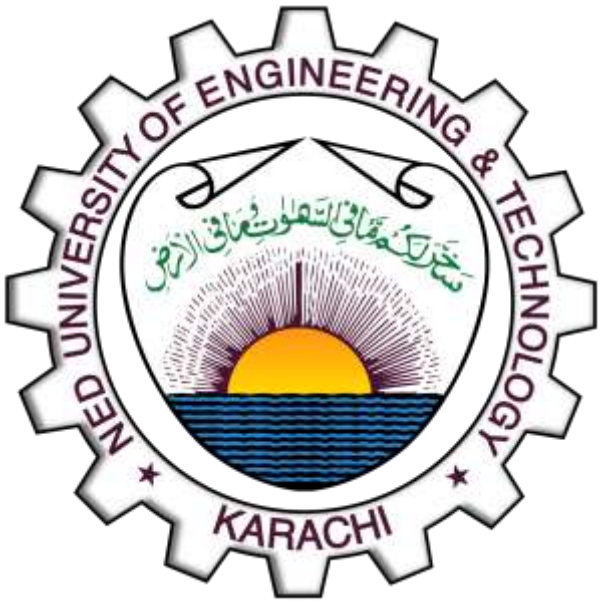
NED University of Engineering & Technology
Department of Computer Science and Information Technology



COMPLEX COMPUTING PROBLEM PROJECT REPORT

MULTI-SUBJECT MCQ QUIZ SYSTEM

Submitted to:
Sir Abdullah Ahmed

Course Title:
Programming Fundamentals (CT-175)

Submitted by:
Team: The Quizzards (Section B)

Areefa Samar (CT-25062)

Zainab Kinza Sheikh (CT-25059)

Muzna Rizwan Zubairy (GA-23007)

Submission Date: November 12, 2025

Contents

PROJECT TITLE

An Interactive Multi-Subject Quiz Game

ABSTRACT

This report documents the design, implementation and evaluation of a console-based, Multi-Subject, Multiple Choice Quiz System implemented in C language as part of the Complex Coding Problem (CCP) project. The application supports four subjects — English, Mathematics, General Knowledge and Geography — and delivers structured quizzes of ten questions per session. Questions are stored as external text files and are parsed at runtime to allow easy updates to the question bank without recompilation.

Key features implemented include question loading from simple text files, a strategic hint mechanism limited to three uses per quiz attempt, persistent leaderboard saved in a compact binary format, color-enhanced terminal output using ANSI escape sequences and UTF-8 compatibility, robust input validation and error handling. The codebase is modular: data structures, I/O, game engine, leaderboard management and utility functions are separated across dedicated functions. The implementation prioritizes portability and simplicity; dependencies are limited to the standard C library and basic platform APIs for console encoding where applicable. The program is lightweight, responsive and suitable for offline classroom use or self-study.

INTRODUCTION

Motivation

Traditional assessments often rely on paper tests which introduce delays for feedback, require manual grading, and do not offer immediate metrics for improvement. This project aims to create a small, self-contained assessment tool that gives feedback on completion of quiz, records attempts, and motivates learners through a persistent leaderboard.

Project Goals

The application targets learners and instructors seeking an offline quiz tool that is easy to extend. The program should be simple to run on typical student machines, require no network access, and be maintainable by editing plain text files that contain questions.

Design Principles

Simplicity, modularity and portability guided architectural decisions. The program uses clear data structures for questions and leaderboard entries, file-based persistence for easy maintenance, and a command-line user interface that works across platforms.

Objectives:

1. Offer quizzes spanning at least four different subjects.
2. Load quiz content sequentially from text files to allow question updates without recompilation.
3. Offer an intuitive, color-assisted console interface with clear prompts and feedback.
4. Provide learning support via a limited hint mechanism.
5. Track performance (score and time) and persist results for later review.
6. Rank attempts using a percentage-first and time-second comparator to balance accuracy and speed.
7. Ensure graceful error handling and robust input validation.

