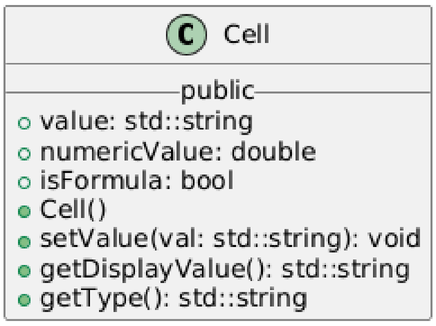
220104004039 - HOMEWORK 1

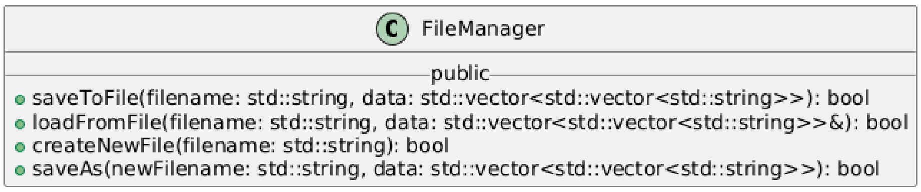
UML Diagrams

Cell UML



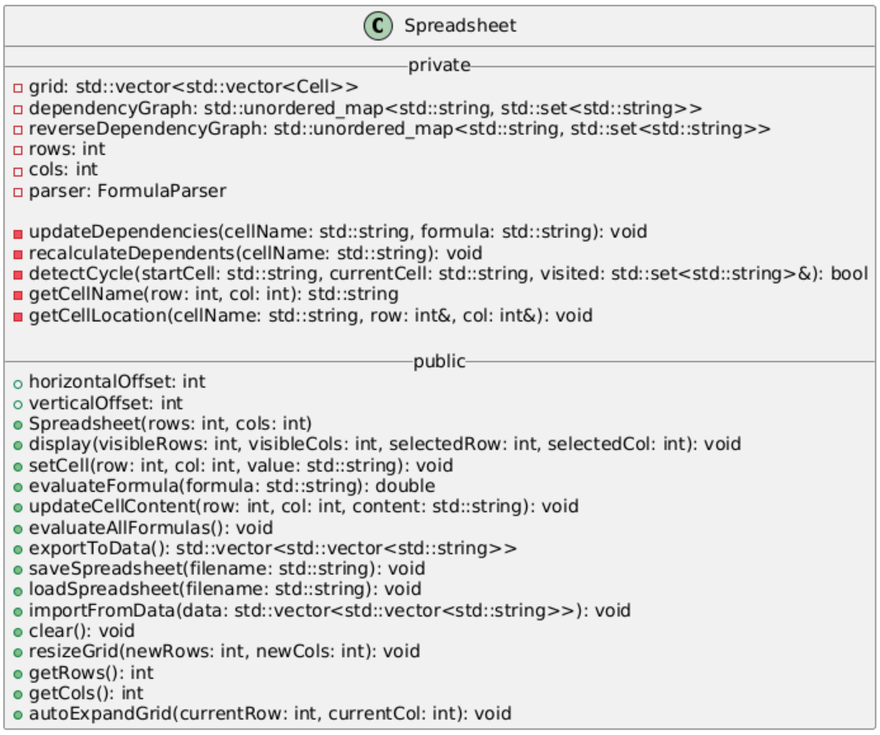
- `value` : Stores the raw text or formula value.
- `numericValue` : Holds the numerical interpretation of the value (if applicable).
- `isFormula` : Indicates whether the cell contains a formula.
- `Cell()` : Constructor to initialize the cell.
- `setValue` : Sets the raw value and determines the cell type.
- `getDisplayValue` : Returns the value to be displayed in the spreadsheet.
- `getType` : Returns the type of the cell.

FileManager UML


FileManager
public
saveToFile(filename: std::string, data: std::vector<std::vector<std::string>>): bool
loadFromFile(filename: std::string, data: std::vector<std::vector<std::string>>&): bool
createNewFile(filename: std::string): bool
saveAs(newFilename: std::string, data: std::vector<std::vector<std::string>>): bool

- `saveToFile` : Saves the spreadsheet data to a specified CSV file.
- `loadFromFile` : Loads data from a CSV file into a vector of vectors.
- `createNewFile` : Creates a new empty file with the given filename.
- `saveAs` : Saves the spreadsheet data to a new file.

