2D AND 3D PROGRAMS DEC 29

1.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 5
#define THRESHOLD 80.0

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

    float avgTemperatures[N] = { 0 };

    for (int sensor = 0; sensor < N; sensor++) {
        printf("Enter the hourly temperature readings for Sensor %d (24 values):\n", sensor + 1);
        for (int hour = 0; hour < 24; hour++) {
            printf("Hour %d: ", hour + 1);
            scanf("%f", &temperatures[sensor][hour]);

            avgTemperatures[sensor] += temperatures[sensor][hour];
        }
    }

    printf("\nDaily Average Temperatures for each sensor:\n");
    for (int sensor = 0; sensor < N; sensor++) {
        avgTemperatures[sensor] /= 24;
        printf("Sensor %d: %.2f °C\n", sensor + 1, avgTemperatures[sensor]);
    }

    printf("\nSensors exceeding the critical threshold (%.2f °C):\n", THRESHOLD);
    for (int sensor = 0; sensor < N; sensor++) {
        int exceededThreshold = 0;
        for (int hour = 0; hour < 24; hour++) {
            if (temperatures[sensor][hour] > THRESHOLD) {
                exceededThreshold = 1;
                break;
            }
        }

        if (exceededThreshold) {
            printf("Sensor %d has exceeded the critical threshold!\n", sensor + 1);
        }
    }

    return 0;
}
```

2.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 R 8
#define C 8

int matrix[R][C] = {0};

static int countOnLEDs = 0;

void toggleLED(int r, int c) {
    if (r >= 0 && r < R && c >= 0 && c < C) {
        if (matrix[r][c] == 1) {
            matrix[r][c] = 0;
            countOnLEDs--;
        }
        else {
            matrix[r][c] = 1;
            countOnLEDs++;
        }
    } else {
        printf("Invalid row or column index. Please use values between 0 and 7.\n");
    }
}

void displayMatrix() {
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < C; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    printf("Total ON LEDs: %d\n", countOnLEDs);
}

int main() {
    int r, c;
    char command;

    printf("Initial LED Matrix:\n");
    displayMatrix();

    while (1) {
        printf("\nEnter command (t <row> <col>) to toggle LED, or 'q' to quit: ");
        scanf(" %c", &command);

        if (command == 'q') {
            break;
        } else if (command == 't') {
```

```
            scanf("%d %d", &r, &c);
            toggleLED(r, c);
            displayMatrix();
        } else {
            printf("Invalid command. Use 't <row> <col>' to toggle or 'q' to quit.\n");
        }
    }

    return 0;
}
```

3.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

// Function to print the grid
void printGrid(int grid[M][N]) {
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", grid[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int grid[M][N] = {0};

    int x = 0, y = 0;
    grid[x][y] = 1;

    char direction;
    while (1) {
        printf("Enter direction (U=Up, D=Down, L=Left, R=Right, Q=Quit): ");
        scanf(" %c", &direction);

        if (direction == 'Q' || direction == 'q') {
            break;
        }

        if (direction == 'U' || direction == 'u') {
            if (x > 0) {
                grid[x][y] = 1;
                x--;
            }
        }
```

```
        else if (direction == 'D' || direction == 'd') {
            if (x < M - 1) {
                grid[x][y] = 1;
                x++;
            }
        }
        else if (direction == 'L' || direction == 'l') {
            if (y > 0) {
                grid[x][y] = 1;
                y--;
            }
        }
        else if (direction == 'R' || direction == 'r') {
            if (y < N - 1) {
                grid[x][y] = 1;
                y++;
            }
        }
        else {
            printf("Invalid direction!\n");
            continue;
        }

        grid[x][y] = 1;

        printGrid(grid);
    }

    printf("Robot movement terminated.\n");

    return 0;
}
```

4.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 5
#define Y 5
#define Z 3

float sensorData[X][Y][Z];

static int criticalCount = 0;
static float layerAverages[Z];

void populateSensorData() {
    for (int x = 0; x < X; x++) {
        for (int y = 0; y < Y; y++) {
```

```
            for (int z = 0; z < Z; z++) {
                sensorData[x][y][z] = (rand() % 100);
            }
        }
    }
}

void analyzeSensorData() {
    float layerSum[Z] = {0};
    int sensorCount[Z] = {0};

    for (int x = 0; x < X; x++) {
        for (int y = 0; y < Y; y++) {
            for (int z = 0; z < Z; z++) {
                float temp = sensorData[x][y][z];

                if (temp > 50.0) {
                    criticalCount++;
                }

                layerSum[z] += temp;
                sensorCount[z]++;
            }
        }
    }

    for (int z = 0; z < Z; z++) {
        if (sensorCount[z] > 0) {
            layerAverages[z] = layerSum[z] / sensorCount[z];
        }
    }
}

void printResults() {
    printf("Critical sensor readings (temperature > 50): %d\n", criticalCount);

    for (int z = 0; z < Z; z++) {
        printf("Average temperature for layer %d: %.2f°C\n", z, layerAverages[z]);
    }
}

int main() {
    populateSensorData();

    analyzeSensorData();

    printResults();
    return 0;
}
```