TECHNICAL TERMS

File I/O Terms

Binary file - Non-text file format (.dat)

Text file - Human-readable file format (.txt)

Append mode - "ab" (add to end of file)

Read mode - "rb" or "r" (read from file)

Memory & Buffer

Buffer - Temporary storage area

Sorting & Comparison

qsort() - Quick sort function

Comparator - Comparison function (compareScores)

Sorting algorithm - Method to arrange data

ANSI Escape Codes

ANSI - American National Standards Institute

Escape sequence - \x1b[(special character sequence)

Color code - Terminal text color

RESET - Return to default color

Platform Terms

UTF-8 - Unicode Transformation Format - 8-bit

Console - Text-based interface

Algorithm Terms

Sequential access - Linear reading

Case-insensitive - Ignores upper/lower case

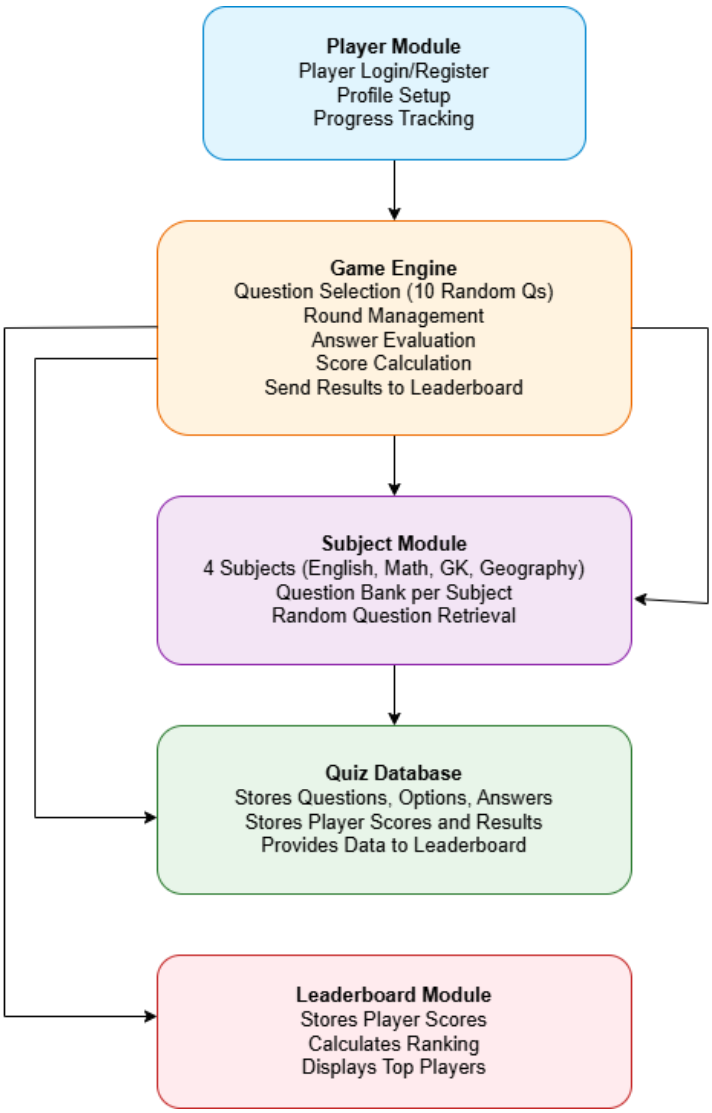
Compilation Terms

MinGW - Minimalist GNU for Windows

WORKING AND FLOW DIAGRAM

The system follows a modular architecture with a straightforward data flow:

This quiz system employs a modular architecture where five distinct components manage the application's flow and data. The Player Module handles user setup, while the Subject Module defines the content and accesses quiz questions from specific text files. The central Game Engine orchestrates the quiz, evaluating answers, calculating scores (including timing), and managing the console-based UI presentation. Finally, the Quiz Database module handles persistence routines, reading content from text files and writing the final session results as an append-only entry into a separate binary file, which the Leaderboard Module then processes to calculate and display rankings.



Modules Description

Module 1 — Header

This module defines the core data structures, and global constants used by other modules. It introduces the Question structure, storing the question text, four options, the correct answer, and a hint. It also defines LeaderboardEntry for player records, including name, subject, score, time taken, and date. Centralizing these definitions ensures consistent access to shared types and improves modularity, while updates propagate automatically at compile time.

