ELEC278 Assignment

Ryan Silverberg

20342023

I, Ryan Silverberg, attest that all the materials which I am submitting for this assignment are my own and were written solely by me. I have cited in this report any sources, other than the class materials, which I used in creating any part of this assignment. Furthermore, this work adheres to the policy on generative artificial intelligence as documented in the instructions.

Executive Summary

This product is a spreadsheet designed to support addition operations and custom formulas referencing other cells. The cells are also able to update based on their dependencies up to a certain extent (25 cells). It works by using a 2D array to model the entire spreadsheet and a tree system to update all dependent cells. It supports text, addition, and simple decimal numbers, along with dependent cell updates.

Design Proposal

To solve the spreadsheet problem at hand, I decided to use a 2D Array as the main structure to represent the spreadsheet as it resembles an excel spreadsheet. But within this array is a custom type used to house all data related to each cell. Accessing each element of the array is O(1) time as well to perform operations on it. This type would be a structure that consisted of four main components.

1. The textual value of the cell.
2. The evaluated value of the cell.
3. The dependency structure of the cell which contained within
    a. An array of pointers to its parents.
    b. An array of pointers to its children.
    c. The number of children.
    d. The number of parents.
4. The class structure which contained.
    a. The cell's type.
    b. Its location corresponds to the array using the Enums provided.

I believe this to be the best solution as it allows for all necessary data for a cell to be contained within one variable at a time. It also makes for readable paths in code when writing for dependencies. A good analogy to represent this data structure is houses on a street, everything one may need is contained within the house, which is why I believed that this data structure would work, it contains all the data.

This design will be able to satisfy the functional requirements due to being able to maintain its initial textual value inside of each cell structure and it will be able to recognize if the text is a formula by using an iterative solution to check the first character, and then iterate over each character to analyze and evaluate the formula at hand.

The 1st functional requirement was satisfied by just referencing the text value stored in each cell. It is an intuitive way of referencing each cell, this is O(1).

The 2nd functional requirement can be completely handled by the algorithm which iterates over each character in the formula and analyzes it for its type of input. This would be O(n) for each character. It would only progress through the string if the first character was either an equal's sign or a number. If those conditions were met along with the 2nd character not being invalid (i.e. 9e), then the helper function would detect the type of expression it is dealing with. If it is a number, it would parse the number. If it was a formula it would use another helper function to get the cell reference of the cell in the formula and then would use that to add the value from that parent cell to the overall sum. Then it would update the dependencies if the parent cell referenced was unique to the current cell. Then it would skip over the cell text in the string and check if the next character was a plus sign, otherwise it would return an error as it would become an invalid function. If it was true, it would continue and allow the loop to continue until the string only contains the null character.

The 3rd functional requirement was handled by using the textual value stored within the cell to display in the content window.

The 4th functional requirement will be satisfied by using an iterative loop to locate each dependent child and parent by referencing their location of the cell. This is effectively a tree structure as each cell recursively calls a cell set function which in turn rechecks if the cell has children and this continues until all cells are accounted for.

NFR 1, I know will not be completely optimal. But I know that this process of logic will work. There is a better way to implement this, specifically FR 4. The time complexity of this will be dependent on the size/depth of the tree.

In summary, the data structures implemented to solve this problem were a static 2D array, a dynamic array to store all pointers to parents/children, and a tree system based on each cell's dependencies.

Implementation

Each implementation of each functional requirement is listed below:

1st: This already works upon the base of the program visually. To implement this within the data structure, I freed all the text and set it to the null character, then set the eval_value to 0. After this, I had to remove the dependencies to not leave any dangling pointers. To get this to work I had to use my dependencies to recognize themselves. It basically becomes an acyclic graph. Each parent of the cell must validate that it does have the current cell as a child, if this is true, then it removes the connection between the two. This works inversely for the children. I looped through each child and checked if the cell existed within their parent array and removed the connection if it did.

2nd: Within the set_cell_value function, I created my own function called eval_expression, which took as parameters the text within the cell, a float pointer to contain the evaluated value, and a pointer to the cell that was currently being referenced. This would be vital for FR 4 and for error handling. The text in the cell would conditionally increase dependent on the initial character. This would act differently for each case.

