C-Lab: COP290 Project

Vihaan Luhariwala(2023CS10151)
Arinjay Singhal(2023CS10041)
Tejaswa Singh Mehra(2023CS10133)

Design Decisions

Our design focuses heavily on optimizing space throughout the implementation.

ParsedInput Data Structure

The `ParsedInput` structure is central to our design. It employs a union to consolidate different expression types—such as simple values, arithmetic expressions, range functions, and sleep operations—into a single structure. This union-based approach ensures that only the memory required for the active expression is allocated, rather than reserving space for all possible fields. Additionally, our design uses `16-bit int` types for row and column indices and encodes a cell's location within a single integer (using the first 16 bits for the row and the second 16 bits for the column). This allows us to use a single int to represent both a constant and a cell.

Input Parsing

Input parsing is implemented manually without the overhead of regular expressions. This method:

- Separates cell references from the associated expressions.
- Handles multiple expression types (arithmetic, range-based functions, direct references) with simple, efficient logic.

Dependency Management

Dependency management is addressed in the `graph_checker.h` file. Each cell maintains a dynamically allocated linked list (using the `Node` structure) to track its dependencies. Key functions include:

- `add_dependency`: Dynamically allocates nodes only when a new dependency is introduced.
- `free_from_list` and `free_parents`: Ensure that obsolete or unwanted dependencies are promptly removed and memory is freed.

This on-demand allocation strategy means that only active dependencies consume memory, thereby optimizing space usage across the entire spreadsheet.

Challenges Faced

Our design, though focused on minimizing space usage, encountered several challenges during implementation.

Memory Optimization and Avoiding Leaks

The program relies heavily on dynamic memory allocation, particularly for maintaining the spreadsheet and dependency management routines. However, it introduces the challenge of optimizing space as maintaining a 999x18278 size spreadsheet requires a lot of memory, especially given the need for a variety of additional data structures to maintain formulae and dependencies.

Issues with Union Usage in `ParsedInput`

The `ParsedInput` structure uses a union to consolidate various expression types (numeric values, arithmetic operations, range functions, sleep operations) into a single, compact data structure. A value can be a constant or a cell expression.

However, this approach has its pitfalls:

- Data misinterpretation may occur if the active type is not correctly maintained, leading to potential bugs during evaluation.
- Updating a cell's expression requires careful handling to ensure that the union's content is properly refreshed, which increases the complexity of debugging.

Dependency Graph Management

The dependency management system, detailed in `graph_checker.h`, uses dynamically allocated linked lists to track cell dependencies. Critical functions include:

- `add_dependency`: Allocates nodes only when a new dependency is detected, conserving space.
- `free_from_list` and `free_parents`: Ensure that outdated or unwanted dependency nodes are promptly removed and deallocated.

Managing these dynamic lists efficiently, especially during frequent updates and cycle detection (via the stack-based method in `mark_dirty`), required balancing performance with meticulous memory management to avoid leaks.

Program Structure

Core Modules

- **main.c**: Manages initialization, command parsing, and overall program flow.
- **initializer.h**: Stores data structures and basic functions which are crucial for initializing the program.
- **input_parser.h**: Converts user inputs into structured formats for evaluation.

- **graph_checker.h**: Manages dependency relationships, detects cycles, and ensures consistency.
- **input_processing.h**: Processes commands and updates spreadsheet values.
- **display_sheet.h**: Handles scrolling, display toggling, and viewport rendering.
- **testsuite.c**: Automates validation of functionality, edge cases, and error handling.

