreadSheet Hierarcy UML Diagram 4
Figure 4 CellMatrix Hierarcy UML Diagram 5
Figure 5 Tokenizer Hierarchy UML Diagram 6
Figure 6 Lexical Analysis Hierarchy UML Diagram 7
Figure 7 MainFlow Uml Diagram 8

1. Introduction

The ANSI Terminal-Based Spreadsheet Program is a simplified spreadsheet application inspired by the functionality of early spreadsheet software like VisiCalc. Designed for terminal environments, this project combines object-oriented programming (OOP) principles, modern C++ features, and interactive terminal-based operations to provide users with an efficient way to manage tabular data. The program is built to be both functional and user-friendly, emphasizing core spreadsheet features such as:

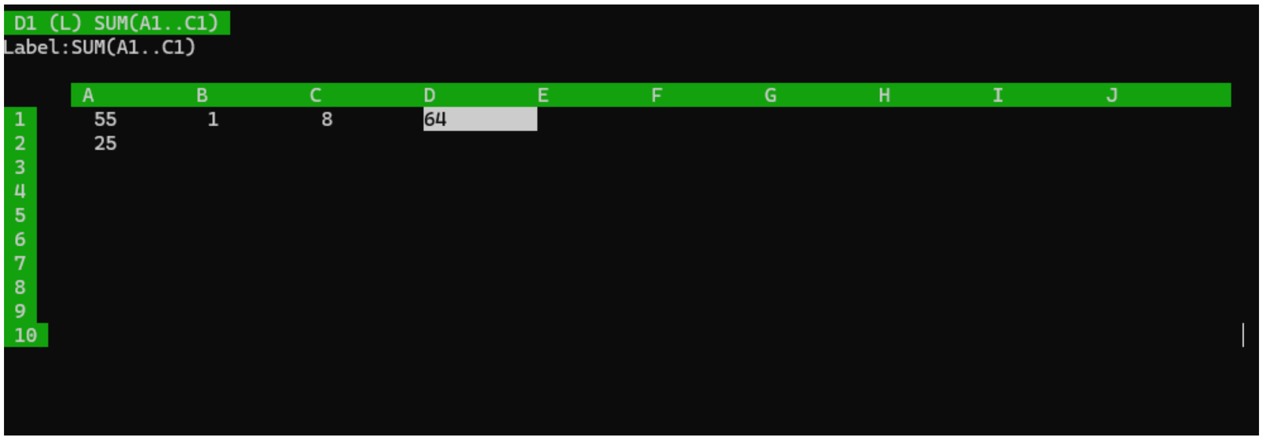
- Cell Referencing: Allowing users to reference other cells in formulas.
- Basic Arithmetic Calculations: Supporting addition, subtraction, multiplication, and division.
- CSV File Integration: Enabling the saving and loading of data for persistence and interoperability.
- Dynamic Updates: Automatically recalculating dependent cells when their references change.



```
1. Create New | 2. Select File | 3. Save File | 4. Save As | 5. Show Current File | q. Quit

Current File: Untitled
```

Figure 1 Initial Teminal Page



```
D1 (L) SUM(A1..C1)
Label:SUM(A1..C1)

    A         B         C         D         E         F         G         H         I         J
1    55        1         8        64
2    25
3
4
5
6
7
8
9
10
```

Figure 2SpreadSheed Terminal

NOTE: Created two class as Tokenizer Class and LexicalAnalysis Classes for FormulaParser. You can see this classes uml and their descriptions on the following sections.
During the development process, I utilized AI tools like ChatGPT to assist in documenting, understanding, and refining certain algorithms.
For example: Some algorithms, such as formula parsing and evaluation, were clarified or optimized with the help of AI.

Comments and documentation were enhanced to ensure the code is easy to follow and maintain.

Specific sections of the project where AI was utilized will be clearly highlighted and explained in the corresponding documentation.

2. Class Structure and UML Diagram

The project employs a modular design, with each class playing a distinct role in implementing the spreadsheet functionality. Below is an in-depth look at the class structure and their relationships, accompanied by an outline for a UML diagram.

2.1 Spreadsheet

Purpose: Manages the overall spreadsheet grid, including data storage, cell rendering, and formula evaluation.