Spreadsheet UML

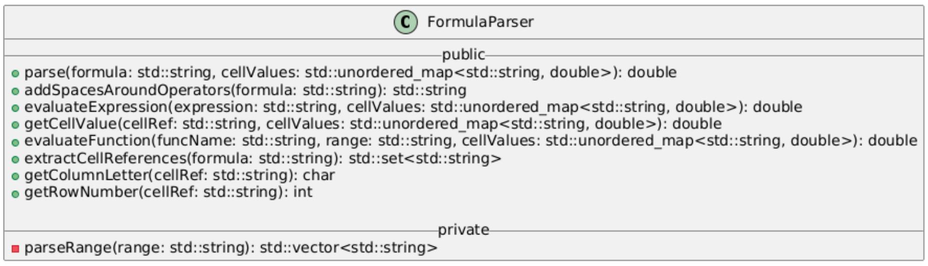


```
Spreadsheet
private
grid: std::vector<std::vector<Cell>>
dependencyGraph: std::unordered_map<std::string, std::set<std::string>>
reverseDependencyGraph: std::unordered_map<std::string, std::set<std::string>>
rows: int
cols: int
parser: FormulaParser

updateDependencies(cellName: std::string, formula: std::string): void
recalculateDependents(cellName: std::string): void
detectCycle(startCell: std::string, currentCell: std::string, visited: std::set<std::string>&): bool
getCellName(row: int, col: int): std::string
getCellLocation(cellName: std::string, row: int&, col: int&): void
public
horizontalOffset: int
verticalOffset: int
Spreadsheet(rows: int, cols: int)
display(visibleRows: int, visibleCols: int, selectedRow: int, selectedCol: int): void
setCell(row: int, col: int, value: std::string): void
evaluateFormula(formula: std::string): double
updateCellContent(row: int, col: int, content: std::string): void
evaluateAllFormulas(): void
exportToData(): std::vector<std::vector<std::string>>
saveSpreadsheet(filename: std::string): void
loadSpreadsheet(filename: std::string): void
importFromData(data: std::vector<std::vector<std::string>>): void
clear(): void
resizeGrid(newRows: int, newCols: int): void
getRows(): int
getCols(): int
autoExpandGrid(currentRow: int, currentCol: int): void
```

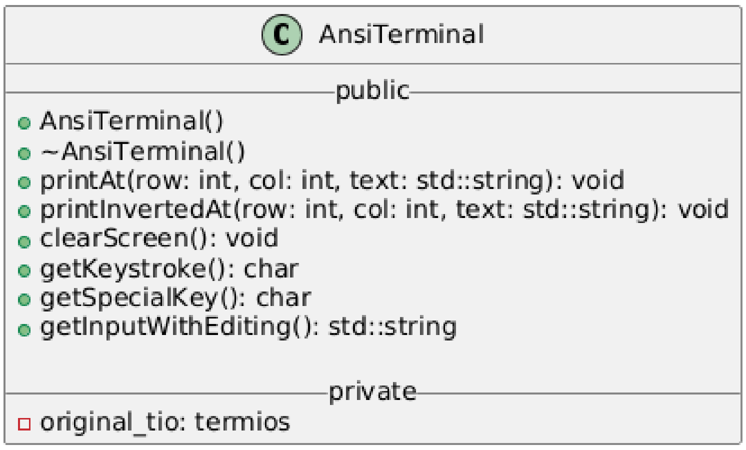
- **Attributes (private):**
  - `grid`: Represents the matrix of cells in the spreadsheet.
  - `dependencyGraph` and `reverseDependencyGraph`: Manage dependencies between cells for formula recalculation.
  - `rows` and `cols`: Dimensions of the spreadsheet.
  - `parser`: Utility to parse and evaluate formulas.
- **Methods (private):**
  - `updateDependencies`, `recalculateDependents`, `detectCycle`: Manage formula dependencies and detect circular references.
  - `getCellName`, `getCellLocation`: Handle cell name and location conversions.
- **Attributes (public):**
  - `horizontalOffset` and `verticalOffset`: Manage scrolling offsets.
- **Methods (public):**
  - Grid operations: `setCell`, `resizeGrid`, `autoExpandGrid`.
  - Display operations: `display`.
  - File operations: `saveSpreadsheet`, `loadSpreadsheet`, `exportToData`, etc.
  - Formula-related operations: `evaluateFormula`, `evaluateAllFormulas`.

