Write the Pseudocode and Flowchart for the problem statements mentioned below:

---

1. Smart Home Temperature Control

Problem Statement:
Design a temperature control system for a smart home. The system should read the current temperature from a sensor every minute and compare it to a user-defined setpoint.

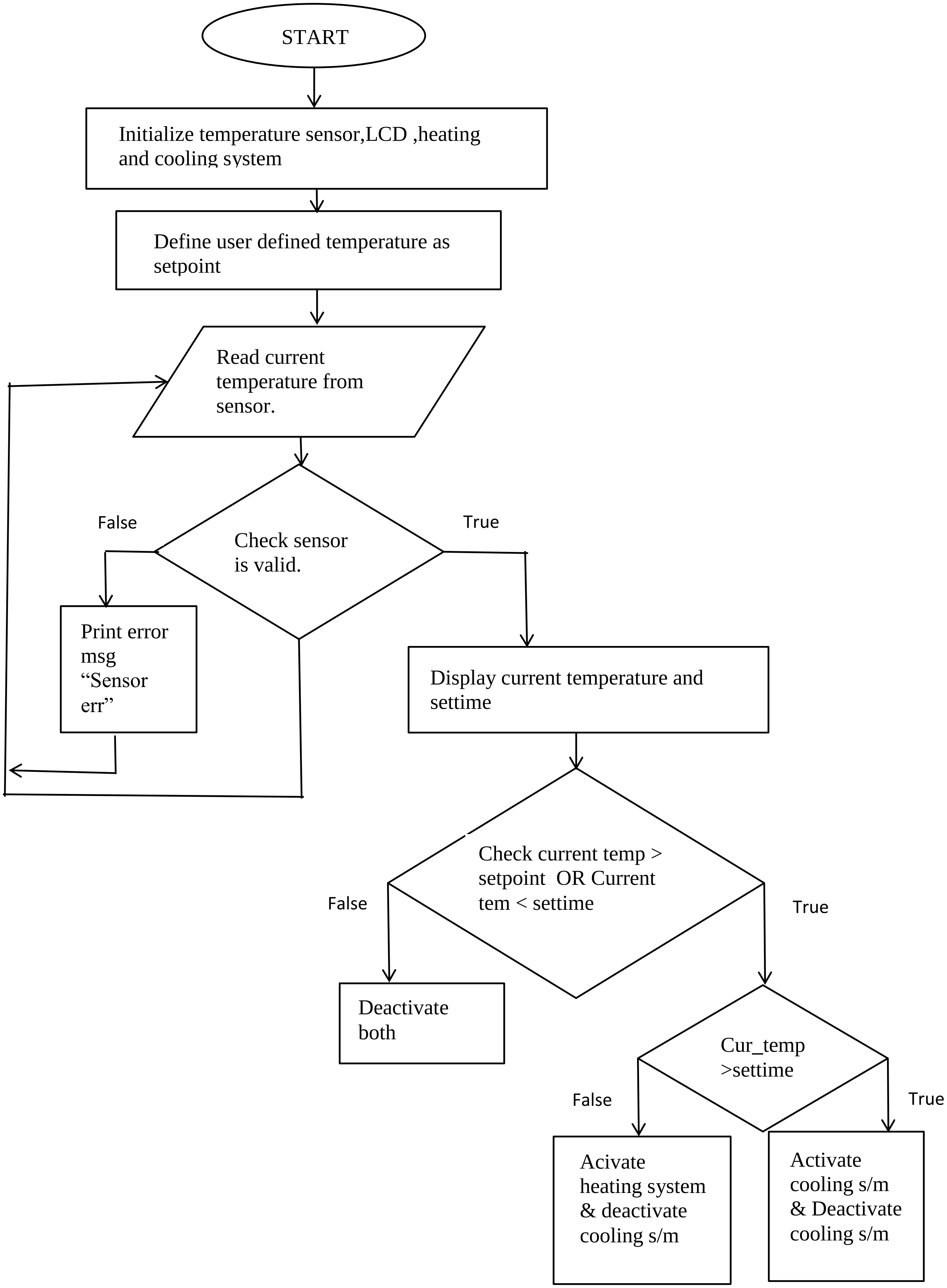
Requirements:

- If the current temperature is above the setpoint, activate the cooling system.
- If the current temperature is below the setpoint, activate the heating system.
- Display the current temperature and setpoint on an LCD screen.
- Include error handling for sensor failures.

Pseudocode:

```
1)      Start
2)      Initialize the system by initializing the temperature sensor, LCD display, heating and
        cooling system.
3)      Run a loop which repeat in every minute
        a.  Define a user defined temperature as setpoint.
        b. Read the current temperature from the sensor
        c.  Check the sensor reading is valid
              i. If the sensor failed print an error message on LCD showing "Sensor error".
              ii. Skip to next iteration if sensor error happened
        d.  Display the current temperature and setpoint on an LCD screen.
        e.  Compare current temperature with setpoint:
                i.   If  current temperature  >  setpoint
                     * Activate cooling system and deactivate heating system
                ii.  Else if  current temperature  <  setpoint
                     * Activate heating system and deactivate cooling system
                iii. Else
                     * Deactivate both cooling and  heating system as it reached ideal temperature
4)      Wait for one minute
```

Flowchart:


START
Initialize temperature sensor,LCD ,heating and cooling system
Define user defined temperature as setpoint
Read current temperature from sensor.
False
True
Check sensor is valid.
Print error msg "Sensor err"
Display current temperature and settime
Check current temp > setpoint OR Current tem < settime
False
True
Deactivate both
Cur_temp >settime
False
True
Acivate heating system & deactivate cooling s/m
Activate cooling s/m & Deactivate cooling s/m

2. Automated Plant Watering System

Problem Statement:

Create an automated watering system for plants that checks soil moisture levels and waters the plants accordingly.

