Problem Statement 1: Temperature Monitoring System

Objective: Design a temperature monitoring system that reads temperature data from a sensor and triggers an alarm if the temperature exceeds a predefined threshold.

Requirements:

Read temperature data from a temperature sensor at regular intervals.

Compare the read temperature with a predefined threshold.

If the temperature exceeds the threshold, activate an alarm (e.g., LED or buzzer).

Include functionality to reset the alarm.

Algorithm :

1. Initialize the system components like temperature sensor, alarm (buzzer) and button to rese
2. Start a loop to continuously monitor the temperature.
3. Read the temperature data from the sensor.
4. Set a threshold temperature (e.g. 30°C).
5. Compare the current temperature value with the threshold:
   - If the temperature exceeds the threshold turn on alarm.
   - If the temperature is below or equal to the threshold, keep monitoring.
6. Wait for button press to reset the alarm.
7. If button is pressed turn off the alarm and continue monitoring.

Problem Statement 2: Motor Control System

Objective: Implement a motor control system that adjusts the speed of a DC motor based on user input.

Requirements:

Use a potentiometer to read user input for desired motor speed.

Control the motor speed using PWM (Pulse Width Modulation).

Display the current speed on an LCD.

Algorithm :

1. Initialize the system components like potentiometer, DC motor and LCD display.
2. Read the analog value from the potentiometer.
3. Map the potentiometer value to a valid PWM range (0-1023 for 10-bit PWM).
4. Use the mapped PWM value to adjust the motor speed using a PWM signal.
5. Display the current motor speed on the LCD.

Problem Statement 3: LED Blinking Pattern

Objective: Create an embedded system that controls an array of LEDs to blink in a specific pattern based on user-defined settings.

Requirements:

Allow users to define blink patterns (e.g., fast, slow).

Implement different patterns using timers and interrupts.

Provide feedback through an LCD or serial monitor.

Algorithm :

1. Initialize LEDs, LCD, timer, and interrupt.
2. Wait for user input (e.g., button press)
3. If input is received:

   Set blink pattern (e.g., fast, slow etc.)

   Set blink speed (fast or slow)

4. Trigger LED blinking based on pattern and speed:
5. Use timer to control blink duration.
6. Update LCD/Serial monitor with current pattern and speed

Problem Statement 5: Data Logger

Objective: Develop a data logger that collects sensor data over time and stores it in non-volatile memory.

Requirements:

Read data from sensors (e.g., temperature, humidity) at specified intervals.

Store collected data in EEPROM or flash memory.

Implement functionality to retrieve and display logged data

Algorithm :

1. Initialize sensors (e.g., temperature, humidity), EEPROM,   LCD .
2. Wait for data collection interval (e.g., every minute)
3. Read data from sensors (e.g., temperature, humidity)
4. Store data in EEPROM or flash memory: (timestamp and sensor data).
5. Provide option to retrieve and display stored data (e.g., via button or serial input)

   If user requests data:

   - Retrieve data from EEPROM or flash memory

   - Display data on LCD or Serial Monitor

6. Repeat data collection and storage

Problem Statement: Write a program that functions as a simple calculator. It should be able to perform addition, subtraction, multiplication, and division based on user input.

Requirements:

1. Prompt the user to enter two numbers.
2. Ask the user to select an operation (addition, subtraction, multiplication, division).
3. Perform the selected operation and display the result.
4. Handle division by zero appropriately.

Pseudocode:

```
Enter num1 and num2
Enter the operation (op)
Switch(op)
case'+':
        then result = num1+ num2

case '-':

    then result = num1 - num2

case '*':
      then result = num1 * num2
case '/' :
     if num2 == 0
     return INVALID
     else
     result = num1 / num2
Print the result
```

Problem Statement: Write a program to calculate the factorial of a given non-negative integer.

Requirements:

1. Prompt the user to enter a non-negative integer.
2. Calculate the factorial using a loop.
3. Display the factorial of the number.

Pseudocode :

```
Read a non negative number from user = num
if  num is greater than or equal to 0
   factorial =1
   then, Run a loop i=1 to num
      factorial = factorial * i
      return factorial
```

Pseudocode(Using recursion):

```
//In main function

 Read number(num)

//call function

result=fact(num);

 print result;
```

```
//In function
if (num <= 1)
  return 1;
else
  return num*fact(num -1);
```

Problem Statement: Smart Irrigation System

Objective: Design a smart irrigation system that automatically waters plants based on soil moisture levels and environmental conditions. The system should monitor soil moisture and activate the water pump when the moisture level falls below a predefined threshold.