Module 2 — Main Program

This module acts as the main controller for the application. It handles the display of the main menu, processes user input, and directs flow toward either taking a test, viewing the leaderboard, or exiting the application. It also manages screen clearing, colour settings, and UTF-8 console encoding (for special characters on Windows). Through a simple loop, it keeps the program running until the user chooses to quit. The main program coordinates between modules and provides the user-facing command-line interface for the entire system.

Module 3 — Question Loader

The question loader reads quiz questions from external .txt files for each subject (e.g., english.txt, maths.txt, gk.txt, geography.txt). It parses each question in a set sequence: question line, four options, correct answer, hint, and a separator. Up to ten questions are loaded into an array of Question structures. Error checks handle missing files or incomplete data to prevent runtime crashes. This design allows users to update questions by editing text files—no recompilation needed.

Module 4 — Game Engine

This is the heart of the application, controlling how quizzes are executed. It manages the flow of questions, hint availability (maximum 3 per session), input validation, and real-time scoring. The engine also measures the total time taken for the quiz using time functions and calculates the final score accordingly. Each round dynamically interacts with the user: displaying coloured questions and options, accepting input, and providing immediate feedback. Once the quiz ends, the game engine passes the player's score and time to the leaderboard module for recording.

Module 5 — Leaderboard Manager

The leaderboard module handles performance tracking and ranking. It stores quiz results in a binary file (leaderboard.dat) using append-only writes, ensuring that previous records are preserved. When viewing the leaderboard, all records are read, sorted, and displayed according to a comparison function that prioritizes accuracy (percentage) and, in case of ties, shorter completion time. The module provides persistent progress tracking between sessions and motivates players through a sense of competition. It includes safeguards for file-access errors and ensures efficient binary I/O operations.

Data Formats

Question File Format (Plain Text)

Each subject in the quiz system maintains its own external text file (for example english.txt, maths.txt, gk.txt, and geography.txt) containing the question bank. Every question follows a fixed eight-line structure, making it predictable and easy to parse using sequential file I/O. The order of lines ensures that each Question structure is filled consistently across subjects. The lines represent:

1. Question text
2. Option A
3. Option B
4. Option C
5. Option D
6. Correct answer (a single character: A/B/C/D)
7. Hint text
8. Separator (“---”)

During loading, the program reads these lines sequentially using standard file functions such as fgets() and fscanf(). This layout simplifies the logic inside the loadQuestions() function, allowing it to populate arrays of Question structures efficiently. By keeping the format human-readable, educators or developers can modify the questions without touching the program’s code—making the database extendable and user-friendly. The clear separation between data and logic reflects sound modular design principles and makes the quiz system adaptable to new subjects or question sets.

Leaderboard File (Binary Format)

The leaderboard data is stored in a compact binary file named leaderboard.dat. Each quiz attempt is saved as a serialized LeaderboardEntry record containing the player’s name, subject, score, total questions, date of attempt, and time taken in seconds. This design allows new results to be appended efficiently using fwrite() without overwriting previous entries. Binary storage ensures faster read/write performance, smaller file size, and consistent data alignment.

When displaying the leaderboard, entries are read using fread(), sorted with the built-in qsort() function and a custom comparison function, first by percentage score (score ÷ total) and then by completion time to resolve ties. The time_t field allows timestamps to be converted into readable dates via strftime(), supporting accurate and persistent recordkeeping between sessions with minimal disk usage.

Extended System Working

Execution Lifecycle

At startup, the program initializes essential console settings to prepare the environment for interaction. It loads predefined colour schemes, enabling clear visual feedback throughout the application. UTF-8 encoding is set to ensure proper display of all characters and symbols. The main() function then enters a continuous loop that listens for user inputs. This loop handles choices such as Take Test, View Leaderboard, or Exit, providing a predictable and stable flow.

Player registration

The first step is to ask for the player's name and the subject for which they like to take the quiz.

Question File Handling

When a subject is selected, the program opens the corresponding question file, for example english.txt. Each file is expected to follow a strict 8-line format including the question, options, correct answer, and hints. Buffered input ensures efficient reading, while newline and whitespace trimming keeps parsing clean. The system checks for missing or invalid entries and handles them safely to prevent crashes. This structure allows easy addition of new subjects without modifying core code.