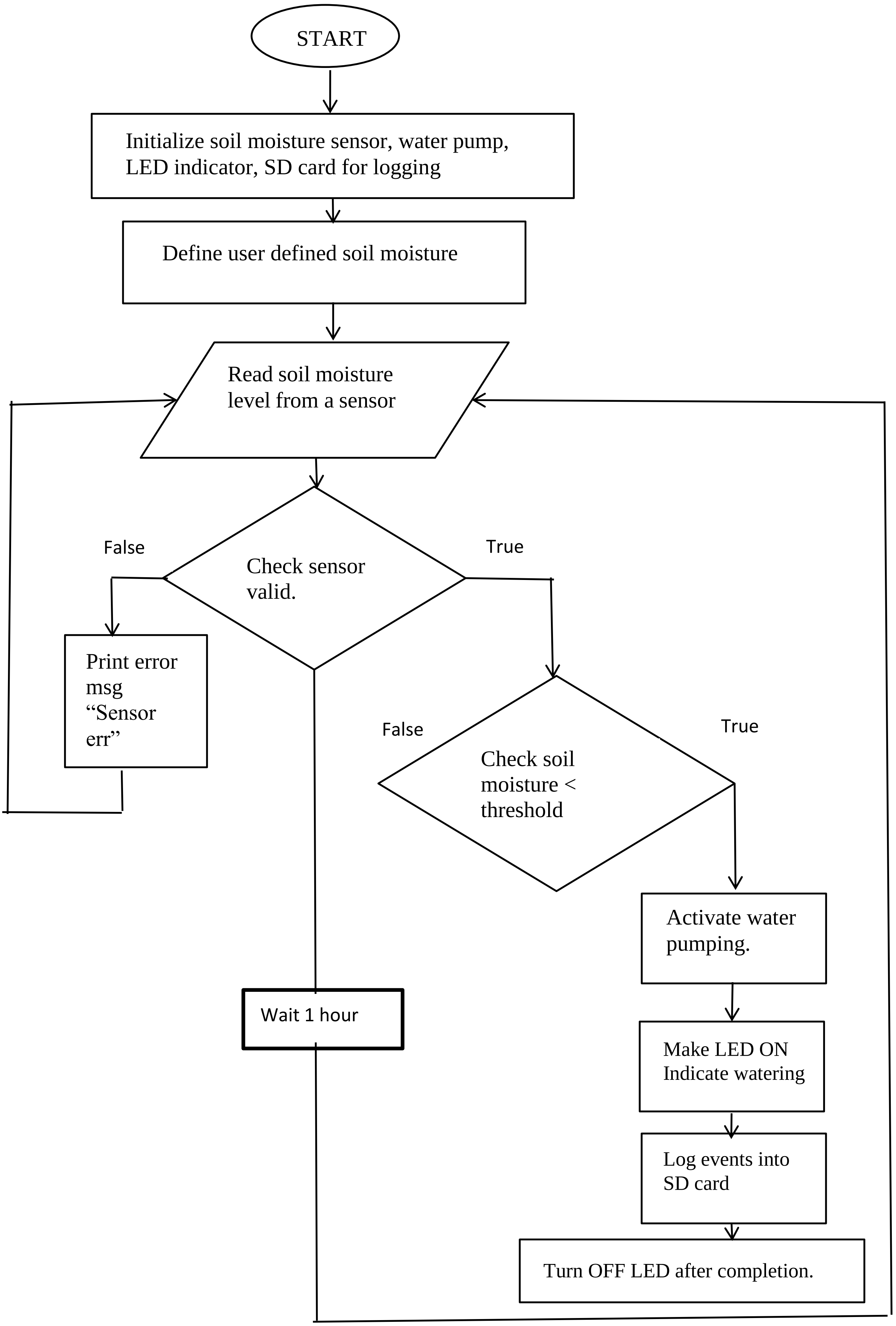
Requirements:

- Read soil moisture level from a sensor every hour.
- If moisture level is below a defined threshold, activate the water pump for a specified duration.
- Log the watering events with timestamps to an SD card.
- Provide feedback through an LED indicator (e.g., LED ON when watering).

Pseudocode:

```
1) Start
2)  Initialize soil moisture sensor, water pump, LED indicator, SD card for logging
3) Define user defined soil moisture threshold
4) Run a loop for every each hour
        a. Read soil moisture level from a sensor every hour.
        b. Check if sensor reading is valid
            i. If the sensor fails, log "Sensor Error" to the SD card with a timestamp
             ii. Skip to next iteration if sensor error
        c. Check whether the moisture level < threshold
            i. If yes ,then activate watering pump for specified duration
            ii. Make the LED ON Indicating watering
            iii. Log the watering events with timestamps to an SD card.
            iv. Turn OFF  LED after watering completed.
        d. If moisture level  >  threshold , then indicates adequate moisture .So do nothing.
5) Wait for 1 hour.
```

Flowchart:


START
Initialize soil moisture sensor, water pump, LED indicator, SD card for logging
Define user defined soil moisture
Read soil moisture level from a sensor
False
True
Check sensor valid.
Print error msg "Sensor err"
False
True
Check soil moisture < threshold
Activate water pumping.
Wait 1 hour
Make LED ON Indicate watering
Log events into SD card
Turn OFF LED after completion.

3. Motion Detection Alarm System

Problem Statement:
Develop a security alarm system that detects motion using a PIR sensor.

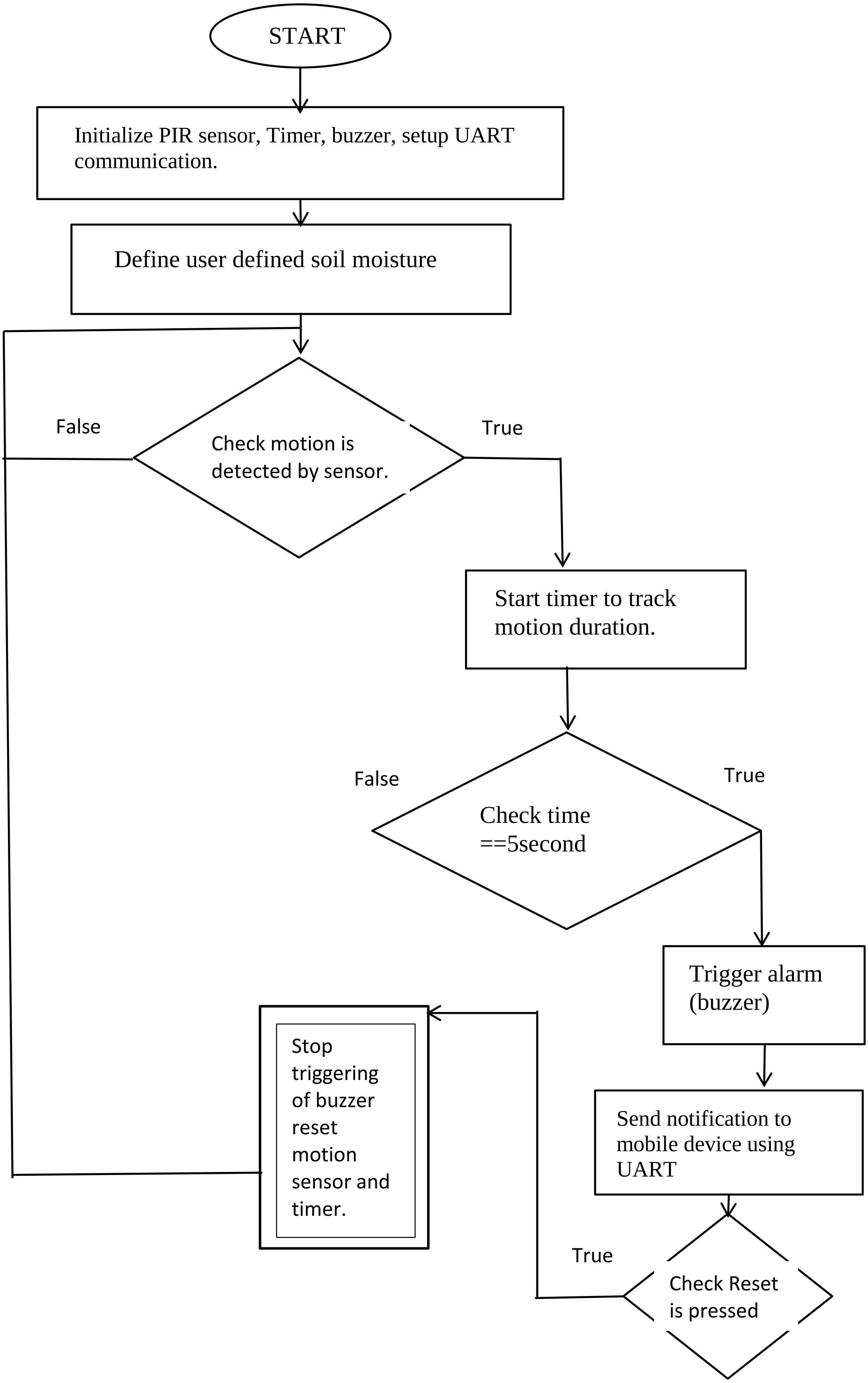
Requirements:

