Project title : Developing a smart Alarm clock system

1. Project Requirements Gathering

- **Features**: Define the core features for the alarm clock:
  - Setting the time
  - Setting multiple alarms with customizable times
  - Snooze functionality
  - Displaying the current time
  - Optional: Alarm sound or notification
- **Tools**: Decide which libraries and tools are necessary:
  - **Standard C++ Libraries**: `<iostream>`, `<iomanip>`, `<ctime>`, `<thread>`, etc.
  - **Optional**: For sound functionality, consider cross-platform libraries like **SDL** or **SFML** for sound management.

2. Initial Setup

1. **Development Environment**
   a. **IDE/Editor**: Visual Studio, CLion, Code::Blocks, or a text editor with `g++`/`clang++`.
   b. **Compiler**: Use a C++ compiler that supports at least C++11, as threading is a critical feature.
2. **Project Structure**
   a. **Root Directory**: Create a directory named `AlarmClock`.
   b. **Source Folder**: Inside `AlarmClock`, create folders:
      i. `src`: Holds source files.
      ii. `include`: Holds header files.
      iii. `lib` (optional): For external libraries if needed.
   c. **Files**:
      i. `main.cpp`: Entry point.
      ii. `AlarmClock.h` and `AlarmClock.cpp`: Core alarm clock functionality.
      iii. Additional files (e.g., `SoundManager.h`, `SoundManager.cpp`) if sound functionality is implemented.

3. Design and Development

a. Class Design

Define classes and methods based on requirements. Below are key classes:

- **AlarmClock Class**
    - **Attributes**:
        - `current_time`: Keeps track of the system time.
        - `alarms`: Stores the list of alarms.
        - `snooze_time`: Defines snooze interval.
    - **Methods**:
        - `void setAlarm(int hour, int minute)`: Sets an alarm.
        - `void checkAlarms()`: Checks if any alarm should go off.
        - `void snooze(int minutes)`: Sets snooze time for an alarm.
        - `void updateCurrentTime()`: Updates the current time using `std::chrono`.
- **Optional SoundManager Class**
    - **Methods**:
        - `void playSound()`: Plays an alarm sound.
        - `void stopSound()`: Stops the sound when the alarm is dismissed or snoozed.

b. Implement Core Functionality

- **Time Display and Update**:
    - Use `std::chrono` to manage and update time.
    - `std::this_thread::sleep_for()` to introduce pauses for checking alarms.
- **Alarm Management**:
    - Store alarms in a data structure, e.g., a `std::vector`.
    - Implement logic to trigger alarms at specified times.
- **Snooze Feature**:
    - Calculate snooze intervals and add them to the current time of the alarm.

4. Testing

1. **Unit Tests**:

    a. Test the time update mechanism.
    b. Test alarm setting and triggering.
    c. Test snooze functionality.
2. **Integration Tests**:
    a. Run end-to-end tests to ensure alarms trigger correctly.
    b. Verify correct functionality when multiple alarms are set.
3. **User Testing** (Optional):
    a. If sound or other interactive features are added, test on different platforms to confirm compatibility.

5. Code Documentation and Comments

- Comment code blocks and use `Doxygen`-style comments for functions and classes if the project will be expanded or handed off.

6. Compilation and Execution

- **Makefile (Optional)**: Create a `Makefile` to automate compilation.
- **Build**: Compile with `g++ main.cpp src/*.cpp -o alarm_clock`.
- **Execution**: Run the binary and validate functionality.

7. Further Enhancements (Optional)

- **User Interface**: Implement a simple text-based UI, or if ambitious, a GUI using libraries like Qt.
- **Sound Improvements**: Allow users to select different alarm tones.
- **Persistence**: Save and load alarm settings to/from a file.