Real-Time Quiz Flow

The quiz engine runs in an interactive loop, presenting one question at a time with its options. User input is validated using toupper() to ensure case-insensitive matching. Hints can be requested, and their usage is tracked for scoring purposes. Invalid or unexpected inputs trigger immediate error prompts without breaking the session. This design maintains a smooth, engaging quiz experience until all questions are answered.

Scoring and Feedback Logic

After each response, the system updates the score and keeps track of the total time taken. At the end, difftime() calculates the duration of the quiz for performance tracking. Results are stored in a LeaderboardEntry structure for persistence. Color-coded feedback enhances clarity: green for correct answers, yellow for statistics, and red for invalid attempts. Users can quickly review performance, including hint usage and time spent per question.

Persistent Record Management and Recovery

Once the quiz is completed, results are appended to a binary leaderboard file using fwrite(). When viewing the leaderboard, entries are read with fread() and sorted by percentage score and time using qsort(). Error handling ensures that invalid or incomplete data does not disrupt the program. Buffers are cleared as needed, allowing control to safely return to the main menu. This approach guarantees stability and a reliable experience for repeated use.

Key Algorithms (Pseudocode)

Question Loading

```
OPEN file
WHILE not EOF and count < 10:
   READ question line
   READ 4 option lines
   READ answer character
   READ hint line
   SKIP separator
   INCREMENT count
CLOSE file
RETURN count
```

Quiz Execution

```
FOR each question:
   DISPLAY question and options
   IF hints_available:
      OFFER hint (y/n)
   READ user answer
   VALIDATE and update score
DISPLAY final score, hints used and time
```

Leaderboard Sorting

```
READ all entries from leaderboard.dat
COMPUTE percentage = score / total_questions
SORT descending by percentage; if equal, ascending by time_taken
DISPLAY sorted table
```

CODE:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
#include <ctype.h>
#include "game.h"

#define MAX_QUESTIONS 10
#define MAX_HINTS 3

#define RED "\x1b[31m"
#define GREEN "\x1b[32m"
#define YELLOW "\x1b[33m"
#define BLUE "\x1b[34m"
#define CYAN "\x1b[36m"
#define LIGHT_CYAN "\x1b[96m"
#define RESET "\x1b[0m"

// Function declarations
int loadQuestions(Question q[], const char *filename);
int playQuiz(Question q[], int total);
void saveScore(LeaderboardEntry entry);
void clearScreen();
void line();
```

```
void takeTest();
void displayMenu();
void viewLeaderboard();
int compareScores(const void *a, const void *b);

int main() {

#ifdef _WIN32
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);
#endif

    int choice = 0;

    while (1) {
        displayMenu();
        printf(LIGHT_CYAN "Enter your choice: " RESET);
        if(scanf("%d", &choice)!=1) {
            while(getchar()!='\n'); // flush invalid input
            printf(RED "Invalid input! Try again.\n" RESET);
            continue;
        }
        while(getchar()!='\n'); // flush newline

        switch (choice) {
            case 1:
```

```
            takeTest();
            break;
        case 2:
            viewLeaderboard();
            break;
        case 3:
            printf(GREEN "Thank you for using MCQ Program!\n" RESET);
            exit(0);
        default:
            printf(RED "Invalid choice! Please try again.\n" RESET);
      }
   }
}

void displayMenu() {
   clearScreen();
   printf(CYAN "\n========================================\n" RESET);
   printf(GREEN "          MCQ QUIZ PROGRAM\n" RESET);
   printf(CYAN "========================================\n" RESET);
   printf(YELLOW "1. Take Test\n" RESET);
   printf(YELLOW "2. View Leaderboard\n" RESET);
   printf(YELLOW "3. Exit\n" RESET);
   printf(CYAN "========================================\n" RESET);
}

void takeTest() {
```

```
    int choice;
    char playAgain;
    Question questions[MAX_QUESTIONS];
    time_t start_time, end_time;
    LeaderboardEntry entry;
    const char *filename;