For normal numbers, I used the strtof function to convert the single number into a float value. This dealt with handling float values.

For formulas, if the first character was an equal sign, it would evaluate the text character by character. IN the case of a cell reference. I created a helper function to dissect its reference into the correct Enums using the characters provided by the string. Then after I had the reference, I used another function to update the dependencies (FR 4) of the cell if it was a unique parent. This eliminated the problem of having two A1s as parents for example. It enabled the cells to be connected both ways, parent -> child and child -> parent. Whenever a + sign was reached, I recursively called eval_expression which started the process over again.

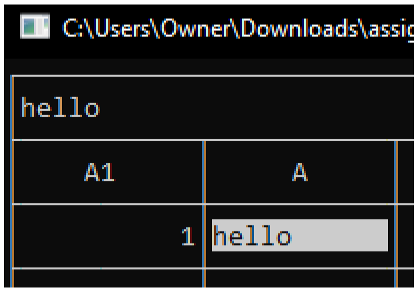
For text, if the initial character was not considered 'Valid' by my program, it would register in the cell's type as TEXT and the evaluated value would be 0 as to not have an impact on any dependent cells.

After some analysis, it seems that this is NOT the most optimal solution. As the time complexity seems to be $O(2^n)$. An example of this is shown as the dependencies continue to scale. Around 25 dependencies it takes a noticeable amount of time for the code to compute every dependency. This is because it is checking every possible cell that is dependent on each other and that scales with each new dependency. The tree becomes massive. If the spreadsheet is filled, it does not even finish.

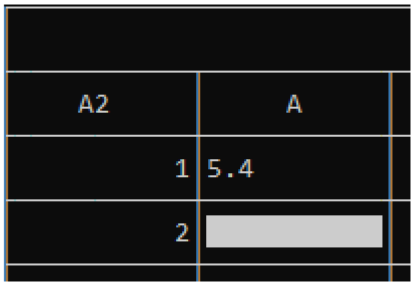
NFR 2 was handled well, all memory is freed upon cell clearance. There are no dangling pointers. NFR 3 is also covered as every function and mostly every line of code has a comment if it is not self-explanatory.

Testing

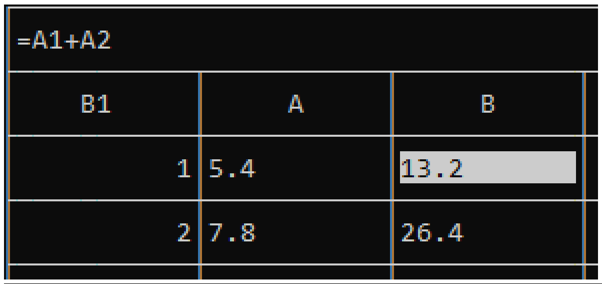
Testing was handled by using the program. The different test cases include:

- Textual Evaluation
  - 

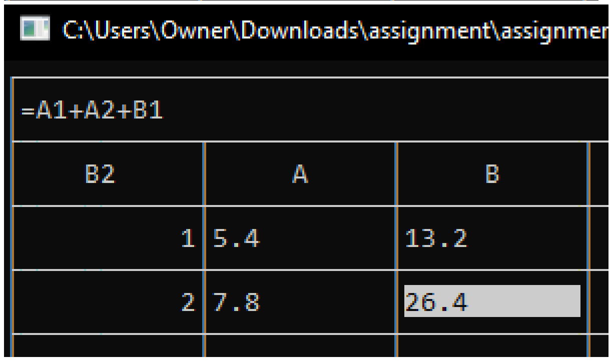
```
C:\Users\Owner\Downloads\assig
hello
      A1  |     A
         1|hello
```

