weekend 2_task

---

1. Temperature Data Logger (2D Array)

Problem Statement: Design a program to log temperature readings from multiple sensors for 24 hours, sampled every hour.

Requirements:

Use a 2D array of size [N][24] to store temperature data, where N is the number of sensors (defined as a const variable).

Use static variables to calculate and store the daily average temperature for each sensor.

Use nested for loops to populate and analyze the array.

Use if statements to identify sensors exceeding a critical threshold temperature.

```
#include <stdio.h>
#define N 3  // Number of sensors
#define CRITICAL_TEMP 50

int main() {
    float tempData[N][24];
    float dailyAverage[N] = {0};

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < 24; j++) {
            tempData[i][j] = (i + 1) * 10 + j;  // Example values
            dailyAverage[i] += tempData[i][j];
        }
        dailyAverage[i] /= 24;
    }
```

```
    for (int i = 0; i < N; i++) {
        printf("Sensor %d average: %.2f\n", i + 1, dailyAverage[i]);
        if (dailyAverage[i] > CRITICAL_TEMP) {
            printf("Sensor %d exceeded critical threshold!\n", i + 1);
        }
    }
    return 0;
}
```

2. LED Matrix Control (2D Array)

Problem Statement: Simulate the control of an LED matrix of size 8x8. Each cell in the matrix can be ON (1) or OFF (0).

Requirements:

Use a 2D array to represent the LED matrix.

Use static variables to count the number of ON LEDs.

Use nested for loops to toggle the state of specific LEDs based on input commands.

Use if statements to validate commands (e.g., row and column indices).

```
#include <stdio.h>
#define SIZE 8

int main() {
    int ledMatrix[SIZE][SIZE] = {0};
    int onCount = 0;

    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if ((i + j) % 2 == 0) {
```

```
                ledMatrix[i][j] = 1;
                onCount++;
            }
        }
    }

    printf("Number of ON LEDs: %d\n", onCount);
    return 0;
}
```

3. Robot Path Mapping (2D Array)

Problem Statement: Track the movement of a robot on a grid of size M x N.

Requirements:

Use a 2D array to store visited positions (1 for visited, 0 otherwise).

Declare grid dimensions using const variables.

Use a while loop to update the robot's position based on input directions (e.g., UP, DOWN, LEFT, RIGHT).

Use if statements to ensure the robot stays within bounds.

```
#include <stdio.h>
#define M 5
#define N 5

int main() {
    int grid[M][N] = {0};
    int x = 0, y = 0;  // Starting position
    char direction;
```

```
    while (1) {
        printf("Enter direction (W/A/S/D or Q to quit): ");
        scanf(" %c", &direction);
        if (direction == 'Q') break;

        if (direction == 'W' && x > 0) x--;
        else if (direction == 'S' && x < M - 1) x++;
        else if (direction == 'A' && y > 0) y--;
        else if (direction == 'D' && y < N - 1) y++;

        grid[x][y] = 1;
    }

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", grid[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

4. Sensor Data Aggregation (3D Array)

Problem Statement: Store and analyze data from multiple sensors placed in a 3D grid (e.g., environmental sensors in a greenhouse).

Requirements:

Use a 3D array of size [X][Y][Z] to store data, where dimensions are defined using const variables.

Use nested for loops to populate the array with sensor readings.

Use if statements to find and count sensors reporting critical values (e.g., temperature > 50°C).

Use static variables to store aggregated results (e.g., average readings per layer).

```
#include <stdio.h>
#define X 3
#define Y 3
#define Z 3

int main() {
    float data[X][Y][Z];
    float avgPerLayer[X] = {0};
    int criticalCount = 0;

    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                data[i][j][k] = (i + j + k) * 10;  // Example data
                avgPerLayer[i] += data[i][j][k];
                if (data[i][j][k] > 50) criticalCount++;
            }
        }
        avgPerLayer[i] /= (Y * Z);
    }

    printf("Critical sensors count: %d\n", criticalCount);
    for (int i = 0; i < X; i++) {
        printf("Average for layer %d: %.2f\n", i + 1, avgPerLayer[i]);
```

```
    }

    return 0;
}
```

5. Image Processing (2D Array)

Problem Statement: Perform edge detection on a grayscale image represented as a 2D array.

Requirements:

Use a 2D array of size [H][W] to store pixel intensity values (defined using const variables).

Use nested for loops to apply a basic filter (e.g., Sobel filter) on the matrix.

Use decision-making statements to identify and highlight edge pixels (threshold-based).

Store the output image in a static 2D array.