5. 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 5
#define W 5

const int Gx[3][3] = {
    {-1, 0, 1},
    {-2, 0, 2},
    {-1, 0, 1}
};

const int Gy[3][3] = {
    {-1, -2, -1},
    { 0,  0,  0},
    { 1,  2,  1}
};

void sobelEdgeDetection(int input[H][W], int output[H][W], int threshold) {
    for (int i = 1; i < H - 1; i++) {
        for (int j = 1; j < W - 1; j++) {
            int GxSum = 0;
            int GySum = 0;

            for (int k = -1; k <= 1; k++) {
                for (int l = -1; l <= 1; l++) {
                    GxSum += input[i + k][j + l] * Gx[k + 1][l + 1];
                    GySum += input[i + k][j + l] * Gy[k + 1][l + 1];
                }
            }

            int magnitudeSquared = (GxSum * GxSum) + (GySum * GySum);

            if (magnitudeSquared > threshold * threshold) {
                output[i][j] = 255;
            } else {
                output[i][j] = 0;
            }
        }
    }
}

int main() {
    int inputImage[H][W] = {
        { 52, 55, 61, 59, 79},
        { 62, 59, 55, 104, 94},
        { 63, 65, 66, 113, 107},
        { 72, 70, 77, 138, 145},
        { 63, 67, 67, 114, 121}
    };

    int outputImage[H][W] = {0};

    int threshold = 100;
```

```
    sobelEdgeDetection(inputImage, outputImage, threshold);

    printf("Edge Detected Image (0=Black, 255=White):\n");
    for (int i = 0; i < H; i++) {
        for (int j = 0; j < W; j++) {
            printf("%3d ", outputImage[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

6.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>
#include <unistd.h>

#define ROADS 4
#define LIGHTS 3

#define RED 0
#define YELLOW 1
#define GREEN 2

int trafficLights[ROADS][LIGHTS] = {
    {RED, YELLOW, GREEN},
    {RED, YELLOW, GREEN},
    {RED, YELLOW, GREEN},
    {RED, YELLOW, GREEN}
};

static int currentState[ROADS] = {GREEN, GREEN, GREEN, GREEN};
static int timeCounter = 0;

void printLights() {
    for (int i = 0; i < ROADS; i++) {
        printf("Road %d: ", i + 1);
        for (int j = 0; j < LIGHTS; j++) {
            if (trafficLights[i][j] == RED) {
                printf("Red ");
            } else if (trafficLights[i][j] == YELLOW) {
                printf("Yellow ");
            } else if (trafficLights[i][j] == GREEN) {
                printf("Green ");
            }
        }
        printf("\n");
```

```
    }
}

void toggleLights() {
    for (int i = 0; i < ROADS; i++) {
        if (currentState[i] == GREEN) {
            trafficLights[i][GREEN] = 0;
            trafficLights[i][YELLOW] = 1;
            currentState[i] = YELLOW;
        } else if (currentState[i] == YELLOW) {
            // Change YELLOW to RED
            trafficLights[i][YELLOW] = 0;
            trafficLights[i][RED] = 1;
            currentState[i] = RED;
        } else if (currentState[i] == RED) {
            trafficLights[i][RED] = 0;
            trafficLights[i][GREEN] = 1;
            currentState[i] = GREEN;
        }
    }

    for (int i = 0; i < ROADS; i++) {
        for (int j = 0; j < ROADS; j++) {
            if (i != j) {
                if (trafficLights[i][GREEN] == 1 && trafficLights[j][GREEN] == 1) {
                    trafficLights[j][GREEN] = 0;
                    trafficLights[j][RED] = 1;
                    currentState[j] = RED;
                }
            }
        }
    }
}

int main() {
    while (1) {
        printLights();
        toggleLights();
        sleep(3);
        timeCounter++;
    }