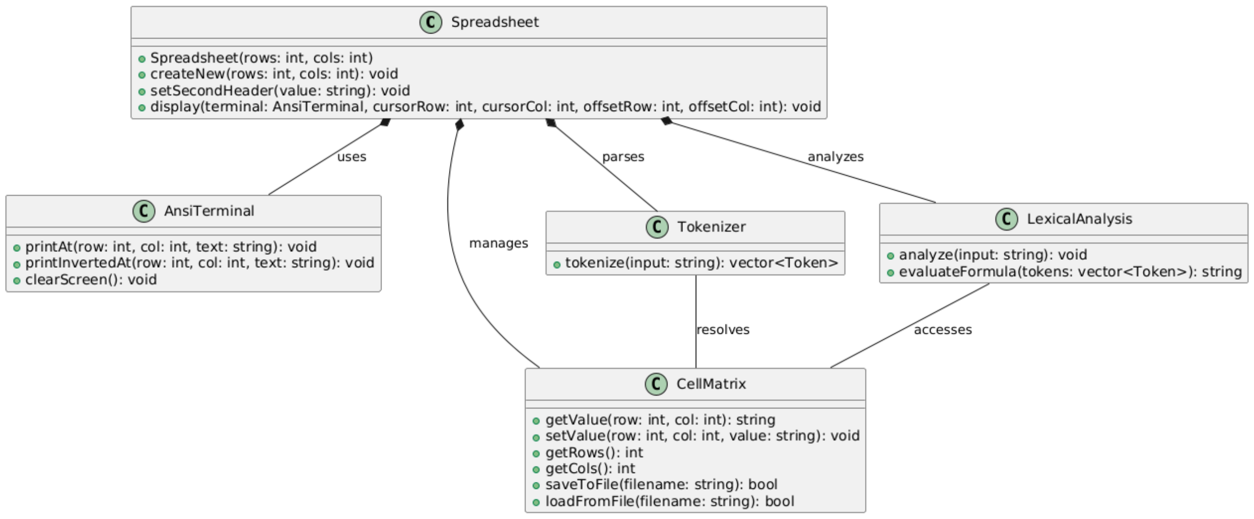

Spreadsheet
Spreadsheet(rows: int, cols: int)
createNew(rows: int, cols: int): void
setSecondHeader(value: string): void
display(terminal: AnsiTerminal, cursorRow: int, cursorCol: int, offsetRow: int, offsetCol: int): void
uses
manages
parses
analyzes
AnsiTerminal
printAt(row: int, col: int, text: string): void
printInvertedAt(row: int, col: int, text: string): void
clearScreen(): void
Tokenizer
tokenize(input: string): vector<Token>
LexicalAnalysis
analyze(input: string): void
evaluateFormula(tokens: vector<Token>): string
resolves
accesses
CellMatrix
getValue(row: int, col: int): string
setValue(row: int, col: int, value: string): void
getRows(): int
getCols(): int
saveToFile(filename: string): bool
loadFromFile(filename: string): bool

Figure 3 SpreadSheet Hierarcy UML Diagram

Key Methods:

- **setSecondHeader**(const std::string &value): Updates the second header to show active cell content and type.
- **display**(AnsiTerminal &terminal, ...): Renders the spreadsheet grid and headers on the terminal.
- **createNew**(int rows, int cols): Initializes a new spreadsheet with specified dimensions.

Attributes:

- **rows** and cols: The dimensions of the spreadsheet grid.
- **data**: A CellMatrix object holding the actual cell data.
- **secondHeader**: A string representing the status or content of the active cell.

Dependencies: Heavily interacts with AnsiTerminal, CellMatrix, and LexicalAnalysis.

2.2 CellMatrix

Purpose: Stores the spreadsheet data as a 2D grid and provides methods for accessing and modifying cell values.