    do {
        clearScreen();
        printf(CYAN
"\n==================================================\n" RESET);
        printf(GREEN "          WELCOME TO MULTI-SUBJECT QUIZ GAME        \n" RESET);
        printf(CYAN
"==================================================\n\n" RESET);

        printf(LIGHT_CYAN "Enter your name: " RESET);
        fgets(entry.name, 50, stdin);
        entry.name[strcspn(entry.name, "\n")] = 0;

        printf(YELLOW "Choose a Subject:\n" RESET);
        printf(YELLOW "1. English\n2. Mathematics\n3. General Knowledge\n4. Geography\n"
RESET);
        printf(LIGHT_CYAN "\nEnter your choice (1-4): " RESET);
        if(scanf("%d", &choice)!=1) {
            while(getchar()!='\n');
            printf(RED "Invalid input! Returning to menu.\n" RESET);
            return;
        }
```

```
    while(getchar()!='\n');


    switch (choice) {
        case 1:
            strcpy(entry.subject, "English");
            filename = "english.txt";
            break;
        case 2:
            strcpy(entry.subject, "Mathematics");
            filename = "maths2.txt";
            break;
        case 3:
            strcpy(entry.subject, "General Knowledge");
            filename = "gk.txt";
            break;
        case 4:
            strcpy(entry.subject, "Geography");
            filename = "geography.txt";
            break;
        default:
            printf(RED "Invalid choice! Try again.\n" RESET);
            continue;
    }


    int total = loadQuestions(questions, filename);
    if (total == 0) {
```

```
            printf(RED "\nError: No questions found or file missing!\n" RESET);
            continue;
        }

        start_time = time(NULL);
        int score = playQuiz(questions, total);
        end_time = time(NULL);

        entry.score = score;
        entry.total_questions = total < MAX_QUESTIONS ? total : MAX_QUESTIONS;
        entry.test_date = end_time;
        entry.time_taken = (int)difftime(end_time, start_time);
        saveScore(entry);

        printf(LIGHT_CYAN "\nDo you want to play again? (y/n): " RESET);
        scanf(" %c", &playAgain);
        while(getchar()!='\n');

    } while (playAgain == 'y' || playAgain == 'Y');

    printf(GREEN "\nThank you for playing! Goodbye!\n" RESET);
}

int loadQuestions(Question q[], const char *filename) {
    FILE *fp = fopen(filename, "r");
    if (!fp) return 0;
```

```
    int i = 0;
    while (i < MAX_QUESTIONS && fgets(q[i].question, sizeof(q[i].question), fp)) {
        fgets(q[i].options[0], sizeof(q[i].options[0]), fp);
        fgets(q[i].options[1], sizeof(q[i].options[1]), fp);
        fgets(q[i].options[2], sizeof(q[i].options[2]), fp);
        fgets(q[i].options[3], sizeof(q[i].options[3]), fp);

        fscanf(fp, " %c\n", &q[i].correctAnswer);  // now expects A/B/C/D
        fgets(q[i].hint, sizeof(q[i].hint), fp);

        char separator[10];
        fgets(separator, sizeof(separator), fp);  // skip ---
        i++;
    }

    fclose(fp);
    return i;
}

int playQuiz(Question q[], int total) {
    clearScreen();
    int score = 0, hintsUsed = 0;
    char answer, useHint;
    char optionLabels[] = {'A', 'B', 'C', 'D'};
```

```
    for (int i = 0; i < total; i++) {
        clearScreen();
        line();
        printf(BLUE "Question %d of %d\n" RESET, i + 1, total);
        line();

        printf(LIGHT_CYAN "%s\n" RESET, q[i].question);
        for (int j = 0; j < 4; j++) {
            printf(YELLOW "%c. %s" RESET, optionLabels[j], q[i].options[j]);
        }

        if (hintsUsed < MAX_HINTS) {
            printf(CYAN "Would you like a hint? (y/n): " RESET);
            scanf(" %c", &useHint);
            while(getchar()!='\n');
            if (useHint == 'y' || useHint == 'Y') {
                printf(GREEN "%s\n" RESET, q[i].hint);
                hintsUsed++;
            }
        }

        printf(LIGHT_CYAN "\nEnter your answer (A/B/C/D): " RESET);
        scanf(" %c", &answer);
        while(getchar()!='\n');