- Continuously monitor motion detection status.
- If motion is detected for more than 5 seconds, trigger an alarm (buzzer).
- Send a notification to a mobile device via UART communication.
- Include a reset mechanism to deactivate the alarm.

Pseudocode:

```
1) Start
2) Initialize PIR sensor, Timer, buzzer, setup UART communication.
3) Run a loop for continuously monitoring the motion detection status.
    a. Check the motion detection using a PIR sensor.
    b. If motion is detected:
        i.  Start a timer to track the duration of motion detected.
        ii. Check if the motion detected for 5 seconds:
            * Trigger an alarm (Buzzer).
            *  Send a notification to a mobile device using UART communication.
            *  wait for the reset command.
        iii.  If reset button is pressed:
            * Stop triggering of buzzer
            * Reset motion detection status and timer.
            * Resume from step 3
    c. If motion not detected , then continue monitoring.
```

Flowchart:


START
Initialize PIR sensor, Timer, buzzer, setup UART communication.
Define user defined soil moisture
False
Check motion is detected by sensor.
True
Start timer to track motion duration.
False
Check time ==5second
True
Trigger alarm (buzzer)
Send notification to mobile device using UART
Stop triggering of buzzer reset motion sensor and timer.
True
Check Reset is pressed

4. Heart Rate Monitor

Problem Statement:

Implement a heart rate monitoring application that reads data from a heart rate sensor.

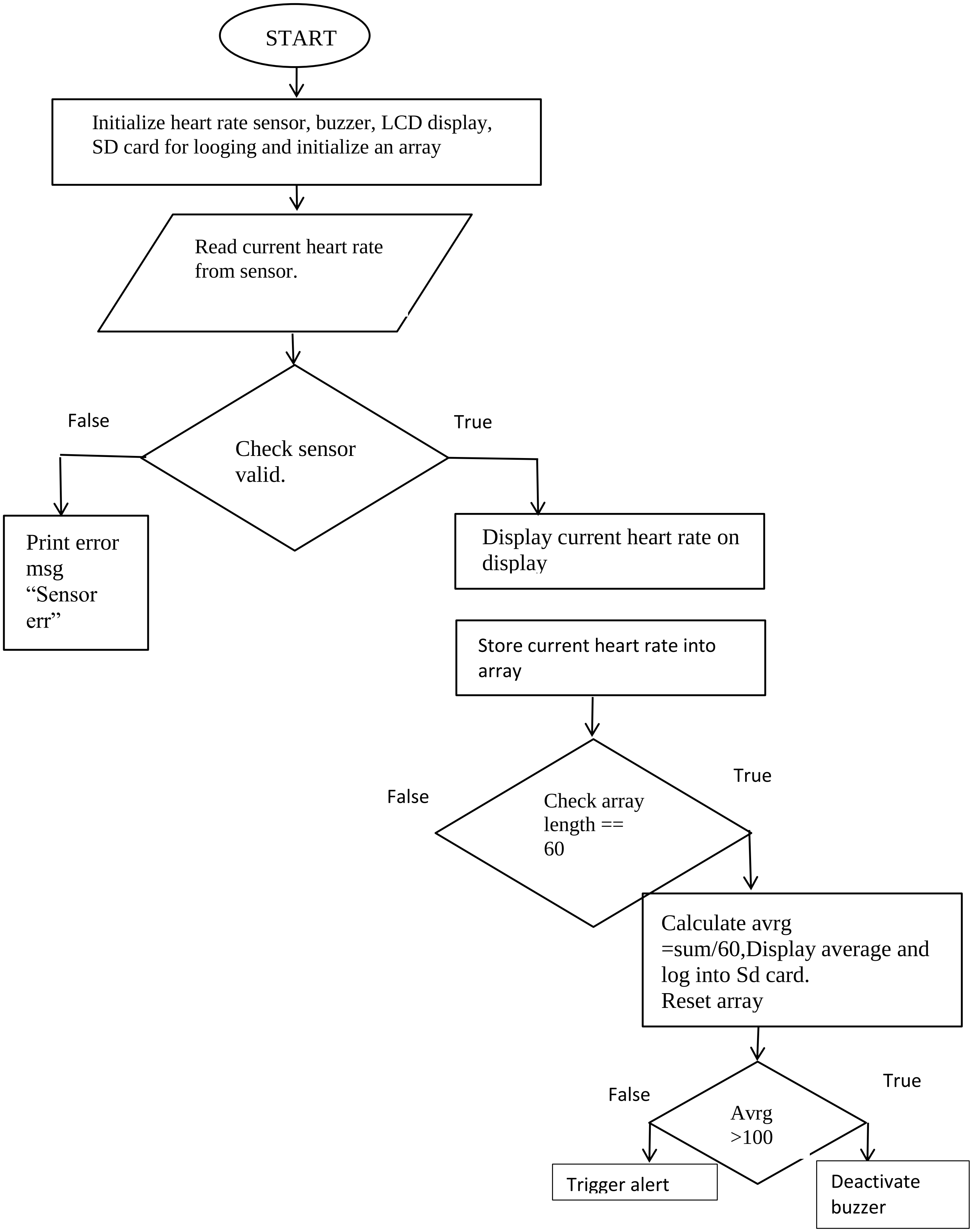
Requirements:

- Sample heart rate data every second and calculate the average heart rate over one minute.
- If the heart rate exceeds 100 beats per minute, trigger an alert (buzzer).
- Display current heart rate and average heart rate on an LCD screen.
- Log heart rate data to an SD card for later analysis.