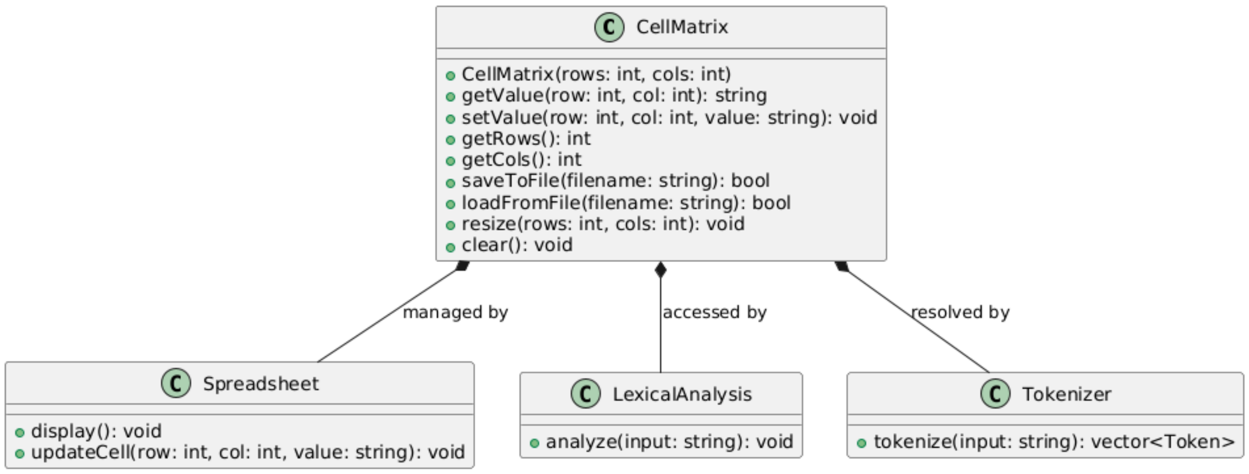

C CellMatrix
CellMatrix(rows: int, cols: int)
getValue(row: int, col: int): string
setValue(row: int, col: int, value: string): void
getRows(): int
getCols(): int
saveToFile(filename: string): bool
loadFromFile(filename: string): bool
resize(rows: int, cols: int): void
clear(): void
managed by
accessed by
resolved by
C Spreadsheet
display(): void
updateCell(row: int, col: int, value: string): void
C LexicalAnalysis
analyze(input: string): void
C Tokenizer
tokenize(input: string): vector<Token>

Figure 4 CellMatrix Hierarcy UML Diagram

Key Methods:

- **getValue**(int row, int col): Retrieves the value of a specific cell.
- **setValue**(int row, int col, const std::string &value): Updates the content of a specific cell.
- **resizeIfNeeded**(int newRow, int newCol): Dynamically resizes the grid to accommodate new rows or columns.
- **loadFromFile**(const std::string &filename): Loads data from a CSV file.

- **saveToFile**(const std::string &filename): Saves the spreadsheet data to a CSV file.

Attributes:

- data: A 2D std::vector storing strings for cell values.
- rows and cols: Track the current dimensions of the matrix.

Dependencies: Provides the primary data store for the Spreadsheet class and interacts with file I/O.

2.3 Tokenizer

Purpose: Parses formulas and cell content into tokens for further evaluation.

