**Code Review Walkthrough**

**Category One: Software Engineering and Design**

**Existing Code Functionality**

The artifact for this category is the **secure login authentication system** developed for the final iteration of my full-stack web application. The system validates user credentials, manages session tokens, and enforces input sanitization for safe user authentication.

**Checklist-Based Review**

**Structure**

* The code correctly implements the design by enforcing secure login and session handling.
* It follows consistent formatting, but some functions are longer than necessary and could be modularized.
* No unreachable code exists, but there are opportunities to replace custom validation with external libraries like express-validator.

**Documentation**

* Comments exist but are sparse in some areas. More descriptive function headers would improve clarity.

**Variables**

* Variable names are mostly meaningful, but a few abbreviations reduce readability.
* Type consistency is maintained, though explicit casting in some user input handling would strengthen safety.

**Loops and Branches**

* Conditional logic for handling login failures is correct but could benefit from a default error branch to catch unexpected states.

**Defensive Programming**

* Input is validated, but more robust length checks and stronger error handling should be added.
* File and session termination states are properly handled.

**Targeted Enhancements**

* Modularize longer functions into smaller, reusable components.
* Replace custom validation with a standardized library.
* Strengthen documentation with consistent commenting.

**Skills Demonstrated & Outcomes**

* Demonstrate secure coding and software engineering best practices.
* Aligns with course outcome: *Employ strategies for building collaborative environments that enable diverse audiences to support organizational decision-making in the field of computer science.*

**Category Two: Algorithms and Data Structures**

**Existing Code Functionality**

The artifact is a **binary search tree (BST) implementation** used to manage course data in the ABCU advising program. The BST allows insertion, search, removal, and in-order traversal of courses.

**Checklist-Based Review**

**Structure**

* The BST fully implements required design features, but search and insert methods could be optimized for readability.
* Code is well-formatted, but repeated logic exists in insertion and search functions.

**Documentation**

* The code contains basic comments, but lacks detailed explanations of algorithmic complexity.

**Variables**

* Variable names like node and temp are clear, but consistency can be improved across recursive calls.

**Arithmetic Operations**

* No floating-point comparisons are present, so safe from rounding concerns.

**Loops and Branches**

* Recursive logic is correct and properly nested.
* Some branches lack explicit else clauses, which can affect clarity.

**Defensive Programming**

* Pointers are checked before dereferencing.
* Input validation from external files could be improved to ensure malformed course data doesn’t crash the program.

**Targeted Enhancements**

* Reduce duplicate code by creating helper functions for insertion and search.
* Add comments explaining runtime complexity of each function.
* Enhance file input validation before inserting into the tree.

**Skills Demonstrated & Outcomes**

* Demonstrate algorithm optimization, modularization, and defensive coding.
* Aligns with course outcome: *Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution.*

**Category Three: Databases**

**Existing Code Functionality**

The artifact is a **hash table program storing course data** from a CSV file into memory for quick retrieval. Each course is mapped by its course number for O(1) lookup efficiency.

**Checklist-Based Review**

**Structure**

* The hash table is correctly implemented with insert, search, and print functions.
* No unneeded code, but collision resolution is basic and could be improved.

**Documentation**

* Some inline comments exist, but function headers are missing.

**Variables**

* Variable names are descriptive and consistent.
* Some unused variables remain in the hash function that could be removed.

**Arithmetic Operations**

* Division is safely handled, but modulus operations could be adjusted for better hash distribution.

**Loops and Branches**

* All branches are correct, but loop termination conditions could be documented for clarity.

**Defensive Programming**

* Array bounds are respected, but error handling for missing CSV files is minimal.
* Input data is not validated thoroughly before insertion into the hash table.

**Targeted Enhancements**

* Add advanced error handling for missing or malformed CSV input.
* Improve collision handling using techniques like double hashing or chaining.
* Expand documentation with function headers and complexity analysis.

**Skills Demonstrated & Outcomes**

* Demonstrate database-related optimization, error handling, and data structure enhancements.
* Aligns with course outcome: *Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.*

**ePortfolio Overall Skill Set**

1. **Skills and outcomes in the code review:**

* Secure software engineering and design principles
* Algorithmic analysis and optimization
* Efficient data storage and retrieval with databases

1. **Skills and outcomes in the narratives:**

* Reflecting on strengths and growth areas
* Explaining technical decisions in a professional context

1. **Skills and outcomes in the professional self-assessment:**

* Demonstrating readiness for real-world software development challenges
* Showcasing ability to design, analyze, and implement secure, efficient solutions