Pseudocode:

```
1) Start
2) Initialize heart rate sensor, buzzer, LCD display, SD card for looging and initialize an array of
   size to store  heart rate in each second over one minute.
3) Run a loop in every second:
   a. Read current heart rate from sensor.
   b. Validate sensor:
      i. If the sensor reading is invalid:
       - Log "Sensor Error" with timestamp to the SD card
       - Skip to the next iteration
   c. Display current heart rate on display
   d. Store the current heart rate into the array
   e. Check 60 samples have been collected:
       i. calculate average (sum of 60 samples / 60)
       ii.Display average heart rate on LCD
       iii.Log the average  heart rate with time stamp into the SD card
       iv.Reset array for next minute
   f. Check if the average heart rate > 100
       i. If yes ,then trigger the alert
       ii. Else then deactivate buzzer
   g. Wait for one minute
```

Flowchart:


START
Initialize heart rate sensor, buzzer, LCD display, SD card for looging and initialize an array
Read current heart rate from sensor.
False
True
Check sensor valid.
Print error msg "Sensor err"
Display current heart rate on display
Store current heart rate into array
False
True
Check array length == 60
Calculate avrg =sum/60,Display average and log into Sd card.
Reset array
False
True
Avrg >100
Trigger alert
Deactivate buzzer

5. LED Control Based on Light Sensor

Problem Statement:
Create an embedded application that controls an LED based on ambient light levels detected by a light sensor.

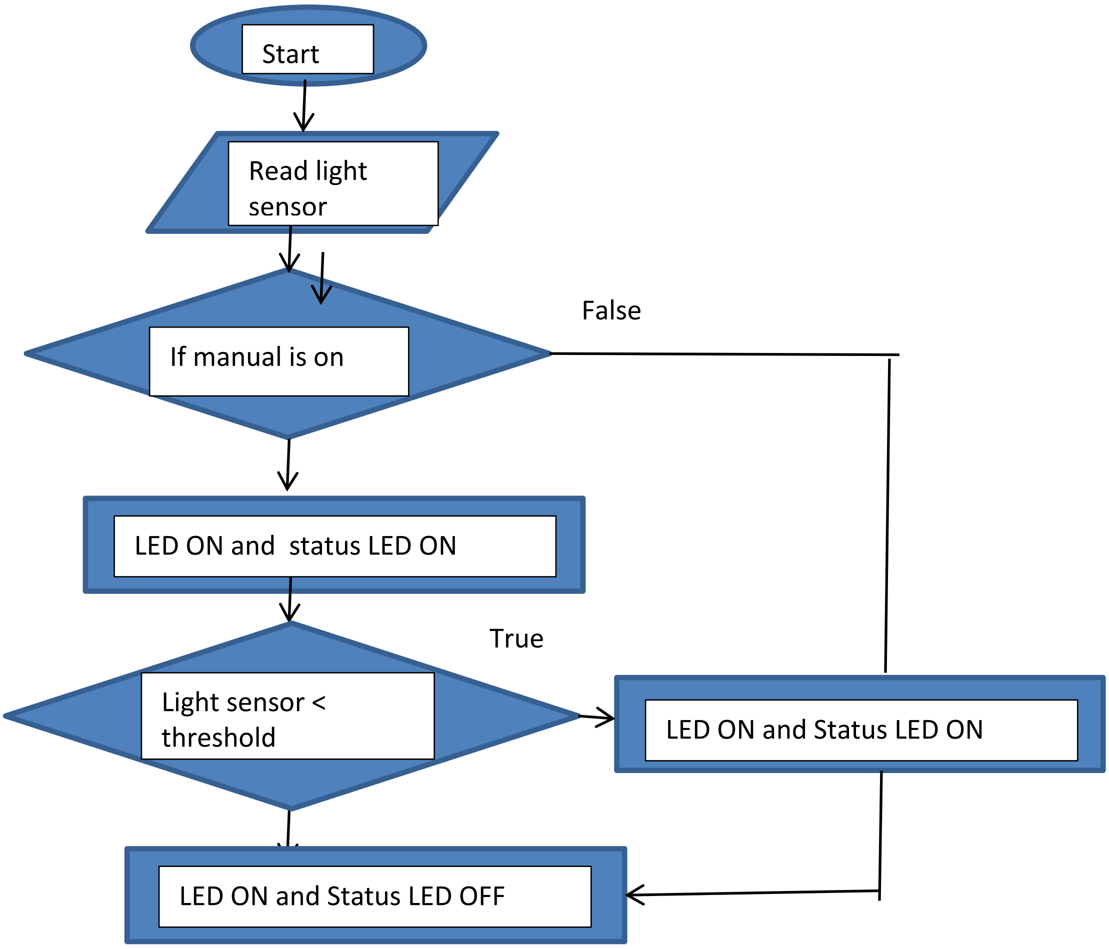
Requirements:

- Read light intensity from the sensor every minute.
- If light intensity is below a certain threshold, turn ON the LED; otherwise, turn it OFF.
- Include a manual override switch that allows users to control the LED regardless of sensor input.
- Provide status feedback through another LED (e.g., blinking when in manual mode).

Pseudocode:

```
1) Start
2) Initialize light sensor, LED controlled by sensor, Led for manual mode indication, manual
   override switch.
3) Define an user define light intensity as threshold
4) Run a loop for every minute
       a. Check the status of manual override switch:
              i. If manual override switch is active:
                     -Turn on the status LED indicating manual mode.
                     -Control the main LED manually as per user need
                     -Continue to next instruction
              ii. If manual override switch is inactive , turn off the LED indicating manual mode.
       b. Read the light intensity from sensor.
       c. Check the intensity < threshold
            i. If yes , then Turn ON main LED
            ii. Else Turn OFF main LED
  5)Wait for 1 minute
```

Flowchart:



Start
Read light sensor
If manual is on
False
LED ON and status LED ON
True
Light sensor < threshold
LED ON and Status LED ON
LED ON and Status LED OFF

6. Digital Stopwatch

Problem Statement:
Design a digital stopwatch application that can start, stop, and reset using button inputs.