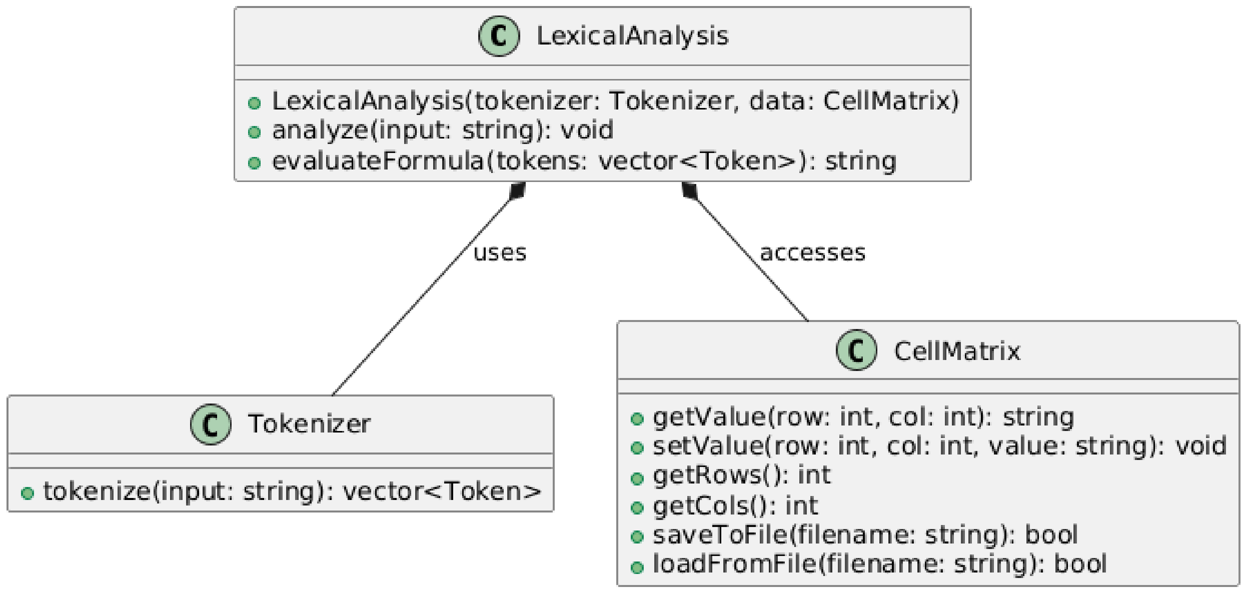

C LexicalAnalysis
LexicalAnalysis(tokenizer: Tokenizer, data: CellMatrix)
analyze(input: string): void
evaluateFormula(tokens: vector<Token>): string
uses
accesses
C Tokenizer
tokenize(input: string): vector<Token>
C CellMatrix
getValue(row: int, col: int): string
setValue(row: int, col: int, value: string): void
getRows(): int
getCols(): int
saveToFile(filename: string): bool
loadFromFile(filename: string): bool

Figure 5 Tokenizer Hierarchy UML Diagram

Key Methods:

- **tokenize**(const std::string &str): Converts a formula or string into a vector of tokens.
- **classifyToken**(const std::string &part): Identifies the type of a token (e.g., number, operator, formula).

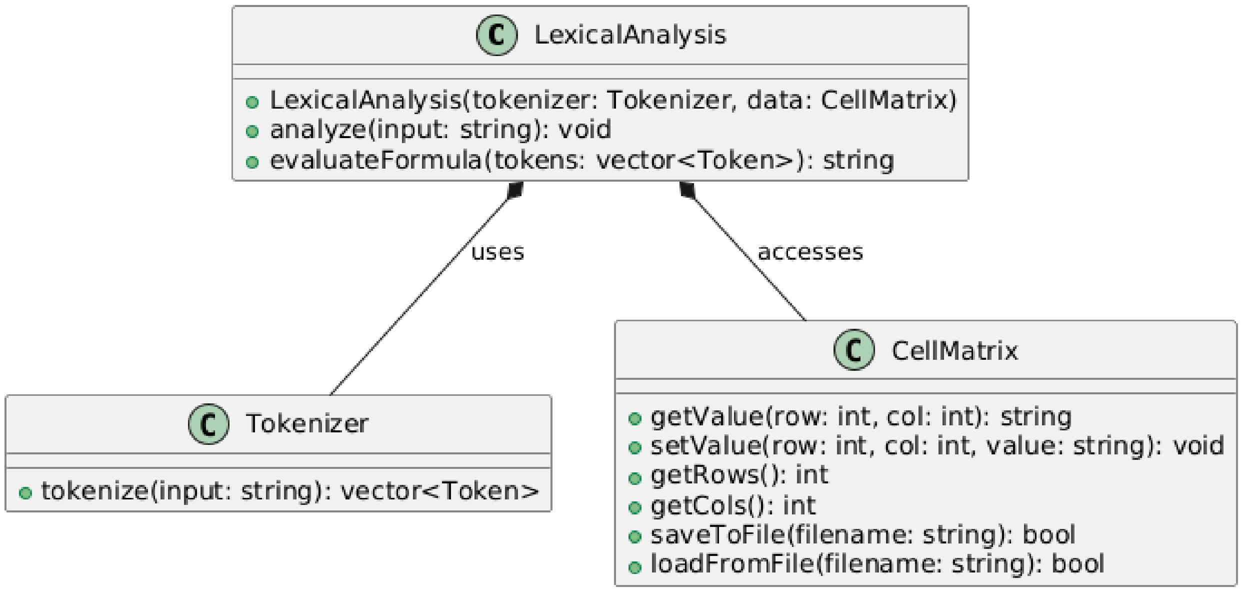
Attributes:

- **regexes**: A map of regex patterns for token matching.
- **operators** and formulaLabels: Lists of valid operators and functions.

Dependencies: Works closely with **LexicalAnalysis** for interpreting and evaluating formulas.

2.4 LexicalAnalysis

Purpose: Evaluates tokens from the Tokenizer and processes formulas for calculations.



