**A1: Hex Quiz Game Architecture Refactoring**

**Project Overview**

Congratulations on completing your initial hex conversion quiz game! This project builds upon your existing main.cpp to demonstrate advanced programming concepts including **software architecture**, **separation of concerns**, and **structured programming** without object-oriented classes.

You've successfully implemented a functional game, and now it's time to transform it into a professionally structured application that follows industry best practices.

**Learning Objectives**

By completing this project, you will:

* **Understand software architecture layers** (Frontend, Business Logic, Backend)
* **Apply separation of concerns** using function organization
* **Implement struct-based data management** with proper encapsulation
* **Practice memory management** with RAII pattern (read R-A-double-I, "Resource Acquisition is Initialization" where an object's constructor acquires a resource and its destructor releases it, ensuring cleanup with no memory leak)
* **Design modular code** that is maintainable and testable
* **Create comprehensive file management systems**

**Project Requirements**

1. **Starting Point**

You have a working main.cpp with these features:

* Random decimal generation (0-255)
* Hex conversion algorithm
* User input validation
* Basic scoring system
* File output for game history

1. **Refactoring Goals**

Transform your monolithic code into a well-structured application with:

**Central Data Structure**

* + Create a GameState struct to manage all game data
  + Implement proper constructor/destructor for memory management
  + Replace global variables with structured data

**Function Organization by Layer**

* + **Frontend Functions**: All user interaction and display
  + **Business Logic Functions**: Game mechanics and algorithms
  + **Backend Functions**: File operations and data persistence

**Enhanced Features**

* + Username system with validation
  + Comprehensive input validation
  + Three separate file outputs:
    - questions\_history.txt - Individual question log
    - session\_history.txt – High-level session summaries with statistics
    - decimalHistory.txt - Decimal batch storage of generated numbers
  + Real-time accuracy tracking
  + Professional user interface

**Memory Management**

* + Dynamic memory allocation for decimal storage
  + Proper cleanup with destructors
  + No memory leaks

**Architecture Requirements**

**GameState Struct Design**

struct GameState {

string username;

unsigned int points;

int totalQuestions;

int correctAnswers;

int\* decimals; *// Dynamic array*

int decimalCount;

const int ARRAY\_SIZE; *// = 3*

GameState(); *// Constructor*

~GameState(); *// Destructor*

};

**Function Categories**

**Frontend Functions (9+ functions)**

* User input collection and validation
* All display and output formatting
* Error message handling

**Business Logic Functions (8+ functions)**

* Game flow control
* Mathematical operations (hex conversion)
* Scoring and validation logic

**Backend Functions (4+ functions)**

* File I/O operations
* Data persistence
* Memory management for decimal array

**Deliverables**

1. **Refactored Source Code**
   * main.cpp - Complete refactored program
   * Proper commenting and documentation
   * Clean code formatting
2. **Testing Evidence**
   * Sample output files (questions\_history.txt, session\_history.txt, decimalHistory.txt)
   * Screenshots of program execution
   * Test cases demonstrating input validation
3. **Self-Assessment Grading Rubric**

**Instructions**: Read each criterion carefully and assign yourself points based on your implementation. Be honest in your assessment - this helps identify areas for improvement

1. **Submission:** Zip all the above and name the output in the format of A1\_lastname\_firstname.zip; submit the zip file.