Requirements:

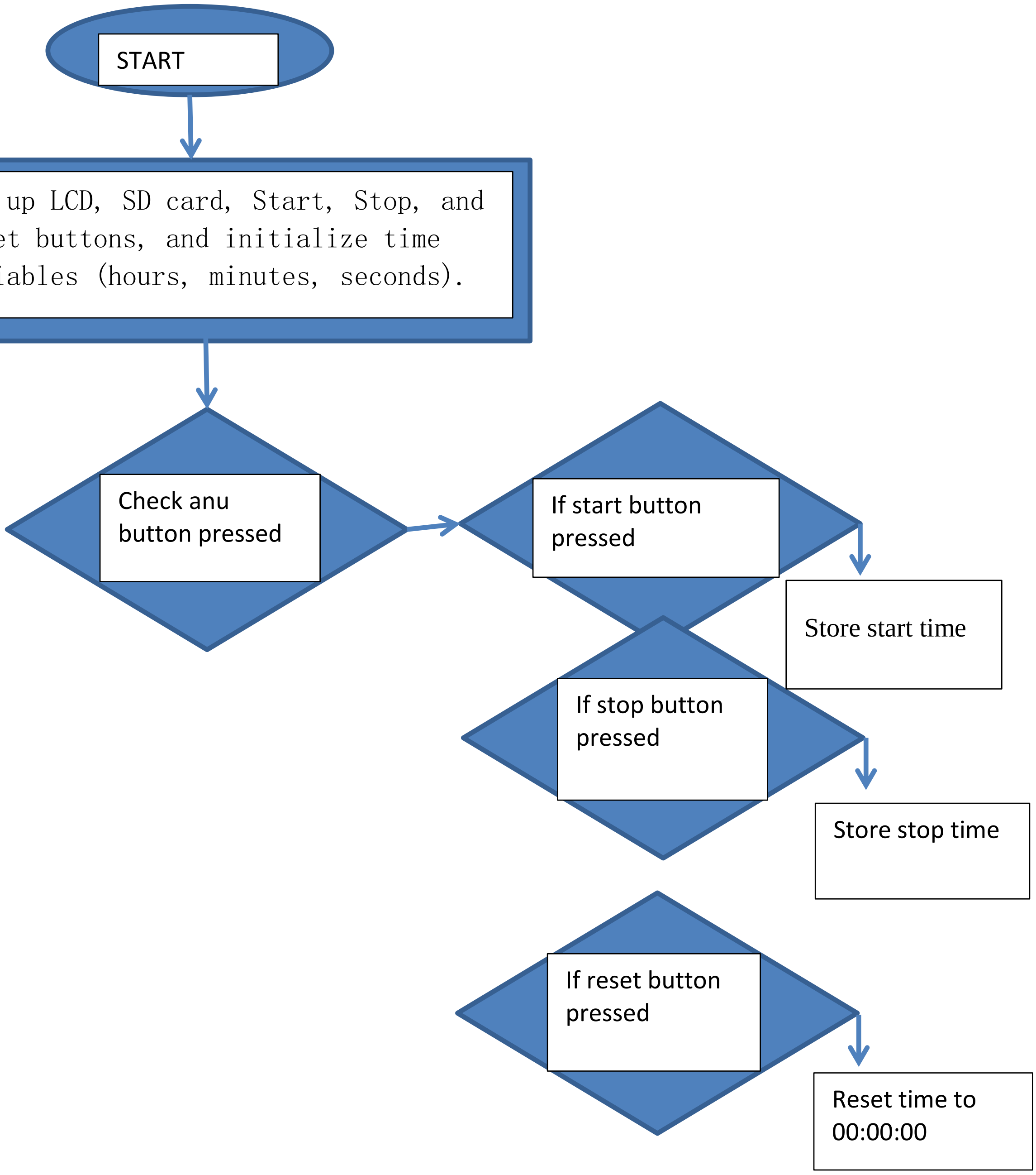
- Use buttons for Start, Stop, and Reset functionalities.
- Display elapsed time on an LCD screen in hours, minutes, and seconds format.
- Include functionality to pause and resume timing without resetting.
- Log start and stop times to an SD card when stopped.

Pseudocode:

```
1) Start
2) * Initialize buttons for start, stop and reset.
   * Initialize LCD display
   * Initialize SD card for logging time.
   * Initialize variables for hours , minute ,second stopwatch_running
   * Set the value of stopwatch_running as false
3) Run a loop:
    a) Check the button inputs:
       i. If start button is pressed:
          *Check if stopwatch_running is false:
              - Set stopwatch_running true
              - Log the start time into the SD card
       ii.If stop button is pressed:
          *Check if stopwatch_running is  true:
              -Set stopwatch_running false
              - Log the stop time into the SD card
       iii.If Reset button is pressed:
       - Set hours, minutes, and seconds to 0
       - Display "00:00:00" on LCD
       - Set stopwatch_running to False
    b) If stopwatch_running is True:
    - Increment seconds
    - If seconds == 60:
        - Set seconds to 0
```

```
     - Increment minutes
  - If minutes == 60:
    - Set minutes to 0
     - Increment hours
    - Display elapsed time (hours:minutes:seconds) on the LCD
        d. Wait for 1 second
```

Flowchart:



7. Temperature Logging System

Problem Statement:

Implement a temperature logging system that records temperature data at regular intervals.

