READ ME:::::

* **Summarize the project and what problem it was solving.**

This project is an application that allows users to see how their investments will grow over time.

* **What did you do particularly well?**

This code implements an investment calculator that helps users visualize how their investment grows over a specified number of years, taking into account factors like the initial investment amount, monthly deposits, and annual interest rate.

The code effectively simulates investment growth scenarios, allowing users to understand how their investments could potentially accumulate over time based on the provided parameters. It demonstrates good programming practices by organizing code logically, providing user prompts and formatted output, and accurately calculating investment growth with compound interest.

* **Where could you enhance your code? How would these improvements make your code more efficient, secure, and so on?**

1. \*\*Input Validation\*\*: Add input validation to ensure that user inputs are within reasonable ranges and are of the expected data types. This will prevent unexpected behavior or crashes due to invalid inputs.

2. \*\*Encapsulation and Information Hiding\*\*: Consider encapsulating the class members by making them `private` and providing public methods for accessing or modifying them, adhering to the principles of information hiding and encapsulation.

3. \*\*Use of Constructors\*\*: Instead of initializing class members in the constructor body, use an initializer list in the constructor's declaration. This can improve performance and ensure proper initialization order.

4. \*\*Use of `const`\*\*: Mark parameters and member functions that don't modify the object as `const`. This improves code readability and helps prevent accidental modifications.

5. \*\*Separate Display Logic\*\*: Separate the table display logic from the calculation logic. This will make the code more modular and allow for easier customization of output formats.

6. \*\*Avoid Redundant Calculations\*\*: In the `calculateInvestmentGrowth` method, the interest is calculated twice, once for each balance calculation. You can calculate it once and use it for both balance calculations.

7. \*\*Performance Optimization\*\*: Since the vectors store data for each month, memory usage could increase significantly for long investment periods. Consider using a more memory-efficient data structure or limiting the data stored, especially if you're only interested in certain milestones (e.g., year-end balances).

8. \*\*Use Standard Library Algorithms\*\*: When working with vectors or other containers, consider using standard algorithms from the C++ Standard Library to simplify and improve the efficiency of certain operations.

9. \*\*Input Sanitization\*\*: Validate and sanitize user inputs to prevent potential security vulnerabilities, such as input that could lead to buffer overflows or code injection.

10. \*\*Error Handling\*\*: Implement explicit error handling mechanisms for cases where memory allocation fails or file operations encounter issues. Provide appropriate error messages to aid in debugging.

11. \*\*Unit Testing\*\*: Create unit tests to verify the correctness of individual functions and methods. This helps catch bugs early and ensures that future changes don't introduce regressions.

12. \*\*Documentation and Comments\*\*: Add comments and documentation to clarify the purpose and behavior of classes, functions, and methods. This makes the code more understandable for other developers and for future maintenance.

13. \*\*Namespace Usage\*\*: Consider placing your code within a namespace to avoid potential naming conflicts.

14. \*\*Code Reusability\*\*: If you anticipate needing similar functionality in the future, you could turn this into a library with reusable components.

15. \*\*Flexibility in Compounding Frequency\*\*: If you want to provide flexibility in compounding frequency (not just monthly), you could add an option for users to specify the compounding frequency (e.g., annually, semi-annually, quarterly).

* **Which pieces of the code did you find most challenging to write, and how did you overcome this? What tools or resources are you adding to your support network?**

All of it and Stack Overflow.

* **What skills from this project will be particularly transferable to other projects or course work?**

C++ itself.

* **How did you make this program maintainable, readable, and adaptable?**

Code's organization, clear naming, separation of concerns, and user-friendly design contribute to its maintainability, readability, and adaptability. It demonstrates good coding practices that help in avoiding complexity and making future modifications less error-prone and more straightforward.