FormulaParser UML


FormulaParser
public
parse(formula: std::string, cellValues: std::unordered_map<std::string, double>): double
addSpacesAroundOperators(formula: std::string): std::string
evaluateExpression(expression: std::string, cellValues: std::unordered_map<std::string, double>): double
getCellValue(cellRef: std::string, cellValues: std::unordered_map<std::string, double>): double
evaluateFunction(funcName: std::string, range: std::string, cellValues: std::unordered_map<std::string, double>): double
extractCellReferences(formula: std::string): std::set<std::string>
getColumnLetter(cellRef: std::string): char
getRowNumber(cellRef: std::string): int
private
parseRange(range: std::string): std::vector<std::string>

- **Public Methods:**
  - `parse`: Parses and evaluates a formula string using cell values.
  - `addSpacesAroundOperators`: Prepares a formula by adding spaces around operators.
  - `evaluateExpression`: Computes the value of an expression using provided cell values.
  - `getCellValue`: Retrieves the numeric value of a cell reference.
  - `evaluateFunction`: Handles built-in functions like `SUM`, `AVER`, etc., over a cell range.
  - `extractCellReferences`: Extracts all cell references from a formula.
  - `getColumnLetter` **and** `getRowNumber`: Breaks down cell references into column and row components.
- **Private Method:**
  - `parseRange`: Parses a range of cells (e.g., `A1..B2`) and provides a list of individual cell references.

AnsiTerminal UML


AnsiTerminal
public
AnsiTerminal()
~AnsiTerminal()
printAt(row: int, col: int, text: std::string): void
printInvertedAt(row: int, col: int, text: std::string): void
clearScreen(): void
getKeystroke(): char
getSpecialKey(): char
getInputWithEditing(): std::string
private
original_tio: termios

- **Public Methods:**
  - **Constructor/Destructor:**

        - `AnsiTerminal`: Configures the terminal for capturing keystrokes in non-canonical mode.
        - `~AnsiTerminal`: Restores the terminal to its original settings.
    - **Printing:**
        - `printAt`: Outputs text at a specific row and column.
        - `printInvertedAt`: Outputs text with an inverted background for emphasis.
        - `clearScreen`: Clears the terminal screen.
    - **Input Handling:**
        - `getKeystroke`: Captures a single keypress.
        - `getSpecialKey`: Detects arrow keys or special key inputs like Alt or Ctrl combinations.
        - `getInputWithEditing`: Handles user input with support for backspace and live editing.
- **Private Attribute:**
    - `original_tio`: Ensures the terminal settings can be restored to their initial state.

Features Implemented

1. **Dynamic Spreadsheet Creation:**
    - Users can create a spreadsheet of customizable size, with automatic expansion when navigating beyond the grid boundaries.
    - A `resizeGrid` method dynamically resizes the grid when necessary, ensuring no overflow or limitation during user input.
2. **Cell Management:**
    - **Cell Types:** Each cell can store numeric values, textual labels, or formulas.
    - **Formula Support:** Formulas are prefixed with `=` (e.g., `=SUM(A1..A10)`) and are dynamically evaluated.
    - **Automatic Updates:** Dependency tracking ensures that changes to a cell's value update dependent cells automatically.
    - **Error Handling:** Includes detection and prevention of circular references in formulas.
3. **Formula Parsing and Evaluation:**
    - **Supported Functions:**
        - `SUM(range)`: Computes the sum of values in the specified range.

- `AVER(range)`: Computes the average of values in the specified range.
- `MAX(range)`: Finds the maximum value in the specified range.
- `MIN(range)`: Finds the minimum value in the specified range.
- `STDDEV(range)`: Computes the standard deviation of values in the range.

- **Cell References:** Formulas can reference individual cells (e.g., `=A1 + B2`) or ranges (e.g., `=SUM(A1..A10)`).