- Numbers
  - 

```
      A2  |     A
         1|5.4
         2|
```

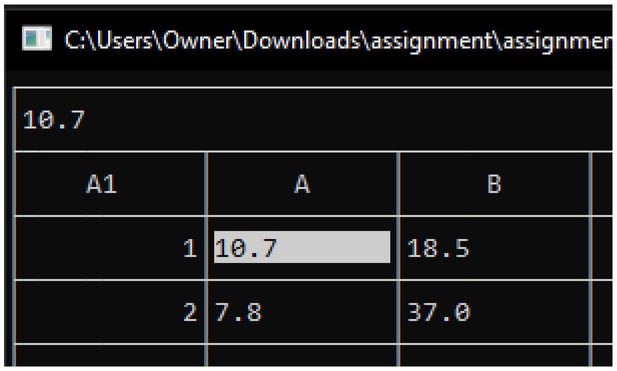
- Formulas
  - 

```
=A1+A2
      B1  |     A     |     B
         1|5.4        |13.2
         2|7.8        |26.4
```

  - 

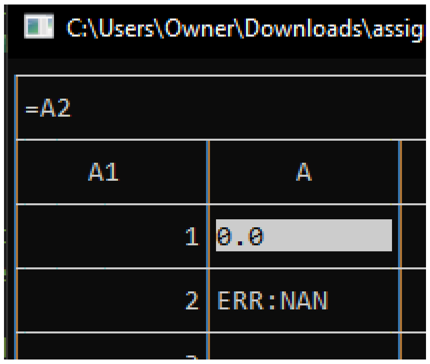
```
C:\Users\Owner\Downloads\assignment\assignmen
=A1+A2+B1
      B2  |     A     |     B
         1|5.4        |13.2
         2|7.8        |26.4
```

- Update Dependent Cells
  - Now I change A1 to 10.7

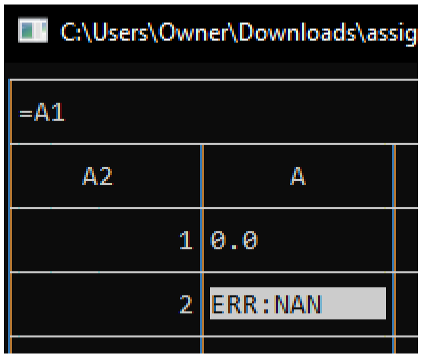


```
C:\Users\Owner\Downloads\assignment\assignmen
10.7
   A1  |  A   |  B
     1 | 10.7 | 18.5
     2 | 7.8  | 37.0
```

- Circular Dependencies

    ○ 

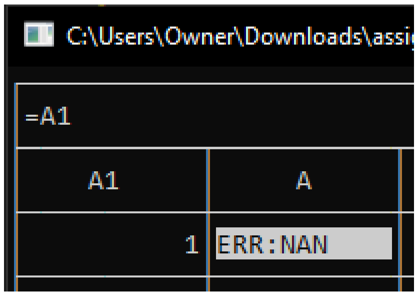
```
C:\Users\Owner\Downloads\assig
=A2
   A1  |  A
     1 | 0.0
     2 | ERR:NAN
```

    ○ 

```
C:\Users\Owner\Downloads\assig
=A1
   A2  |  A
     1 | 0.0
     2 | ERR:NAN
```

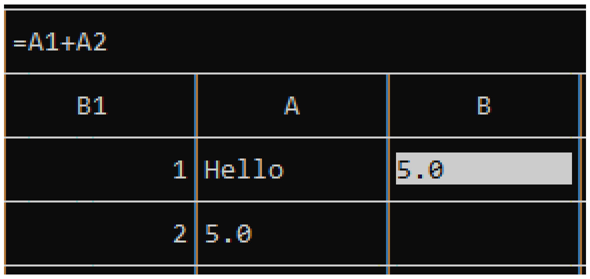
    ○ This recognizes that A2 = A1

- Self-Referencing

    ○ 

```
C:\Users\Owner\Downloads\assi
=A1
   A1  |  A
     1 | ERR:NAN
```

- Formula containing a cell with text.

- 

```
=A1+A2
      B1 |    A     |    B
        1| Hello    | 5.0
        2| 5.0      |
```

- This works as intended. The idea was to have if the cell was evaluated as a textual representation, its EVALUATED value would be 0, so it would not impact any cells that contained its formula. I can change the value in A1 to a number and it will still evaluate all the dependencies.

If I did it again

If I were to restart this assignment. I realize that a better way to do this problem, especially dependencies would be to use a graph and a stack to handle dependencies based on how far each child is the root node. This would compute all dependencies in order and not just be blindly searching for all the dependent cells.