ELEC 278 Final Assignment

Foster Ecklund

December 12th, 2023

Presented to

Mr. Nick Mertin and Dr. Jianbing Ni

I, Foster Ecklund, attest that all of 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

The report covers the design and implementation of a spreadsheet application in the C programing language. The application’s functionality includes storing text and numbers as well as evaluating formulas. Formulas can compute addition of numbers as well as other cells and common statistical functions such as Sum, Average, Minimum and Maximum. When Formulas depend on other cells the dependencies are recorded in order to prevent circular dependencies and update the necessary cells when values change.

The data structures and algorithms implemented provide the best possible time complexity to access values and update dependencies. The algorithm which handles circular dependencies is not the most ideal time complexity due to time limitations.

Information is included on the test that have been ran as well as ways to replicate the tests. All test results are provided in the report and appendix.

Table of Contents

[Executive Summary i](#_Toc152774674)

[Design Proposal 1](#_Toc152774675)

[Data Structure and Algorithm Choices 1](#_Toc152774676)

[Expected Behavior 1](#_Toc152774677)

[Implementation 1](#_Toc152774678)

[Data types 1](#_Toc152774679)

[Functions 2](#_Toc152774680)

[Memory Handling and Value Initialization 3](#_Toc152774681)

[Test cases 3](#_Toc152774682)

[Additional Functionality 3](#_Toc152774683)

[Statistical Analysis Functions 3](#_Toc152774684)

[Implementation 3](#_Toc152774685)

[Testing 3](#_Toc152774686)

[Testing Functions 3](#_Toc152774687)

[Testing Results 6](#_Toc152774688)

[Replication 6](#_Toc152774689)

[References 7](#_Toc152774690)

[Appendix A: Relevant Tables 8](#_Toc152774691)

[Appendix B: Interface.c Testing Results 12](#_Toc152774692)

Table of Figures

[Figure 1: Data type Definition in C 2](#_Toc152774707)

[Figure 2: Code to run Test 1 using run\_test\_1 4](#_Toc152774708)

[Figure 3: Code to run Test 2 using run\_test\_2 4](https://queensuca-my.sharepoint.com/personal/21fje_queensu_ca/Documents/278%20Assignment%20tests.docx#_Toc152774709)

[Figure 4: Code to run Test 3 using run\_test\_3 4](https://queensuca-my.sharepoint.com/personal/21fje_queensu_ca/Documents/278%20Assignment%20tests.docx#_Toc152774710)

[Figure 5: Code to run Test 4 using run\_tests\_4 5](#_Toc152774711)

[Figure 6: Code to Run Test 5 Using run\_test\_5 5](#_Toc152774712)

[Figure 7: Test results for Tests 1-5 and Main function of Testrunner.c 6](#_Toc152774713)

[Figure 8: Interface results of Test 1 12](#_Toc152774714)

[Figure 9: Interface results of Test 2. Before and After Changing the value of A1 13](#_Toc152774715)

[Figure 10: Interface Results of Test 3 Showing a Circular Dependency 13](#_Toc152774716)

[Figure 11: Interface Result of Test 4 13](#_Toc152774717)

[Figure 12: Interface Result of Test 5. Before and After Changing the Value of B1 14](#_Toc152774718)

Table of Tables

[Table 1: Worst Case time complexity of different data structures for different operations [1] 8](#_Toc152774719)

[Table 2: Expected Behavior of Different Functions and Input Values 9](#_Toc152774720)

[Table 3: Functions used in the Code to Implement the FRs 10](#_Toc152774721)

# Design Proposal

## Data Structure and Algorithm Choices

For this assignment the FRs require a data structure to store data for an excel type spreadsheet. This spreadsheet can contain text, number, and formulas. The only operation the program needs to do with this data structure is access values from specific indexes. According to

Table 1, the best option would be an array as the worst case time-complexity of access is O(1) satisfying NFR1, for this assignment a 2D Array is used. Stored within the array are structures which contain a type indicator, a value, and a string of text.

Additionally, FR4 requires “When the value of a cell changes, the displayed contents of all formula cells which (directly or indirectly) depend on it are updated” . This will require another data structure. For this a graph, more specifically a Directed Acyclic Graph (DAG), was implemented. The graph node contains the number of node used as well as a 2D array containing the, location of every node allowing us to access them in O(1) time satisfying NFR1.

Two algorithms that are required to implement FR4, one to update all dependent cells and one to check for circular dependencies so they can be handled gracefully.