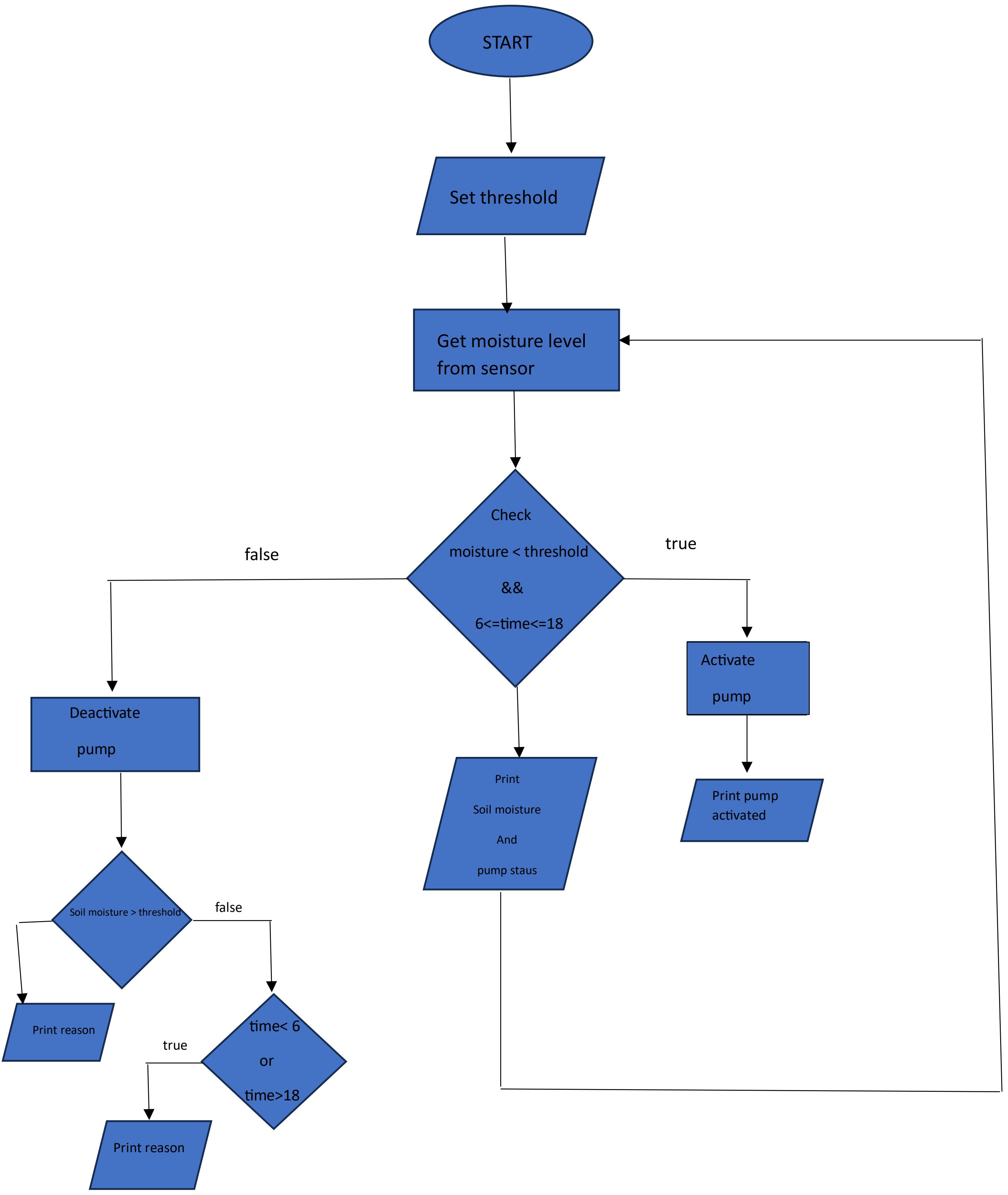
Requirements:

1. Inputs:
2. Outputs:
3. Conditions:
    - The pump should only activate if the soil moisture is below the threshold and it is daytime (e.g., between 6 AM and 6 PM).
    - If the soil moisture is adequate, the system should display a message indicating that watering is not needed.
    - Activate the water pump when the soil moisture is below the threshold.
    - Display the current soil moisture level and whether the pump is activated or not.
    - Soil moisture sensor reading (percentage).
    - User-defined threshold for soil moisture (percentage).
    - Time of day (to prevent watering during rain or at night).

Pseudo code :

```
Initialize soilMoistureSensor, waterPump, display, and clock
Set a moistureThreshold = 30%;

Loop indefinitely:
    //Read current soil moisture level
    soilMoistureLevel = read_soil_moisture(soilMoistureSensor) ;

    // Get current time
    time = get_current_hour(clock);    //24 hr format
```

```
  //Check conditions to activate pump
  If soilMoistureLevel < moistureThreshold AND (6 <= time < =18)
   {
       activate_pump(waterpump);
        printf("Soil moisture is low. Pump activated.");
 }
  else
 {
    deactivate_pump(waterpump);
           //Display the reason for no watering
           if soilMoistureLevel >= moistureThreshold:
                  printf("Soil moisture is adequate. No watering needed.");
            else if time< 6 OR currentHour >= 18:
                  printf("Outside watering hours. No watering needed.");
}

  //Display current soil moisture level and pump status
  printf("Current Soil Moisture: " soilMoistureLevel );
  printf("Pump Status: "   get_pump_status(waterpump) );

  //delay to prevent excessive reading
  delay(600) ;
```


START
Set threshold
Get moisture level from sensor
Check moisture < threshold && 6<=time<=18
false
true
Deactivate pump
Activate pump
Print Soil moisture And pump staus
Print pump activated
Soil moisture > threshold
false
Print reason
time< 6 or time>18
true
Print reason