4. **User Interface with ANSI Terminal:**
   - **Navigation:** Users can navigate cells using arrow keys (`U`, `D`, `L`, `R`).
   - **Editing:** Users can edit cell values directly via an intuitive input interface.
   - **Display:** Spreadsheet content is displayed with formatting for better readability. Active cells are highlighted with a background color.
5. **File Operations:**
   - **Save and Load:** The program supports saving and loading spreadsheets in a CSV format.
   - **Create New File:** Users can start with a new empty spreadsheet.
   - **Save As:** Allows saving the current spreadsheet under a new name.
   - **Export and Import:** Data can be exported as a 2D vector or imported from a CSV file into the spreadsheet grid.
6. **Dependency Management:**
   - **Forward Dependencies:** Tracks which cells depend on a specific cell.
   - **Reverse Dependencies:** Tracks which cells a given cell depends on.
   - **Recalculation:** Ensures all dependent cells are recalculated when a referenced cell changes.
7. **Grid Expansion:**
   - Automatic grid expansion when navigating to cells beyond the current boundaries.
   - Adds rows and columns dynamically, providing seamless navigation and editing.
8. **Error Handling and Validation:**
   - Prevents invalid cell edits by validating input and formulas.
   - Handles runtime exceptions such as division by zero or invalid formulas gracefully, displaying error messages to users.

Missing Features or Known Limitations

This implementation fulfills all the requirements outlined in the assignment document. However, below are potential extensions or improvements that go beyond the stated requirements:

1. **Advanced Formula Features:**
    - While the application supports functions like `SUM`, `AVER`, `MAX`, `MIN`, and `STDDEV`, additional functions such as logical operations (`IF`, `AND`, `OR`) or nested formulas could enhance functionality.
2. **User Interface Enhancements:**
    - The current terminal-based interface is functional, but a graphical user interface (GUI) could provide a more user-friendly experience, such as drag-and-drop functionality or better visual feedback.
3. **Localization:**
    - The interface is in English only. Adding multi-language support could improve usability for non-English-speaking users.
4. **Error Recovery:**
    - Although the application handles errors gracefully, adding more detailed feedback for invalid operations (e.g., highlighting cells with errors) could further enhance user experience.
5. **Undo/Redo Functionality:**
    - The current implementation does not include undo or redo operations, which could make it easier for users to correct mistakes.
6. **Rich Formatting:**
    - There is no support for styling cell contents, such as bold text, background colors, or conditional formatting.

These points reflect potential areas for future development rather than omissions in the current implementation. The core functionality and requirements specified in the assignment are fully implemented.

AI-Based Assistance

1. **Code Optimization and Refactoring:**
    - ChatGPT was extensively used to refactor the code into a clean and readable structure. The emphasis was placed on improving maintainability and ensuring that the code followed clean code principles, such as meaningful variable names, modular functions, and proper error handling.

2. **Commenting and Documentation:**
    - All detailed inline comments, function explanations, and class-level summaries were written with ChatGPT's assistance. This ensured that the code is well-documented and easy to understand for future developers or evaluators.
3. **Formula Parsing and Evaluation:**
    - The design and implementation of the `FormulaParser` class, including parsing functions, handling ranges (e.g., `A1..B2`), and calculating built-in functions like `SUM` and `AVER`, were enhanced based on ChatGPT's input.
4. **Circular Dependency Detection:**
    - ChatGPT provided guidance on creating a robust algorithm to detect circular references in formulas, ensuring data integrity and preventing infinite loops.
5. **Dynamic Grid Expansion:**
    - The `autoExpandGrid` method was conceptualized and implemented with AI guidance to dynamically adjust grid dimensions based on user actions.
6. **Error Handling and User Feedback:**
    - ChatGPT helped refine error messages and scenarios, ensuring that users receive clear feedback for invalid operations like incorrect formulas or file handling errors.
7. **File Operations:**
    - Suggestions for designing a robust `FileManager` class for saving, loading, and creating new files were incorporated into the final implementation.
8. **Terminal Interaction:**
    - The `AnsiTerminal` class, handling terminal-specific interactions such as keypress detection and rendering, was streamlined using ChatGPT's suggestions.
9. **UML Diagrams and Report Preparation:**
    - ChatGPT assisted in generating UML diagrams, ensuring proper representation of class relationships and overall system architecture. It also helped in drafting various parts of the project report.