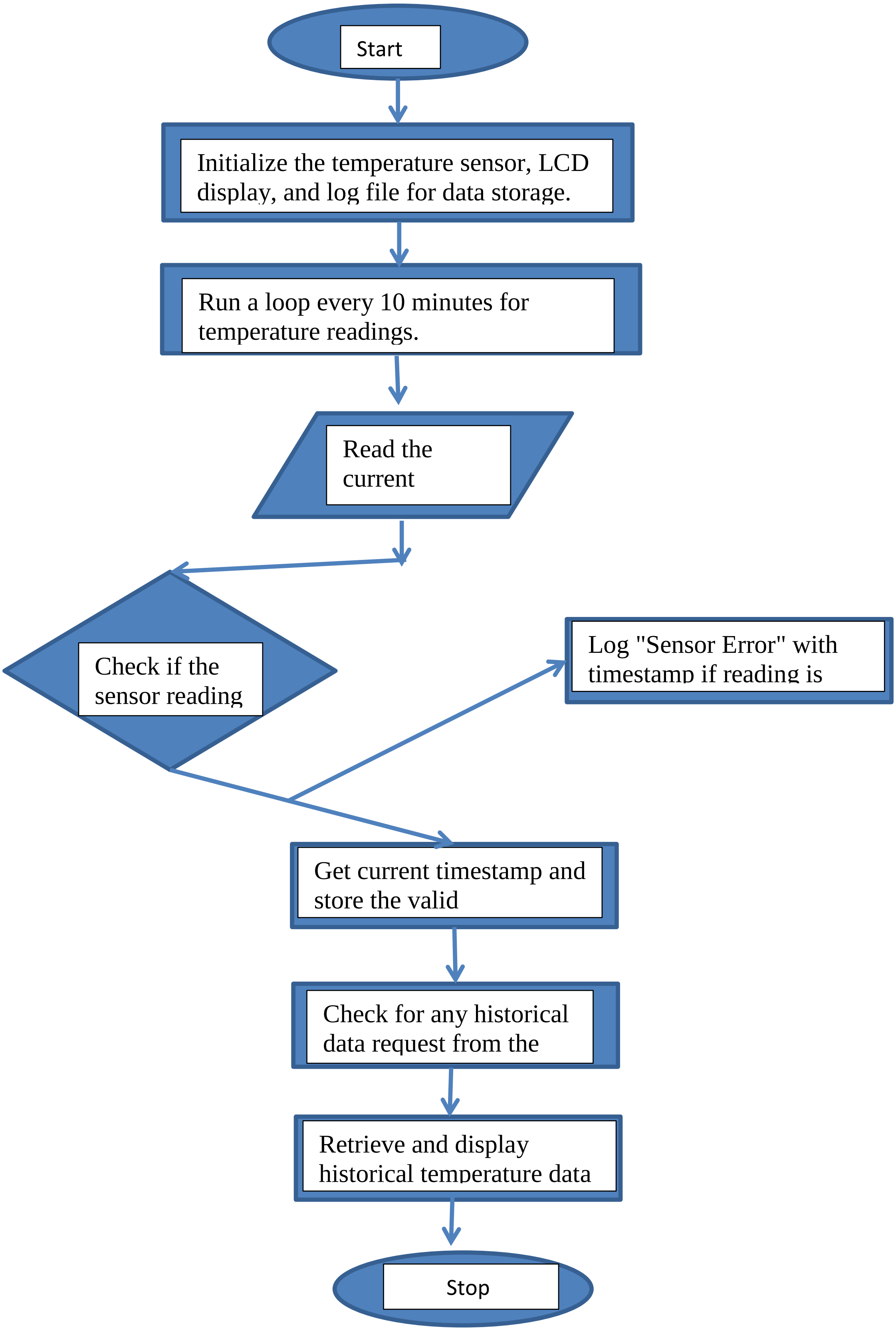
Requirements:

- Read temperature from a sensor every 10 minutes.
- Store each reading along with its timestamp in an array or log file.
- Provide functionality to retrieve and display historical data upon request.
- Include error handling for sensor read failures.

Pseudocode:

```
1. Start
2. * Initialize temperature sensor, LCD display
   * Initialize array or log file for storing temperature data with timestamps
   * Set data collection interval to 10 minutes
2. Run a loop for each 10 minute:
  a. Read current temperature from the sensor
  b. Check if sensor reading is valid
    i. If the sensor reading is invalid:
      - Log "Sensor Error" with timestamp in the log file or array
      - Skip to the next iteration
  c. If the reading is valid:
    - Get current timestamp
    - Store the temperature reading and timestamp in the log file or array
  d. Check for historical data request
    i. If a request is received:
      - Retrieve and display historical temperature data from the log file or array
  e. Wait for 10 minutes before the next reading
```

Flowchart:


Start
Initialize the temperature sensor, LCD display, and log file for data storage.
Run a loop every 10 minutes for temperature readings.
Read the current
Check if the sensor reading
Log "Sensor Error" with timestamp if reading is
Get current timestamp and store the valid
Check for any historical data request from the
Retrieve and display historical temperature data
Stop

8. Bluetooth Controlled Robot

Problem Statement:
Create an embedded application for controlling a robot via Bluetooth commands.

Requirements:

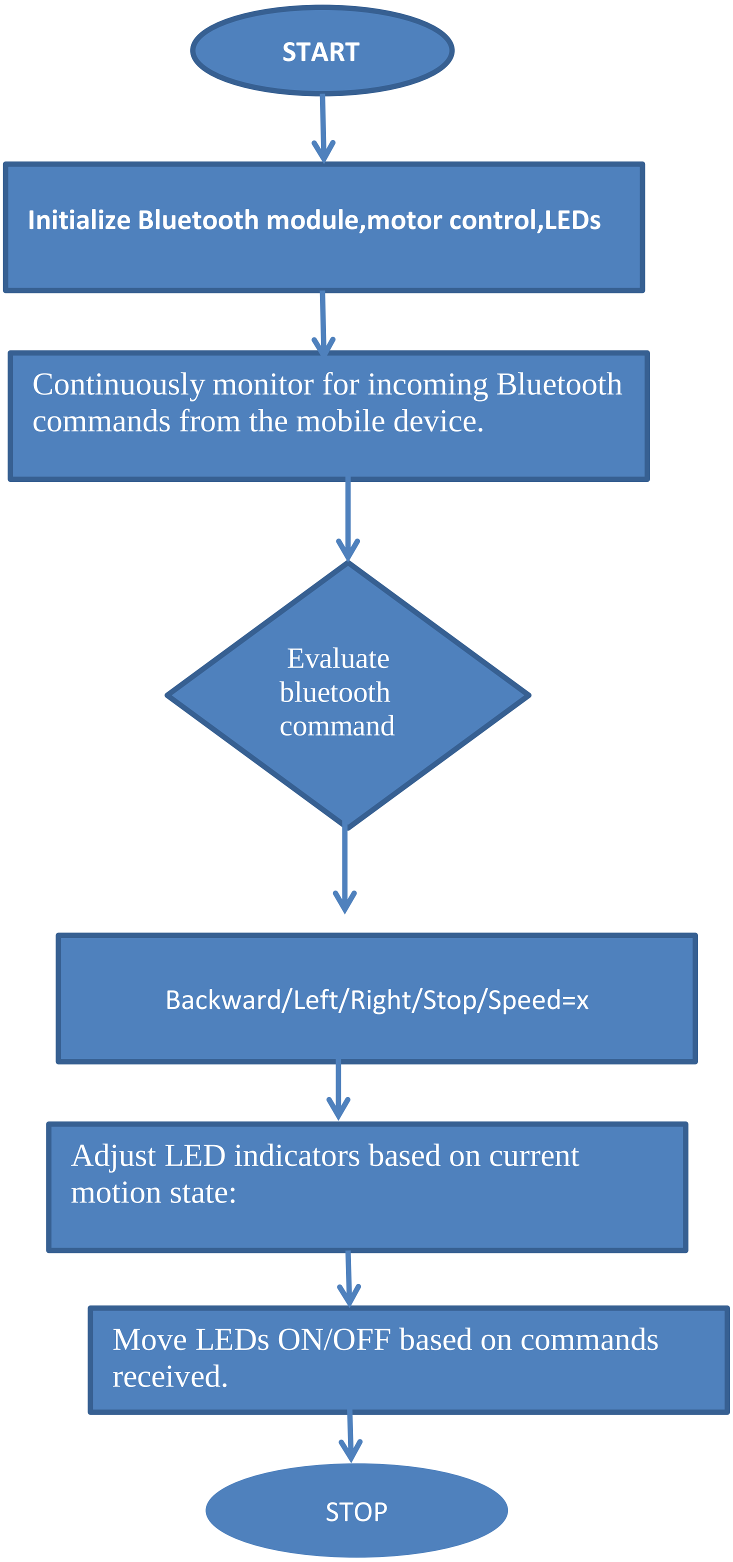
- Establish Bluetooth communication with a mobile device.
- Implement commands for moving forward, backward, left, and right.
- Include speed control functionality based on received commands.
- Provide feedback through LEDs indicating the current state (e.g., moving or stopped).