The first algorithm is a depth first traversal of the dependency graph where it will update the cells with no dependencies first. From there it works in reverse updating each cell. This algorithm run in O(|V|+|E|) where V is the number of vertexes and E represents the number of edges in the graph. This is because the traversal goes through every vertex and down every single edge even if that node has already been visited. In scenarios where the graph could contain cycles this algorithm could be worse or not terminate, however we can assert our graph has no cycles using the following algorithm.

The second algorithm checks if there exists a circular dependency. My implementation of this algorithm uses a recursive check to assert whether any node in the graph is equal to another node. This algorithm run in O([V+E] ^2) since every node is compared to every other node. This was not the most efficient algorithm meaning it does not meet the NFRs. As a future improvement something such as Tarjan's strongly connected components algorithm could be implemented to compute this in O(V+E) [1].

## Expected Behavior

Throughout this assignment there are multiple cases where the spreadsheet needs to handle unexpected values and error. The error handling in this assignment was designed to prevent crashes or unexpected behaviour. All situation where unexpected values are given the program will alter the text in the cell where the error is present to “ERROR”. This process does not indicate what the error was caused by but does tell the user that there was an error without resulting in a crash.

Key errors that can happen include but are not limited to: Adding text, having a circular dependency, or inputting an invalid range. For the complete list of accepted values and errors see Table 2.

# Implementation

## Data types

There are 3 custom data types defined for the C implementation of this project, Cell\_content, Node, and Graph seen in Figure 1.

A computer screen with green text

Description automatically generated

Figure : Data type Definition in C

Cell\_content is used in an 2D array to store the values in the table. It stores the type of data stored using an Enumerated type, a Data value as a Double and a pointer to a char which stores a String. Graph stores a 2D array of Nodes and an Int representing the number of nodes used. Finally, Node stores its row and column in Integers and stores the number of dependencies. Node also has a 1D list of Node pointers. This enabled each node to link to the next Node.

## Functions

For each FR in the design Proposal the following functions are used to implement them.

FR1, set\_cell\_value and Clear\_cell are used to insert data and clear cells.

FR2, set\_cell\_value and Comp\_formula are used to insert data and formulas into cells and compute their results.

FR3, get\_textual\_value is used to provide the user with an editable textual value of the data stored in a given cell.

FR4, Update\_dependencies (called within set\_cell\_value) is used to update any cells that depend on the cell being edited. Additional functions, such as has\_circular\_dependency, are used to ensure proper error handling.

To implement the FRs many different helper functions were created in addition to the ones listed. These functions can be classified into three categories: Value handling Functions, Calculation Functions, and Dependency Functions. These functions are described in detail in Table 3

## Table 3Memory Handling and Value Initialization

In C memory handling is very important in order to prevent lifetime bugs and memory leaks. In the implementation of this project any dynamically allocated memory was either freed with in the function after it was no longer need or, in the case the memory was handed out of a function, proper notation was included. This satisfies both NFRs 2 and 3.

Value initialization is also very important in C to prevent unexpected behavior. In the implementation anytime a new variable was declared it was initialized to an adequate value. For counters and other integers, they were set to 0, for rows and column they were set to -1, and pointers were set to NULL.

## Test cases

Additional Functionality was implemented in order to run preset tests. An extra file was included in the starter code containing functions such as assert\_display\_text, assert\_edit\_text which assert the values displayed in the display and edit field. Using these two functions and set\_cell\_value, test cases could be written in the Test.c file. The testing procedure is described further in Testing.

# Additional Functionality

## Statistical Analysis Functions

It is common for spreadsheet applications, such as Excel, to have macro commands to compute commonly used statistical analytics of a range of cells such as Sum, Average, Min and Max. The implementation of project includes these 4 common functions. The required syntax for the functions is as follows:

=(operator)[start of range : end of range]

The valid Opperators are SUM, AVG, MIN and MAX and the start and end of range are any valid Coordinate in form A1.

All 4 statistical functions are able to run in O(N) where N is the number of cells in the range. This is the optimal time complexity as every cell in the range must be either added or compared depending on the function.

## Implementation

To implement this functionality more helper functions were required. These functions were integrated into the comp\_formula function to provide seamless integration with the existing addition functionality. Each Function is described in detail in Table 3.