        if (toupper(answer) == toupper(q[i].correctAnswer)) {
```

```
            score += 1;  // each correct answer gives 1 point
        }

        if (i < total - 1) {
            printf(LIGHT_CYAN "\nPress Enter for next question..." RESET);
            getchar();
        }
    }

    clearScreen();
    printf(CYAN "\n==========================================\n" RESET);
    printf(GREEN "            QUIZ COMPLETED!\n" RESET);
    printf(CYAN "==========================================\n" RESET);
    printf(YELLOW "Your Total Score: %d / %d\n" RESET, score, total);
    printf(YELLOW "Hints Used: %d / %d\n" RESET, hintsUsed, MAX_HINTS);

    return score;
}

void saveScore(LeaderboardEntry entry) {
    FILE *fp = fopen("leaderboard.dat", "ab");
    if (!fp) {
        printf(RED "Error saving score!\n" RESET);
        return;
    }
```

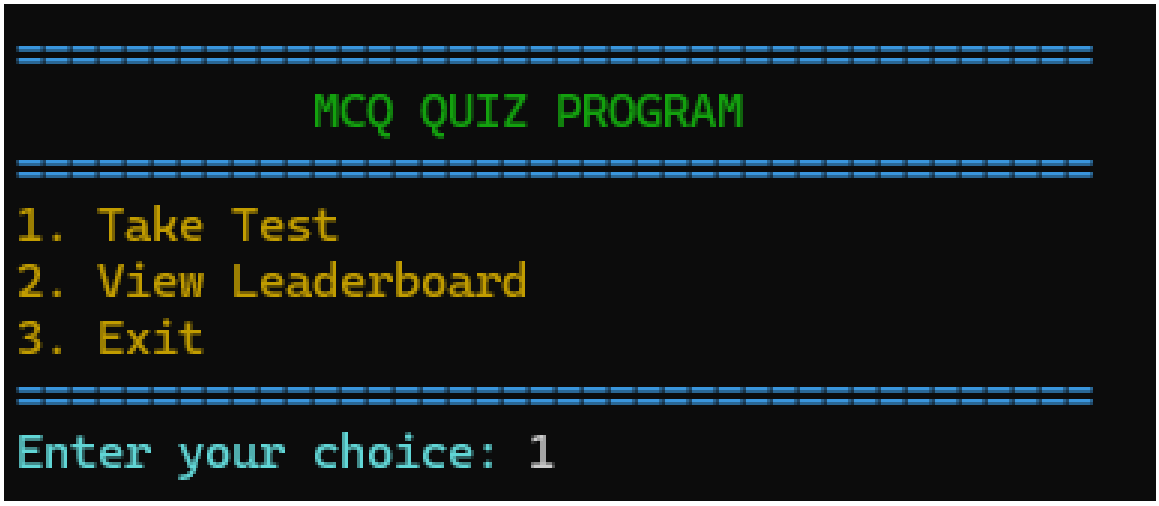
```
    fwrite(&entry, sizeof(LeaderboardEntry), 1, fp);
    fclose(fp);

    printf(GREEN "\nYour score has been saved!\n" RESET);
}

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

void line() {
    printf(CYAN "-------------------------------------------------\n" RESET);
}

void viewLeaderboard() {
    clearScreen();
    FILE *fp = fopen("leaderboard.dat", "rb");
    LeaderboardEntry entries[1000];
    int count = 0;
    char time_str[26];

    if (!fp) {
```

```
        printf(RED "No leaderboard data found!\n" RESET);
        return;
    }

    while (fread(&entries[count], sizeof(LeaderboardEntry), 1, fp) == 1) count++;
    fclose(fp);

    if (count == 0) {
        printf(RED "Leaderboard is empty!\n" RESET);
        return;
    }

    qsort(entries, count, sizeof(LeaderboardEntry), compareScores);

    printf(CYAN
"\n=========================================================================
======================\n" RESET);
    printf(GREEN "          LEADERBOARD\n" RESET);
    printf(CYAN
"=========================================================================
======================\n" RESET);
    printf(YELLOW "%-4s %-20s %-20s %-10s %-10s %-20s\n" RESET,
           "Rank", "Name", "Subject", "Score", "Time(s)", "Date");
    printf(CYAN
"=========================================================================
======================\n" RESET);