Pseudocode:

```
    1. Start
    2. a.Initialize Bluetooth module for communication
       b. Initialize motor control (forward, backward, left, right)
       c. Initialize LEDs for state feedback (e.g., moving or stopped)
       d. Set default speed
2. Main Loop (Continuous Monitoring for Bluetooth Commands)
   a. Check for incoming Bluetooth command from the mobile device
   b. If a command is received:
      i. Parse the command to determine action
      ii. Execute command based on action:
         - If command = "FORWARD":
            - Set motors to move forward
            - Turn ON "Moving" LED
         - If command = "BACKWARD":
            - Set motors to move backward
            - Turn ON "Moving" LED
         - If command = "LEFT":
            - Set motors to turn left
            - Turn ON "Moving" LED
         - If command = "RIGHT":
            - Set motors to turn right
            - Turn ON "Moving" LED
         - If command = "STOP":
            - Stop all motors
            - Turn OFF "Moving" LED
```

```
        - If command includes "SPEED=X":
          - Set robot speed to X
  c. Update LED state based on motion:
    - If any movement command (FORWARD, BACKWARD, LEFT, RIGHT) is active:
      - Ensure "Moving" LED is ON
    - If STOP command is active:
      - Ensure "Moving" LED is OFF
```

Flowchart:


START
Initialize Bluetooth module,motor control,LEDs
Continuously monitor for incoming Bluetooth commands from the mobile device.
Evaluate bluetooth command
Backward/Left/Right/Stop/Speed=x
Adjust LED indicators based on current motion state:
Move LEDs ON/OFF based on commands received.
STOP

9. Battery Monitoring System

Problem Statement:
Develop a battery monitoring system that checks battery voltage levels periodically and alerts if voltage drops below a safe threshold.

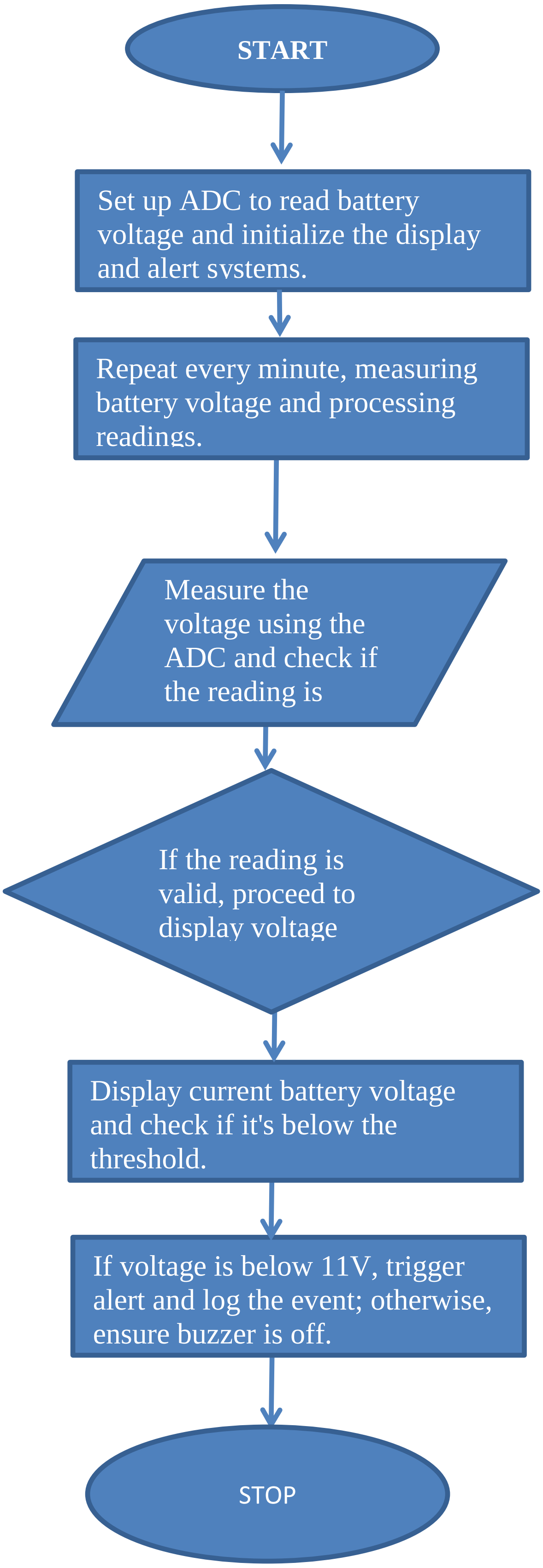
Requirements:

- Measure battery voltage every minute using an ADC (Analog-to-Digital Converter).
- If voltage falls below 11V, trigger an alert (buzzer) and log the event to memory.
- Display current voltage on an LCD screen continuously.
- Implement power-saving features to reduce energy consumption during idle periods.