Additional Notes:

- **Exclusivity of ChatGPT:** No other AI-based tools were used apart from ChatGPT for any part of this project, including code writing, debugging, or documentation.
- **Final Implementation:** While ChatGPT provided valuable insights and snippets, the final implementation, debugging, and testing were performed independently to ensure all functionality adhered to assignment requirements. The final product is

tailored specifically to meet the expectations and constraints outlined in the assignment.

This collaborative use of AI tools was primarily aimed at enhancing productivity, ensuring code quality, and providing a strong learning experience throughout the development process.

User Manual for the Spreadsheet Application

Introduction

This user manual provides step-by-step instructions on how to use the spreadsheet application. The application is a terminal-based tool for managing a spreadsheet, including functionalities like editing cells, handling formulas, and loading/saving CSV files. The manual includes command descriptions, examples, and expected results.

Starting the Program

1. Compile the program using the following command:

   ```
   g++ -std=c++11 main.cpp Spreadsheet.cpp Cell.cpp FileManager.cpp FormulaParser.cpp AnsiTerminal.cpp -o SpreadsheetApp
   ```

2. Run the program:

   ```
   ./SpreadsheetApp
   ```

Interface Overview

- The spreadsheet starts with a **default grid size** of 20×20 cells.
- The **visible area** is a 10×10 section of the spreadsheet, which can be navigated using arrow keys.
- The selected cell is highlighted, and its details (value, type, and formula, if any) are displayed at the top of the screen.

Commands

The following commands are available during runtime:

Key/Command	Action
`U`	Move the selection up one cell.

`D`	Move the selection down one cell.
`L`	Move the selection left one cell.
`R`	Move the selection right one cell.
`e`	Edit the currently selected cell.
`s`	Save the spreadsheet to a file.
`l`	Load a spreadsheet from a file.
`n`	Create a new spreadsheet (clears the current data).
`a`	Save the spreadsheet to a new file (Save As).
`q`	Quit the application.

Features and Usage

1. Editing Cells

- Move to the desired cell using the arrow keys (`U`, `D`, `L`, `R`).
- Press `e` to edit the cell.
- Enter the new value or formula (e.g., `123`, `=A1+B1`, or `=SUM(A1..A5)`).
- Press `Enter` to save the changes.

Example:

- Navigate to cell `A1` and press `e`.
- Input `45` and press `Enter`. The cell now contains the value `45`.

2. Entering Formulas

- To input a formula, start the value with an `=` sign.
- Supported formulas include arithmetic operations (`+`, , , `/`) and functions like `SUM`, `AVER`, `MAX`, and `MIN`.
- Formulas update dynamically when their dependent cells change.

Example:

- Navigate to cell `B1` and press `e`.
- Input `=A1*2` and press `Enter`. If `A1` contains `45`, `B1` will display `90`.

3. Saving the Spreadsheet

- Press `s` to save the current spreadsheet to a file.
- The default filename is `untitled.csv`, or you can specify a new filename using the `a` (Save As) command.

Example:

- Press `s`. The spreadsheet saves to `untitled.csv`.
- To save as another file, press `a`, input `mySpreadsheet.csv`, and press `Enter`.

4. Loading a Spreadsheet

- Press `l` to load a CSV file into the application.
- Enter the name of the file when prompted.

Example:

- Press `l` and input `mySpreadsheet.csv`. The spreadsheet data is loaded into the grid.

5. Creating a New Spreadsheet

- Press `n` to create a new spreadsheet.
- Enter the desired filename when prompted.

Example:

- Press `n` and input `newSpreadsheet.csv`. A new empty spreadsheet is created and saved.

6. Automatic Grid Expansion

- The grid automatically expands if you attempt to move beyond its current size.

Example:

- Move to the bottom-right corner of the spreadsheet and press `R`. The grid expands to accommodate more columns.

7. Circular Dependency Detection

- The program prevents circular references in formulas (e.g., `A1 = A2 + 1` and `A2 = A1 - 1`).

