Istanbul Health and Technology University

SWE208

Computing Systems

In-Class Project

Berat Kahraman
220610038

Introduction

I created a terminal-based quiz application in C++. The goal was to practice key computing-systems concepts—like file input/output, control flow, data representation, and memory management—while building something useful. The program supports three modes:

- **Quiz Mode:** Reads questions from questions.txt, shuffles them, gives the user 30 seconds per question, shows colored feedback, and saves the score to results.txt.
- **Add Question Mode:** Prompts for a username and password, checks them against credentials.txt, and lets the user append a new question if authentication succeeds.
- **View Stats Mode:** Reads past scores from results.txt and displays the number of attempts, average percentage, and best percentage.

File Structure & Responsibilities

- **main.cpp**
  - Presents the three modes menu.
  - In Add Question Mode, reads credentials and calls authenticate() and then addQuestion().
  - In View Stats Mode, calls viewStats().

- In Quiz Mode, reads the user's name, loads questions, shuffles them, and calls runQuiz().

- **fileio.cpp & fileio.h**
  - **loadQuestions()** parses questions.txt line by line, builds Question objects (text, options, correct answer), and returns a vector of questions.
  - **saveResults()** appends a line like Alice – Score: 8 / 10 (80.00%) to results.txt.
  - **addQuestion()** prompts for question text and four options, validates the correct answer, and appends the new question to questions.txt.
  - **viewStats()** reads each score from results.txt, extracts the percentage, and prints total attempts, average percentage, and highest percentage.
  - **authenticate()** reads credentials.txt (lines of user:pass), splits on the colon, and returns true if the entered username and password match any line.
- **quiz.cpp & quiz.h**
  - **runQuiz()** takes the shuffled question list and the user's name. For each question it:
    1. Prints the question and options.
    2. Shows a live countdown on one line using _kbhit()/_getch().
    3. Accepts a single-key answer or times out after 30 seconds.
    4. Prints "Answer: X" plus green "Correct!" or red "Wrong!" (with the right letter).
    5. Tracks and prints the current accuracy percentage after each question.
  - After all questions, it shows the final score and calls saveResults().

Research & Learning

To solve each problem, I used a "just-in-time" learning approach, looking up resources only when needed:

- **Online Documentation**
    - I read cppreference.com to understand how <fstream>, <thread>, <chrono>, <vector>, and ANSI escape sequences work in C++.
    - I consulted the MSYS2 and Windows API docs to set up the MinGW compiler and learn about console functions like _kbhit().
- **YouTube Tutorials**
    - I watched videos on C++ threading basics to learn how to run a countdown loop without blocking input.
    - I found clips explaining ANSI codes for color output to implement green/red feedback.
    - I viewed short C++ security tutorials to get ideas for simple file-based username/password checking.
- **StackOverflow & Q&A**
    - I searched for "non-blocking console input C++" and compared different methods, eventually choosing _kbhit() over std::async for reliable behavior on Windows.
    - I used Q&A examples to refine my file-parsing logic and error handling in loadQuestions().

Challenges & Solutions

- **Include Path / IntelliSense Errors:** At first VS Code couldn't find standard headers. I switched to MSYS2's MinGW toolchain, updated c_cpp_properties.json, and reset IntelliSense.

- **Timer vs. Input Conflict:** My first async-based timer blocked further questions until hitting Enter repeatedly. I rewrote the loop to use _kbhit() and _getch(), which let me update the countdown on one line and accept a single keypress.
- **Linker Undefined Reference:** When I added authenticate(), I forgot its definition signature. Adding the correct three-parameter definition in fileio.cpp fixed the linker error.

Process Management

- I organized my work in small steps. First I made a simple “Hello, Quiz!” program, then I added file loading, next the shuffle feature, then the timer, authentication, and finally the statistics display.
- After each new feature, I compiled and ran the code immediately to make sure it worked before moving on.
- I kept a plain-text log of every error I saw (missing headers, linker messages, timer conflicts) and the exact change that fixed it.
- I set a goal for each coding session (“today I’ll parse questions,” “tomorrow I’ll add the countdown”), which helped me stay focused and made steady progress.
- I used ChatGPT when setting up the initial project structure and later to troubleshoot and resolve compilation and IntelliSense errors.

Conclusion

By researching targeted resources, experimenting with alternative approaches, and managing my work in small, testable increments, I built a quiz app that clearly demonstrates file I/O, control flow, data representation, and memory safety. This process of focused learning and iterative development reflects exactly the effort and problem-solving approach this course values.