```
#include <stdio.h>
#define H 3
#define W 3

int main() {
    int image[H][W] = {{10, 50, 10}, {30, 90, 30}, {10, 50, 10}};
    int edges[H][W] = {0};
    int threshold = 40;

    for (int i = 1; i < H - 1; i++) {
        for (int j = 1; j < W - 1; j++) {
            int gx = image[i - 1][j + 1] - image[i - 1][j - 1] +
                    2 * (image[i][j + 1] - image[i][j - 1]) +
```

```
                image[i + 1][j + 1] - image[i + 1][j - 1];
            if (gx > threshold) edges[i][j] = 1;
        }
    }

    printf("Edge Matrix:\n");
    for (int i = 0; i < H; i++) {
        for (int j = 0; j < W; j++) {
            printf("%d ", edges[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

6. Traffic Light Controller (State Management with 2D Array)

Problem Statement: Manage the states of traffic lights at an intersection with four roads, each having three lights (red, yellow, green).

Requirements:

Use a 2D array of size [4][3] to store the state of each light (1 for ON, 0 for OFF).

Use nested for loops to toggle light states based on time intervals.

Use static variables to keep track of the current state cycle.

Use if statements to validate light transitions (e.g., green should not overlap with red).

```
#include <stdio.h>
#define ROADS 4
#define LIGHTS 3  // Red, Yellow, Green
```

```
int main() {
    int trafficLights[ROADS][LIGHTS] = {{1, 0, 0}, {1, 0, 0}, {1, 0, 0}, {1, 0, 0}};
    int cycle = 0;

    for (int t = 0; t < 10; t++) {  // Simulate 10 cycles
        printf("Cycle %d:\n", t + 1);
        for (int i = 0; i < ROADS; i++) {
            cycle = (cycle + 1) % 3;  // Change light state
            trafficLights[i][0] = (cycle == 0);  // Red
            trafficLights[i][1] = (cycle == 1);  // Yellow
            trafficLights[i][2] = (cycle == 2);  // Green
            printf("Road %d: R=%d, Y=%d, G=%d\n", i + 1, trafficLights[i][0],
trafficLights[i][1], trafficLights[i][2]);
        }
        printf("\n");
    }

    return 0;
}
```

7. 3D LED Cube Animation (3D Array)

Problem Statement: Simulate an animation on an LED cube of size 4x4x4.

Requirements:

Use a 3D array to represent the LED cube's state.

Use nested for loops to turn ON/OFF LEDs in a predefined pattern.

Use static variables to store animation progress and frame counters.

Use if-else statements to create transitions between animation frames.

```
#include <stdio.h>
#define SIZE 4

int main() {
    int ledCube[SIZE][SIZE][SIZE] = {{{0}}};
    int frame = 0;

    for (frame = 0; frame < SIZE; frame++) {  // Simple animation
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                ledCube[frame][i][j] = 1;  // Turn on one layer at a time
            }
        }

        printf("Frame %d:\n", frame + 1);
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                printf("%d ", ledCube[frame][i][j]);
            }
            printf("\n");
        }
        printf("\n");
    }

    return 0;
}
```

8. Warehouse Inventory Tracking (3D Array)

Problem Statement: Track inventory levels for multiple products stored in a 3D warehouse (e.g., rows, columns, and levels).

Requirements:

Use a 3D array of size [P][R][C] to represent the inventory of P products in a grid.

Use nested for loops to update inventory levels based on shipments.

Use if statements to detect low-stock levels in any location.