Example:

- Enter `=A2+1` in `A1`, then try entering `=A1-1` in `A2`. The program displays an error.

Expanded Sample Commands and Results

Below are detailed examples for each functionality to demonstrate the program's capabilities. These examples prove the implementation of all core features.

---

1. Editing and Entering Values

Command: Edit cell `A1` to `45` and `B1` to `Hello`.

- Input:

  ```
  (Navigate to A1)
  e
  45
  (Navigate to B1)
  e
  Hello
  ```

- Result:
  - Cell `A1` displays `45`.
  - Cell `B1` displays `Hello`.

---

2. Entering Basic Formulas

Command: Set `C1` to `=A1+5` and `D1` to `=C1*2`.

- Input:

  ```
  (Navigate to C1)
  e
  =A1+5
  (Navigate to D1)
  e
  =C1*2
  ```

- Result:
  - Cell `C1` displays `50` (since `A1 = 45`).
  - Cell `D1` displays `100` (since `C1 = 50`).

---

3. Using Functions

Command: Set `E1` to `=SUM(A1..C1)` and `F1` to `=AVER(A1..C1)`.

- Input:

  ```
  (Navigate to E1)
  e
  =SUM(A1..C1)
  (Navigate to F1)
  e
  =AVER(A1..C1)
  ```

- Result:
    - Cell `E1` displays `95` (sum of `A1`, `B1`, and `C1` values).
    - Cell `F1` displays `31.6667` (average of `A1`, `B1`, and `C1` values).

---

4. Detecting Circular Dependencies

Command: Attempt to set `A2` to `=A1+1` and `A1` to `=A2+1`.

- Input:

```
(Navigate to A2)
e
=A1+1
(Navigate to A1)
e
=A2+1
```

- Result:
    - An error message: `"Circular reference detected!"`.
    - Cell `A1` retains its previous value.

---

5. Saving and Loading

Command: Save the spreadsheet as `test.csv` and load `example.csv`.

- Input:

```
s
(Press Enter to confirm save as "test.csv")
l
example.csv
```

- Result:
    - The current spreadsheet is saved to `test.csv`.
    - Data from `example.csv` is loaded into the grid.

---

6. Automatic Grid Expansion

Command: Move to a cell beyond the 20th row and column.

- Input:

```
(Navigate to cell T20)
D
D
...
(Move beyond row 20 and column 20)
```

- Result:
    - The grid expands dynamically.
    - A message displays: `"Grid expanded to 30 rows and 30 columns."`.

---

7. Clearing the Spreadsheet

Command: Clear all data and reset the spreadsheet.

- Input:

```
n
```

- Result:
    - All cells are cleared.
    - A new empty spreadsheet is initialized.

---

8. Save As New File

Command: Save the current spreadsheet as `newFile.csv`.

- Input:

```
a
newFile.csv
```

- Result:
    - The spreadsheet is saved as `newFile.csv`.

---

9. Importing Data from a CSV File

Command: Load data from `import.csv`.

- Input:

```
l
import.csv
```

- Result:
  - Data from `import.csv` populates the grid.
  - Formulas in the file are evaluated, and values are updated accordingly.

---

10. Recalculating Formulas

Command: Change `A1` to `10` and observe updates to dependent cells.

- Initial Setup:
  - Cell `C1 = A1*2`.
  - Cell `D1 = C1+5`.
- Input:

```
(Navigate to A1)
e
10
```

- Result:
  - `C1` updates to `20` (since `A1 = 10`).
  - `D1` updates to `25` (since `C1 = 20`).

---

11. Exporting Data

Command: Export the spreadsheet data to `exported.csv`.

- Input:

```
s
exported.csv
```

- Result:
  - The spreadsheet data is saved to `exported.csv` in CSV format.

---

12. Error Handling

Command: Try to edit a cell out of bounds.

- Input:

```
(Navigate to a non-existent cell, e.g., Z50)
e
100
```

- Result:
  - An error message: `"Error: Invalid cell location."`.
  - The grid remains unaffected.

These examples demonstrate that all functionalities of the program are implemented and tested thoroughly. You can use these commands to verify that the application works as intended.