Coding assessment for **Embedded Engineer** role:

Implementing a Circular Buffer with Random Heart Rate Generator and Exponential Moving Average (EMA)

Objective: The goal of this assessment is to evaluate your ability to implement fundamental data structures in C, manage memory effectively, generate and handle data streams, and use version control systems. You will be tasked with implementing a circular buffer, writing a custom Makefile for compilation, and documenting your work with a comprehensive README. Additionally, you will be required to use Git for version control and submit your project through a local Git repository.

Problem Description

A circular buffer (also known as a ring buffer) is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end. This structure is useful for buffering data streams, such as data read from hardware interfaces or network sockets.

Your task is to implement a circular buffer in C with the following functionalities:

1. Initialisation: Create a function to initialise the circular buffer.
2. Add Element: Create a function to add an element to the buffer.
3. Remove Element: Create a function to remove an element from the buffer.
4. Check Full: Create a function to check if the buffer is full.
5. Check Empty: Create a function to check if the buffer is empty.

Additionally, you will implement a random heart rate generator that creates a new random heart rate every second and stores it in the circular buffer. The size of the circular buffer will be provided as a command line argument. Every time a new heart rate is generated, store it in the buffer, update the buffer, and print the Exponential Moving Average (EMA) calculated from all heart rate values in the buffer.

You can find more information about EMA in https://en.wikipedia.org/wiki/Exponential_smoothing

---

Requirements

1. **Functional requirements:**
   - Implement circular buffer with a fixed size (size can be coming from the command line)
   - Implement a random heart rate generator (values can be from 44 to 185)
   - Calculate and show EMA for the stored heart rate values

2. **Makefile:**
   - Write a custom Makefile to compile your code in different operating systems such as Mac, linux, and windows.
   - Ensure the Makefile has the following targets:
     - **`all`**: Compile the code.:
     - **`clean`**: Remove all compiled objects and binaries.

3. **Unit test:**
   - Write a simple unit test preferably with G-test for your code, and ensure all functions work correctly.

4. **README**:
   - Provide a detailed README file that includes:

    - An overview of the project.
    - Instructions on how to compile and run the code and unit test.
    - Examples of how to use the circular buffer functions.
    - Any assumptions or design decisions you made.

4. **Git Repository:**
    - Initialize a local Git repository.
    - Make regular commits that document your progress.
    - Your final submission should include the `.git` directory to show your commit history.

Deliverables

1. Source files containing your implementation.
2. `Makefile`: Custom Makefile to compile your code.
3. `README.md`: Detailed documentation for your project.
4. Unit test: Custom unit test to test all of the code functions.
5. Local Git repository with commit history.

Submission

1. Initialize a local Git repository in your project directory.
2. Make regular commits with meaningful messages as you progress.
3. Ensure your final commit includes all required files and the `.git` directory.
4. Create a compressed archive (`.zip` or `.tar.gz`) of your project directory, including the `.git` directory.
5. Submit the compressed archive as your final deliverable.

Assessment Criteria

1. **Correctness**: Your implementation correctly performs all required operations on the circular buffer.
2. **Code Quality**: Your code is well-structured, readable, and follows good coding practices.
3. **Makefile**: Your Makefile correctly compiles the code and handles the `all` and `clean` targets.
4. **Unit test:** Your unit test is well-structured, readable, and be able to test all the code functions.
5. **Documentation**: Your README is clear, comprehensive, and provides all necessary information to understand and use your implementation.
6. **Version Control**: Your Git commit history shows regular, meaningful commits that document your development process.

Good luck, and happy coding!