    for (int i = 0; i < count; i++) {
        strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
```

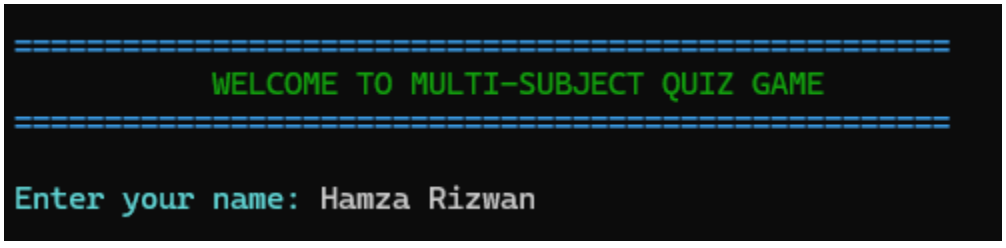
```
            localtime(&entries[i].test_date));
        char score_str[20];
        sprintf(score_str, "%d/%d", entries[i].score, entries[i].total_questions);
        printf(YELLOW "%-4d %-20s %-20s %-10s %-10d %-20s\n" RESET,
               i + 1,
               entries[i].name,
               entries[i].subject,
               score_str,
               entries[i].time_taken,
               time_str);
    }
    printf(CYAN
"=============================================================================
====================\n" RESET);
    printf(LIGHT_CYAN "\nPress Enter to return to menu..." RESET);
    getchar();
}

int compareScores(const void *a, const void *b) {
    LeaderboardEntry *entryA = (LeaderboardEntry *)a;
    LeaderboardEntry *entryB = (LeaderboardEntry *)b;

    float percentA = (float)entryA->score / entryA->total_questions;
    float percentB = (float)entryB->score / entryB->total_questions;

    if (percentB > percentA) return 1;
    if (percentB < percentA) return -1;
```

```
    return entryA->time_taken - entryB->time_taken;
}
```

Output:



TC01_Main_Menu_Display
Displays the main menu with three options: Take Test, view Leaderboard, exit



TC02_User_Name_Input
Prompts user to input their name before starting the quiz.

```
============================================================
            WELCOME TO MULTI-SUBJECT QUIZ GAME
============================================================

Enter your name: Hamza Rizwan
Choose a Subject:
1. English
2. Mathematics
3. General Knowledge
4. Geography

Enter your choice (1-4): 3
```

TC03_Category_Selection

Displays available quiz categories (e.g., General Knowledge, Mathematics).

```
------------------------------------------------
Question 1 of 10
------------------------------------------------
Q1. What is the largest organ in the human body?

A. Heart
B. Liver
C. Brain
D. Skin
Would you like a hint? (y/n): y
Hint: It covers your entire body and protects you from the outside world.


Enter your answer (A/B/C/D): B

Press Enter for next question...
```

TC04_Question_Display_Hint_Accepted

User selects Yes to view a hint. Hint appears, user inputs valid answers.

```
------------------------------------------------------
Question 2 of 10
------------------------------------------------------
Q2. Who was the first person to walk on the Moon?