Pseudocode:

```
    1. Start
    2. a. Initialize ADC to read battery voltage
       b. Initialize LCD display for showing voltage
       c. Initialize buzzer for low-voltage alerts
       d. Initialize memory or log file for storing low-voltage events
       e. Set safe voltage threshold to 11V
2. Main Loop (Repeat every minute)
   a. Measure battery voltage using ADC
   b. Check if the reading is valid
      i. If reading is invalid:
         - Skip to the next iteration
   c. Display the current battery voltage on the LCD
   d. Check if voltage < 11V (low voltage condition)
      i. If voltage < 11V:
         - Trigger alert (activate buzzer)
         - Log the event with timestamp to memory or log file
      ii. If voltage >= 11V:
         - Ensure the buzzer is deactivated
   e. Enter power-saving mode for idle period until next reading
      - Reduce power consumption by disabling unnecessary modules
   f. Wait for 1 minute
```

Flowchart:


START
Set up ADC to read battery voltage and initialize the display and alert systems.
Repeat every minute, measuring battery voltage and processing readings.
Measure the voltage using the ADC and check if the reading is
If the reading is valid, proceed to display voltage
Display current battery voltage and check if it's below the threshold.
If voltage is below 11V, trigger alert and log the event; otherwise, ensure buzzer is off.
STOP

10. RFID-Based Access Control System

Problem Statement:

Design an access control system using RFID technology to grant or deny access based on scanned RFID tags.

Requirements:

- Continuously monitor for RFID tag scans using an RFID reader.
- Compare scanned tags against an authorized list stored in memory.
- Grant access by activating a relay if the tag is authorized; otherwise, deny access with an alert (buzzer).
- Log access attempts (successful and unsuccessful) with timestamps to an SD card.

Pseudocode:

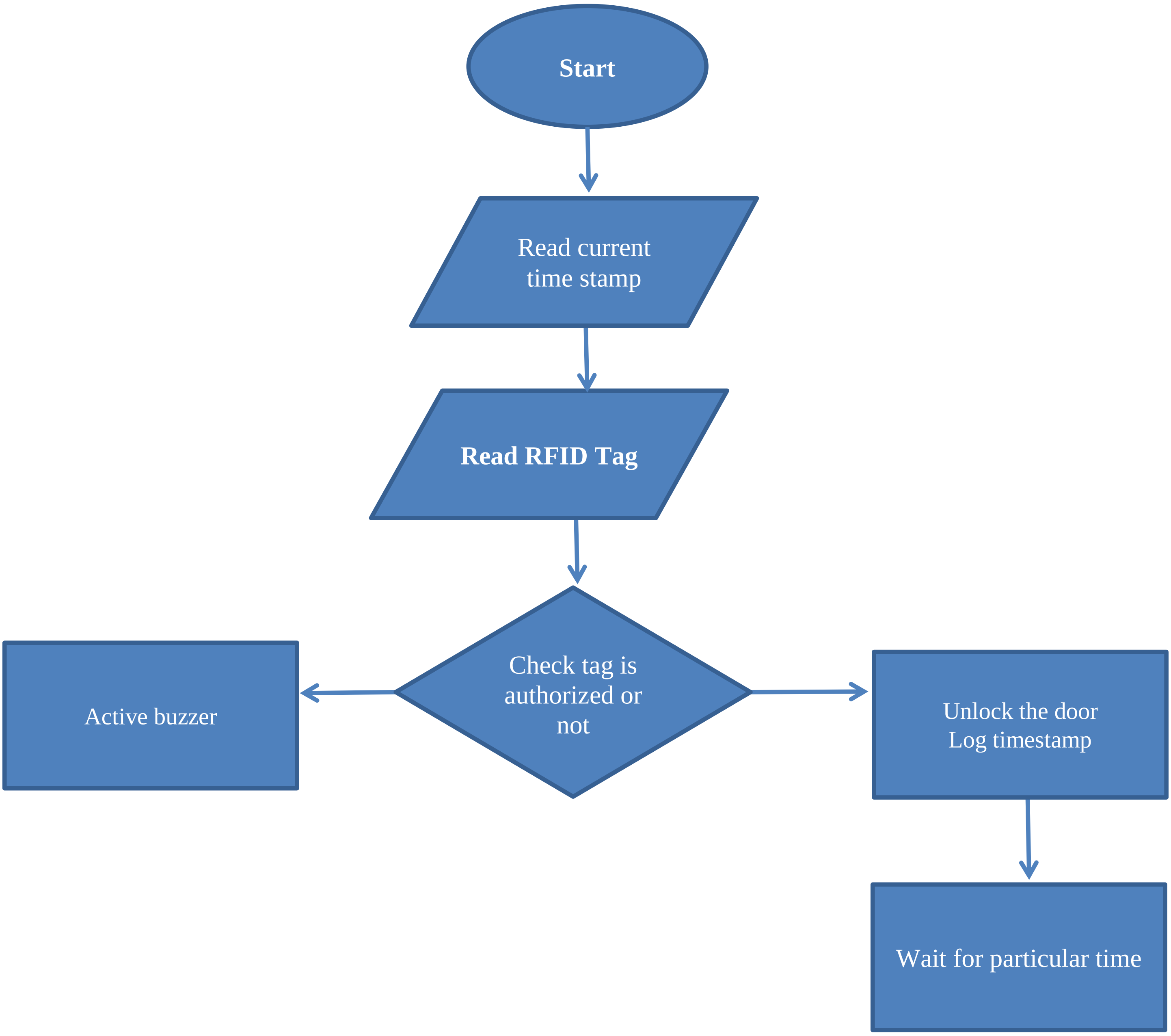
```
1. Start
2.  a. Initialize RFID reader to scan tags
    b. Load the list of authorized RFID tags from memory
    c. Initialize relay for granting access
    d. Initialize buzzer for access denial alerts
    e. Initialize SD card for logging access attempts with timestamps

2. Main Loop (Continuously monitor for RFID tag scans)
   a. Check if an RFID tag is scanned

   b. If a tag is scanned:
      i. Read the tag ID
      ii. Get the current timestamp

      iii. Compare the scanned tag ID with the authorized list
         - If the tag is authorized:
            - Activate the relay to grant access
            - Log "Access Granted" with timestamp and tag ID to the SD card
            - Keep the relay active for a short duration, then deactivate
         - If the tag is unauthorized:
            - Trigger the buzzer to indicate access denied
            - Log "Access Denied" with timestamp and tag ID to the SD card
```

Flowchart:


Start
Read current time stamp
Read RFID Tag
Check tag is authorized or not
Active buzzer
Unlock the door
Log timestamp
Wait for particular time