Homework Report: Enhanced ANSI Terminal-Based Spreadsheet Program

1. Introduction

In this report, I will describe the development of the enhanced version of the ANSI terminal-based spreadsheet program, which was completed as part of the CSE 241/501 course. The main goal of the project was to expand the basic functionality of the previous version by incorporating modern C++ techniques such as dynamic memory management, smart pointers, exception handling, namespaces, templates, and robust class hierarchy.

2. Starting the Project and Planning

At the beginning of the project, I reviewed the previous version to understand its structure and identify areas for improvement. This allowed me to clearly define the new requirements for the enhanced version. I set out to implement dynamic memory management, smart pointers, and more complex formula handling. After determining the goals, I began planning and implementing the project.

3. Dynamic Memory Management and Smart Pointers

The dynamic memory management section was initially challenging because I needed to avoid using STL containers and create a custom 2D array structure. However, after diving deeper into C++ dynamic memory management and smart pointers, I realized the process could be much more controlled and secure. I used std::unique_ptr for dynamically allocating memory for each cell, which ensured no memory leaks occurred. Initially, there were some mistakes, but once I fully understood how smart pointers work, the system functioned smoothly.

4. Abstract Cell Class and Class Hierarchy

This part of the project was one of the most enjoyable and straightforward for me. I created an abstract base class called `Cell` and derived classes for different types of cells. The derived classes included `FormulaCell` and `ValueCell`, which in turn had specialized sub-classes like `IntValueCell`, `StringValueCell`, and `DoubleValueCell`. Although the relationships between the classes were initially confusing, once everything was organized in a meaningful hierarchy, the project became much clearer.

5. Exception Handling

For error handling, I implemented C++ exceptions to catch and handle errors like invalid formulas and file operations. At first, I was unsure how to properly handle exceptions, but after some research, I successfully incorporated exceptions like `std::invalid_argument` to ensure the program was robust and could handle errors gracefully.

6. Templates and Modularity

I used templates to create reusable and modular components, such as the `Range` and `Grid` template classes. These templates made the program more flexible and extensible. This part of the project helped me realize how powerful templates can be in managing different types of data structures with a single codebase. The ability to use templates in this way enhanced the overall design and functionality of the program.

7. Formula Processing and Operators

Formula processing was one of the most interesting aspects of the project. Supporting operations such as `+`, `-`, `*`, `/` and functions like `SUM`, `AVERAGE`, `STDDEV`, and others required building a formula parsing system. Initially, it was difficult to implement a proper system, but once I had a clear structure for parsing and evaluating formulas, the cells dynamically updated their values as expected.

8. File Operations

For file operations, I added functionality to save and load spreadsheet data in CSV format. This was a bit tricky initially, but once I learned how to work with C++ file streams, the process became much easier. Being able to load and save data ensured that the program was more practical and could store and retrieve user data.

9. AI Assistance

During the project, I sought assistance from AI (ChatGPT) for creating the custom 2D array structure. This part of the project involved creating a dynamic 2D array without using STL containers, which was quite challenging. With the help of AI, I was able to quickly develop a solution that handled memory management properly and allowed for efficient data manipulation. The AI's guidance significantly sped up the process and helped me avoid mistakes.

10. Conclusion and Learning Experiences

By the end of the project, I felt that I had successfully met all the requirements and gained valuable experience with modern C++ features such as dynamic memory management, smart pointers, exception handling, and templates. Although I faced challenges at various points, I managed to overcome them and learn important programming concepts. I especially appreciated how templates

and smart pointers allowed for a more efficient and modular design.