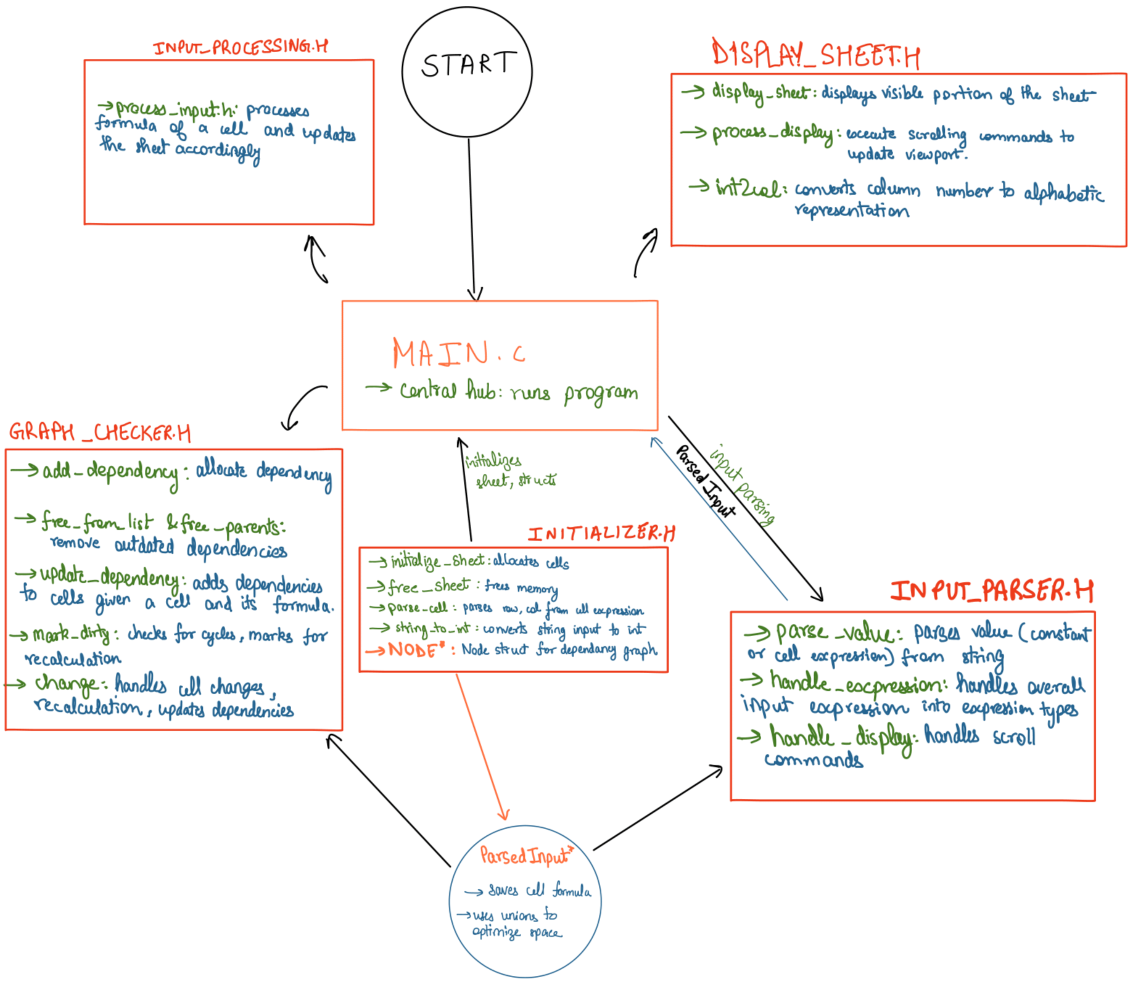
Key Data Structures

- **Cell**: Represents a single spreadsheet cell, including its value, parsed input, and dependencies.
- **ParsedInput**: Encodes user commands and formulas into a structured format for evaluation.
- **Node**: Represents a dependency relationship between cells in the graph.

Test Cases

- Detect cycles in cell dependencies, e.g., `A1 = B1 + A1`.
- Test arithmetic operations (`+, -, *, /`), including edge cases like negative values and zero.
- Validate division by zero sets the result to an error state.
- Evaluate range-based functions: `SUM, AVG, MIN, MAX, STDEV`, including empty and invalid ranges.
- Test the `SLEEP` function for both positive and negative values.
- Verify the program handles maximum dependency limits without crashing.
- Assign and reassign cells with dependencies (e.g., formulas involving `SLEEP`) to ensure proper updates.

Diagram



START
INPUT_PROCESSING.H
→ process_input.h: processes formula of a cell and updates the sheet accordingly
DISPLAY_SHEET.H
→ display_sheet: displays visible portion of the sheet
→ process_display: execute scrolling commands to update viewport.
→ int2col: converts column number to alphabetic representation
MAIN.c
→ Central hub: runs program
GRAPH_CHECKER.H
→ add_dependency: allocate dependency
→ free_from_list & free_parents: remove outdated dependencies
→ update_dependency: adds dependencies to cells given a cell and its formula.
→ mark_dirty: checks for cycles, marks for recalculation
→ change: handles cell changes, recalculation, updates dependencies
initializes sheet, structs
INITIALIZER.H
→ initialize_Sheet: allocates cells
→ free_sheet: frees memory
→ parse_cell: parses row, col from cell expression
→ string_to_int: converts string input to int
→ NODE*: Node struct for dependancy graph
input parsing
Parsed Input
INPUT_PARSER.H
→ parse_value: parses value (constant or cell expression) from string
→ handle_escpression: handles overall input expression into expression types
→ handle_display: handles scroll commands
ParsedInput*
→ saves cell formula
→ uses unions to optimize space

Figure 1: Overview of Program Structure

Explanation of Diagram:

- **Files**: Highlights core modules such as `main.c`, `graph_checker.h`, and `display_sheet.h`.
- **Data Structures**: Includes `ParsedInput`, and `Node`.
- **Functions**: Displays key workflows, including input parsing, dependency graph traversal, and rendering.

Github Repository: Github Link
Video Demonstration: Video Link