LexicalAnalysis
LexicalAnalysis(tokenizer: Tokenizer, data: CellMatrix)
analyze(input: string): void
evaluateFormula(tokens: vector<Token>): string
uses
accesses
Tokenizer
tokenize(input: string): vector<Token>
CellMatrix
getValue(row: int, col: int): string
setValue(row: int, col: int, value: string): void
getRows(): int
getCols(): int
saveToFile(filename: string): bool
loadFromFile(filename: string): bool

Figure 6 Lexical Analysis Hierarchy UML Diagram

Key Methods:

- evaluateFormula(const std::vector<Token> &tokens): Evaluates a formula represented as a vector of tokens.
- getCellValue(const std::string &cell): Retrieves the value of a referenced cell.
- calculateRangeFunction(const std::string &label, const std::string &startCell, const std::string &endCell): Computes results for range-based functions like SUM or AVER.

Dependencies: Relies on Tokenizer for input parsing and CellMatrix for accessing data.

2.5 Main Flow

Purpose: Handles program logic, including the main menu and spreadsheet editing modes.

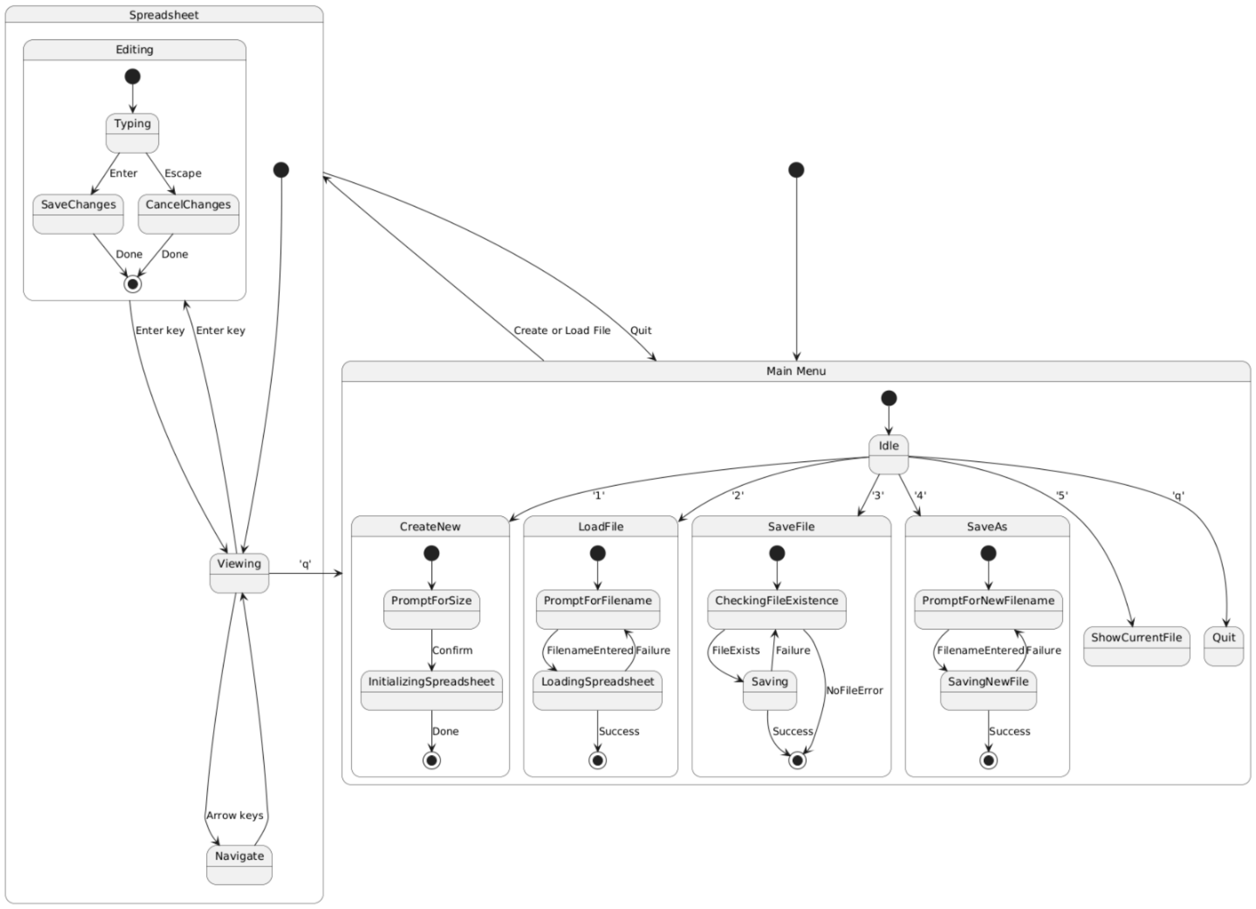


Figure 7 MainFlow Uml Diagram

Key Functions:

- handleMainMenu(...): Manages the main menu options like creating new files, saving, or loading.
- handleSpreadsheet(...): Handles user interactions while navigating and editing the spreadsheet.

Dependencies: Serves as the entry point and coordinates between all other classes.

3. Implementation Features

The ANSI Terminal-Based Spreadsheet Program includes several key features that demonstrate a strong understanding of object-oriented programming and terminal-based application design:

- **Data Entry and Storage:** Users can input numbers, text, or formulas into cells. The content is dynamically categorized as a numerical value (Value) or a label (Label).
- **Arithmetic Operations:** Supports addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`) directly in cell formulas (e.g., `=A1 + B2 - C3`).
- **Built-in Functions:** Includes support for `SUM`, `AVER`, `STDDEV`, `MAX`, and `MIN`, allowing calculations over a range of cells (e.g., `SUM(A1..A10)`).
- **Cell Referencing:** Enables direct referencing of other cells in formulas, including range references for built-in functions (e.g., `=B1 + AVER(C1..C5)`).
- **Automatic Recalculation:** Updates dependent cells whenever a referenced cell's value changes, ensuring all formulas remain consistent.
- **CSV File Support:** Users can save and load spreadsheets in CSV format, facilitating data persistence and compatibility with external tools.
- **Dynamic Terminal Interface:**
  - Displays a 2D grid with numbered rows and alphabetic columns.
  - Highlights the active cell and shows its content in a top-line display.
  - Allows navigation via arrow keys and displays a fixed window size (e.g., 10x10 cells).
- **Error Handling:** Manages common errors gracefully, such as invalid formulas, out-of-bounds cell references, and division by zero. **But User information should be improved**.

4. Missing Features

While the ANSI Terminal-Based Spreadsheet Program implements most of the required features, a few aspects remain incomplete or were simplified to meet project deadlines:

The terminal interface is functional but lacks advanced visual features like resizing the window dynamically or multi-line cell content display.

Feedback:
The program provides basic **feedback** for actions such as saving files or editing cells, but these messages could be more descriptive and user-friendly. For example:

- Notify the user if they try to save to an invalid file path.
- Provide clear feedback when entering an invalid formula or cell reference.

Long-Term Testing:

As a student project, this program was tested primarily for small to medium-sized spreadsheets with common scenarios.

However, long-term testing is necessary to ensure stability, performance, and accuracy in various edge cases, such as:

- Handling complex formulas and large datasets.
- Ensuring consistent behavior across different terminal environments.
- Detecting and resolving potential memory leaks or performance issues.

5. AI Assistance

During the development of this project, AI tools like ChatGPT were used to support various stages of implementation. Below are the details of where and how assistance was utilized:

Algorithm Design:

AI helped refine the formula parsing and evaluation logic. For example, the **LexicalAnalysis::evaluateFormula()** function was improved with AI suggestions to handle operator precedence and token stacking effectively.

Regex Construction:

Regex patterns for tokenizing formulas (e.g., recognizing SUM(A1..A5) or distinguishing numbers and labels) were validated and optimized with AI.

Code Documentation:

AI was used to generate clear and concise comments for complex sections of the code. For instance, the logic in Tokenizer::classifyToken() was documented using AI-generated templates to improve readability.

Error Handling:

Suggestions for improving error messages (e.g., returning Error: Division by zero instead of a generic failure message) were incorporated based on AI feedback.

Terminal Interface:

The layout and interaction flow in the Spreadsheet::display() function were enhanced with ideas generated by AI. Specifically, AI suggested the use of fixed-width formatting for cell content and inverted text for active cell highlighting.

Documentation and Reporting:

The structure and language of the project documentation, including this very document, were enhanced with AI assistance to ensure clarity and professionalism.