# Testing

## Testing Functions

5 functions were created to evaluate the capabilities of the project called run\_test\_1 through run\_test\_5.Figure 2 depicts Test 1, it evaluates the Display and Edit text accuracy of Values and Formulas

A computer screen shot of text

Description automatically generated

Figure : Code to run Test 1 using run\_test\_1

Figure 3 and Figure 4 Depict Tests 2 and 3 which demonstrate both the ability for cells to update their values after a dependent cell changes and the Error handling in the case a circular dependency is created.

A screen shot of a computer code

Description automatically generated

Figure : Code to run Test 2 using run\_test\_2

A screen shot of a computer code

Description automatically generated

Figure : Code to run Test 3 using run\_test\_3

Figure 5 depicts test 4 which evaluates the edit and display values of the additional functions and Figure 6 depicts the dependency functionality of the additional functions.

A screenshot of a computer program

Description automatically generated

Figure : Code to run Test 4 using run\_tests\_4

A screenshot of a computer program

Description automatically generated

Figure : Code to Run Test 5 Using run\_test\_5

## Testing Results

In my testing all the test cases passed as is seen in Figure 7. The Function shown in Figure 7 is the main function the Testrunner.c executable. No functions were altered between the figures shown above and below.

A screen shot of a computer program

Description automatically generated

Figure : Test results for Tests 1-5 and Main function of Testrunner.c

Additionally, all 5 tests can be replicated in the interactive.c executable. Test results in the Interactive file can be found in the Appendix B: Interface.c Testing Results.

## Replication

All test can be replicated on the most recent version of the code found in the included .zip file. The only caveat is due to time limitations when debugging the code the Final code only runs consistently in Debugger mode. When ran in release, the code will run between 1 and 5 test cases successfully. With more time or a more experienced developer the code would be able to run with minor adjustments.

# References

|  |  |
| --- | --- |
| [1] | "Time complexities of different data structures," GeeksforGeeks, 2 November 2023. [Online]. Available: https://www.geeksforgeeks.org/time-complexities-of-different-data-structures/. [Accessed 4 December 2023]. |
| [2] | Tutorials Point, "C Library - <stdlib.h>," [Online]. Available: https://www.tutorialspoint.com/c\_standard\_library/stdlib\_h.htm. [Accessed 20 November 2023]. |
| [3] | DarkAtom, "Convert double to char\* in C," StackOverflow, 10 December 2020. [Online]. Available: https://stackoverflow.com/questions/65242668/convert-double-to-char-in-c. [Accessed 20 November 2023]. |
| [4] | Wikipedia, "Tarjan's strongly connected components algorithm," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Tarjan%27s\_strongly\_connected\_components\_algorithm. [Accessed 2 December 2023]. |
| [5] | W. Kahan, "IEEE Standard 754 for Binary Floating-Point Arithmetic," *Lecture Notes on the Status of IEEE 754,* vol. 1, p. October, 1997. |

# Appendix A: Relevant Tables

Table : Worst Case time complexity of different data structures for different operations [1]

| **Data structure** | **Access** | **Search** | **Insertion** | **Deletion** |
| --- | --- | --- | --- | --- |
| **Array** | O(1) | O(N) | O(N) | O(N) |
| **Stack** | O(N) | O(N) | O(1) | O(1) |
| **Queue** | O(N) | O(N) | O(1) | O(1) |
| **Singly Linked list** | O(N) | O(N) | O(N) | O(N) |
| **Doubly Linked List** | O(N) | O(N) | O(1) | O(1) |
| **Hash Table** | O(N) | O(N) | O(N) | O(N) |
| **Binary Search Tree** | O(N) | O(N) | O(N) | O(N) |
| **AVL Tree** | O(log N) | O(log N) | O(log N) | O(log N) |
| **Binary Tree** | O(N) | O(N) | O(N) | O(N) |

Table : Expected Behavior of Different Functions and Input Values

|  |  |  |
| --- | --- | --- |
| Functionality | Input value | Expected Behavior |
| Inserting in a number | Value between  5.0 x10^-345 and 1.7x10^308 | Display the value in Scientific notation |
| Inserting text | Any non-numerical string of characters | Display the string’s textual value |
| Inserting a function | Addition of and double type numbers | Display the value in Scientific notation |
| Addition of Cells that are not circularly dependent |
| Sum, Average, Minimum or Maximum of a valid range |
| Addition of letters or any non digit Characters | Displays the message “ERROR” |
| Addition of Cells that are circularly dependent |
| Sum, Average, Minimum or Maximum of an invalid range |