A. Buzz Aldrin
B. Neil Armstrong
C. Yuri Gagarin
D. Michael Collins
Would you like a hint? (y/n): n

Enter your answer (A/B/C/D): C

Press Enter for next question...
```

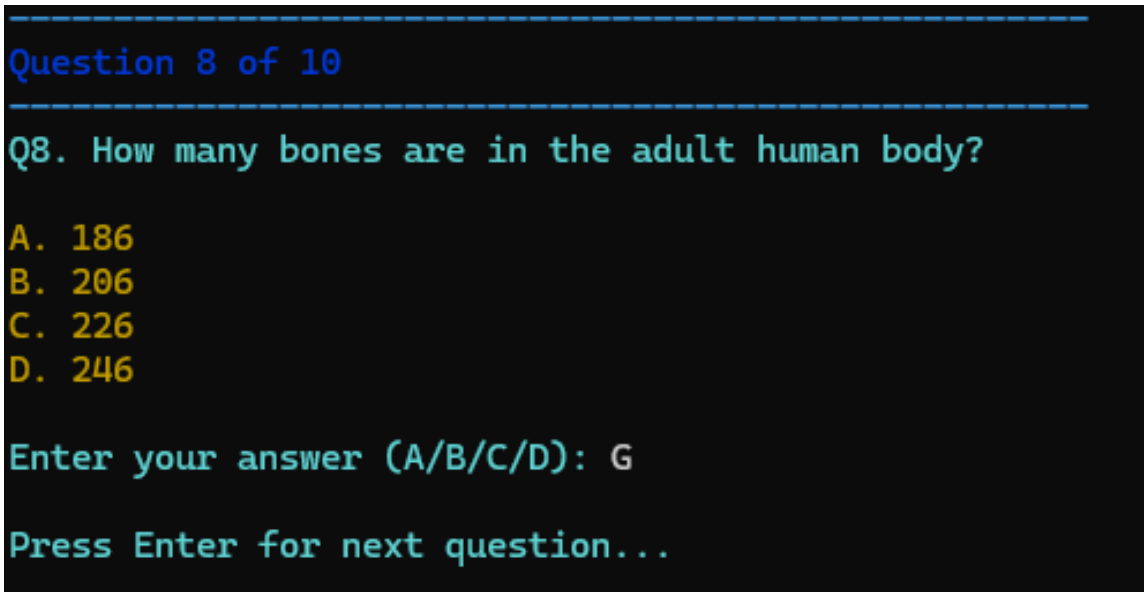
TC05_Question_Display_Hint_Declined

User selects No for hint; no hint shown; question accepts answer directly.

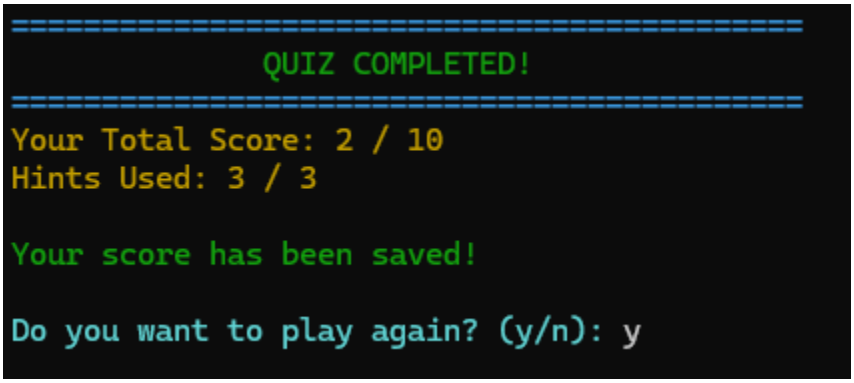
```
------------------------------------------------------
Question 5 of 10
------------------------------------------------------
Q5. What is the currency of Japan?

A. Yuan
B. Yen
C. Won
D. Ringgit

Enter your answer (A/B/C/D): A

Press Enter for next question...
```

TC06_Question_Display_Hint_Limit_Reached

Hint prompt no longer appears because the user already used all 3 hints.



TC07_Question_Display_Invalid_Input_Accepted
User inputs invalid option (e.g., G) and program incorrectly accepts it.



TC08_Result_Summary
Displays final score, number of correct answers, and replay prompt.

CONCLUSION

This project demonstrates how a compact and well-structured C program can provide a practical assessment tool for offline learning. By dividing the system into clear modules, the code remains maintainable, easy to extend, and logically organized. The use of text files for questions and a binary file for the leaderboard provides a lightweight yet effective storage solution, balancing simplicity with performance.

Key Takeaways

- Modular design simplifies testing and future expansion.
- File-based persistence offers a fast and portable alternative to databases.
- Ranking by percentage with a time tiebreaker encourages both accuracy and efficiency.

Future Scope

Possible enhancements include adding a graphical interface, question randomization, or adaptive difficulty levels. The leaderboard could later be migrated to a small database or cloud-based system for advanced analytics. Even so, the current version successfully fulfills its CCP-level objectives, showcasing how core C programming concepts can be applied to create a complete and efficient educational tool.