Use a static variable to store total inventory counts for each product.

```
#include <stdio.h>
#define P 2  // Products
#define R 3  // Rows
#define C 3  // Columns

int main() {
    int warehouse[P][R][C] = {{{10, 20, 15}, {30, 25, 10}, {5, 10, 20}},
                              {{15, 10, 25}, {20, 15, 30}, {10, 5, 15}}};
    int totalInventory[P] = {0};

    for (int p = 0; p < P; p++) {
        for (int r = 0; r < R; r++) {
            for (int c = 0; c < C; c++) {
                totalInventory[p] += warehouse[p][r][c];
                if (warehouse[p][r][c] < 10) {
                    printf("Low stock for Product %d at (%d, %d)\n", p + 1, r, c);
                }
            }
        }
        printf("Total inventory for Product %d: %d\n", p + 1, totalInventory[p]);
```

```
    }

    return 0;
}
```

9. Signal Processing on a 3D Matrix

Problem Statement: Apply a basic signal filter to a 3D matrix representing sampled signals over time.

Requirements:

Use a 3D array of size [X][Y][Z] to store signal data.

Use nested for loops to apply a filter that smoothens the signal values.

Use if statements to handle boundary conditions while processing the matrix.

Store the filtered results in a static 3D array.

```
#include <stdio.h>
#define X 3
#define Y 3
#define Z 3

int main() {
    float signal[X][Y][Z] = {{{10, 20, 30}, {20, 30, 40}, {30, 40, 50}},
                             {{15, 25, 35}, {25, 35, 45}, {35, 45, 55}},
                             {{20, 30, 40}, {30, 40, 50}, {40, 50, 60}}};
    float smoothed[X][Y][Z] = {{{0}}};

    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
```

```
            float sum = signal[i][j][k];
            int count = 1;

            // Check neighbors
            for (int di = -1; di <= 1; di++) {
                for (int dj = -1; dj <= 1; dj++) {
                    for (int dk = -1; dk <= 1; dk++) {
                        int ni = i + di, nj = j + dj, nk = k + dk;
                        if (ni >= 0 && ni < X && nj >= 0 && nj < Y && nk >= 0 && nk < Z) {
                            sum += signal[ni][nj][nk];
                            count++;
                        }
                    }
                }
            }

            smoothed[i][j][k] = sum / count;
        }
    }
}

printf("Smoothed Signal:\n");
for (int i = 0; i < X; i++) {
    for (int j = 0; j < Y; j++) {
        for (int k = 0; k < Z; k++) {
            printf("%.2f ", smoothed[i][j][k]);
        }
        printf("\n");
    }
```

```
        printf("\n");
    }

    return 0;
}
```

10. Weather Data Analysis (3D Array)

Problem Statement: Analyze weather data recorded over multiple locations and days, with hourly samples for each day.

Requirements:

Use a 3D array of size [D][L][H] to store temperature readings (D days, L locations, H hours per day).

Use nested for loops to calculate the average daily temperature for each location.

Use if statements to find the location and day with the highest temperature.

Use static variables to store results for each location.

```
#include <stdio.h>
#define D 2  // Days
#define L 2  // Locations
#define H 3  // Hours per day

int main() {
    float weather[D][L][H] = {{{20.5, 22.1, 19.8}, {25.0, 23.5, 21.0}},
                              {{21.0, 20.5, 18.5}, {26.0, 24.5, 22.5}}};
    float dailyAvg[L] = {0};
    float maxTemp = -1;
    int maxDay = 0, maxLocation = 0;

    for (int d = 0; d < D; d++) {
        for (int l = 0; l < L; l++) {
```

```
            float sum = 0;
            for (int h = 0; h < H; h++) {
                sum += weather[d][l][h];
                if (weather[d][l][h] > maxTemp) {
                    maxTemp = weather[d][l][h];
                    maxDay = d;
                    maxLocation = l;
                }
            }
            dailyAvg[l] += sum / H;
        }
    }

    for (int l = 0; l < L; l++) {
        printf("Location %d average: %.2f\n", l + 1, dailyAvg[l] / D);
    }
    printf("Highest temperature: %.2f on Day %d at Location %d\n", maxTemp,
maxDay + 1, maxLocation + 1);

    return 0;
}
```