Table 3: Functions used in the Code to Implement the FRs

|  |  |
| --- | --- |
| Value handling Functions | |
| model\_init | Initialize the Model.  Set every cell type to Blank, Data to 0 and Text to “ “  Implemented using a for loop to iterate through every cell. |
| set\_cell\_value | Takes a string and inputs it into the designated row and column.  Determines whether the string is text, a number or a formula by first  checking whether the first char is “=”, where comp\_formula is used to compute the value of the formula  Then uses strtod to convert the numerical part of the string to a double.  If there is no textual component the cell is marked as a Data type otherwise the string is marked as Text  Finally updates the value of any nodes that depend on this cell using Update\_dependency |
| clear\_cell | Frees any memory stored in the text value.  Sets data to 0, Text to “ “ and Type to Blank |
| get\_display\_value | Returns a textual representation of what the display should show for a given cell.  Returns a number for Data and Formula types and text otherwise. |
| get\_textual\_value | Returns a textual representation of what the Edit field should show for a given cell.  Returns a Number for Data Cells and text for Text and Formula Cells |
| Calculation Functions | |
| coord\_to\_indicie | Converts a Coordinate stored in a string into a row and column.  Takes a 2- or 3-character string in the form A1 and converts into the respective row and column indexes (0,0) |
| comp\_formula | Takes a Formula starting with “=” as a String and parses through it conducting the proper calculations.  Firstly it checks if the string starts with any of the extra functions (SUM, AVG, MIN or MAX) using starts\_with and computes those as described in Additional Functionality  Using strtok to segment the formula into tokens separated by “+”, each token is evaluated and added to the total. If another cell is referenced a dependency is added using add\_dependency  Returns the sum of all tokens |
| starts\_with | As seen in Lab 4, Check whether a string starts with a prefix string.  Recursively checks each index until either there is a missed-match or the prefix ends  Returns true if the match false otherwise |
| error | Called when an error happens, Clears the Cell where the error happened then changes the Text to “Error” |
| Dependency Functions | |
| graph\_init | Initializes the Dependency Graph  Sets num of nodes to 0 and every pointer to NULL |
| add\_node | Adds a node to a location give by the row and col  Dynamically allocates memory for the node.  Sets row = row, col = col and number of dependents = 0  The node Is then added to the graph and the number of nodes in the graph is incremented |
| add\_dependency | First checks whether the dependency already exists, if so returns a success  Then add the dependent node to the dependency list of the source node and checks whether this introduces circular dependency,  If it does the dependency is removed and the function returns unsuccessful  Otherwise, the function is a success |
| has\_circular\_  dependency | Checks the graph for circularity  First Check: Do any of the First dependents of the source equal the reference  Second Check: Check for circular dependency by updating the source to each first dependent  Third Check: Check for circular dependency by updating the source and the reference to each first dependent  If any check returns true at any point the function ends returning true.  These checks ensure every node will be compared against every other node within the graph. |
| update\_  dependency | Used to update the value of cells that depend on the source  Iterates through all the dependent nodes to the further node first  Updates the formula in the cell and updates the display.  Returns through every node updating the value and display. |
| Functions for Additional Functionality | |
| get\_range | Converts a range of two coordinates in the form A1:B2 to two set of row and column.  This function uses coord\_to\_indicie to convert both sets of coordinates into indices. |
| add\_dependency\_to\_range | Add dependencies to a range using a starting row and column and an ending row and column.  Simply uses a for loop to iterate through every cell in the range and call add\_dependency |
| sum\_of\_range | Takes the sum of all the cells in a range using a for loop |
| avg\_of\_range | Takes the sum of all the cells in a range using a for loop then divides the sum by the number of cells to yield average |
| min\_of\_range | Find the minimum value of a set of cells given by a range |
| max\_of\_range | Find the maximum value of a set of cells given by a range |

# Appendix B: Interface.c Testing Results

A black grid with white text

Description automatically generated

Figure : Interface results of Test 1

A screenshot of a computer

Description automatically generatedA black screen shot of a black grid

Description automatically generated

Figure : Interface results of Test 2. Before and After Changing the value of A1

A screenshot of a computer

Description automatically generated

Figure : Interface Results of Test 3 Showing a Circular Dependency

A black grid with white text

Description automatically generated

Figure : Interface Result of Test 4

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Figure : Interface Result of Test 5. Before and After Changing the Value of B1