    return 0;
}
```

7.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 ledCube[SIZE][SIZE][SIZE] = {{{0}}};

static int animationProgress = 0;

void printCube() {
    printf("LED Cube State:\n");
    for (int z = 0; z < SIZE; z++) {
        for (int y = 0; y < SIZE; y++) {
            for (int x = 0; x < SIZE; x++) {
                printf("%d ", ledCube[x][y][z]);
            }
            printf("\n");
        }
        printf("\n");
    }
    printf("\n");
}

void clearCube() {
    for (int x = 0; x < SIZE; x++) {
        for (int y = 0; y < SIZE; y++) {
            for (int z = 0; z < SIZE; z++) {
                ledCube[x][y][z] = 0;
            }
        }
    }
}

void delay(int cycles) {
    for (volatile int i = 0; i < cycles; i++) {
    }
}

void animate() {
    switch (animationProgress) {
        case 0:
            clearCube();
            for (int z = 0; z < SIZE; z++) {
                ledCube[0][0][z] = 1;
            }
            break;
        case 1:
            clearCube();
            for (int z = 0; z < SIZE; z++) {
                ledCube[1][1][z] = 1;
            }
            break;
        case 2:
            clearCube();
            for (int z = 0; z < SIZE; z++) {
                ledCube[2][2][z] = 1;
            }
            break;
```

```
        case 3:
            clearCube();
            for (int z = 0; z < SIZE; z++){
              ledCube[3][3][z] = 1;
            }
            break;
        default:
            animationProgress = 0;
            break;
    }

    animationProgress++;
    if (animationProgress > 3) {
        animationProgress = 0;
    }

    delay(1000000);
}

int main() {
    while (1) {
        animate();
        printCube();
    }
    return 0;
}
```

8.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 SIZE 4

int ledCube[SIZE][SIZE][SIZE] = {{{0}}};

static int animationProgress = 0;

void printCube() {
    printf("LED Cube State:\n");
    for (int z = 0; z < SIZE; z++) {
        for (int y = 0; y < SIZE; y++) {
            for (int x = 0; x < SIZE; x++) {
                printf("%d ", ledCube[x][y][z]);
            }
            printf("\n");
        }
        printf("\n");
    }
    printf("\n");
}
```

```
void clearCube() {
    for (int x = 0; x < SIZE; x++) {
        for (int y = 0; y < SIZE; y++) {
            for (int z = 0; z < SIZE; z++) {
                ledCube[x][y][z] = 0;
            }
        }
    }
}

void delay(int cycles) {
    for (volatile int i = 0; i < cycles; i++) {
    }
}

void animate() {
    switch (animationProgress) {
        case 0:
            clearCube();
            for (int z = 0; z < SIZE; z++) {
                ledCube[0][0][z] = 1;
            }
            break;
        case 1:
            clearCube();
            for (int z = 0; z < SIZE; z++) {
                ledCube[1][1][z] = 1;
            }
            break;
        case 2:
            clearCube();
            for (int z = 0; z < SIZE; z++) {
                ledCube[2][2][z] = 1;
            }
            break;
        case 3:
            clearCube();
            for (int z = 0; z < SIZE; z++){
              ledCube[3][3][z] = 1;
            }
            break;
        default:
            animationProgress = 0;
            break;
    }

    animationProgress++;
    if (animationProgress > 3) {
        animationProgress = 0;
    }

    delay(1000000);
}
```

```
int main() {
    while (1) {
        animate();
        printCube();
    }
    return 0;
}
```

9. 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 5
#define Y 5
#define Z 5
void applyFilter(double input[X][Y][Z], double output[X][Y][Z]) {
    int i, j, k, xi, yi, zi;
    double sum, count;

    for (i = 0; i < X; i++) {
        for (j = 0; j < Y; j++) {
            for (k = 0; k < Z; k++) {
                sum = 0.0;
                count = 0;

                for (xi = i - 1; xi <= i + 1; xi++) {
                    for (yi = j - 1; yi <= j + 1; yi++) {
                        for (zi = k - 1; zi <= k + 1; zi++) {
                            // Check if the neighbor is within the boundaries of the matrix
                            if (xi >= 0 && xi < X && yi >= 0 && yi < Y && zi >= 0 && zi < Z) {
                                sum += input[xi][yi][zi];
                                count++;
                            }
                        }
                    }
                }

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

int main() {
    double input[X][Y][Z] = {
        {
            {1, 2, 3, 4, 5},
            {6, 7, 8, 9, 10},
            {11, 12, 13, 14, 15},
            {16, 17, 18, 19, 20},
```

```
            {21, 22, 23, 24, 25}
        },
        {
            {26, 27, 28, 29, 30},
            {31, 32, 33, 34, 35},
            {36, 37, 38, 39, 40},
            {41, 42, 43, 44, 45},
            {46, 47, 48, 49, 50}
        },
        {
            {51, 52, 53, 54, 55},
            {56, 57, 58, 59, 60},
            {61, 62, 63, 64, 65},
            {66, 67, 68, 69, 70},
            {71, 72, 73, 74, 75}
        },
        {
            {76, 77, 78, 79, 80},
            {81, 82, 83, 84, 85},
            {86, 87, 88, 89, 90},
            {91, 92, 93, 94, 95},
            {96, 97, 98, 99, 100}
        },
        {
            {101, 102, 103, 104, 105},
            {106, 107, 108, 109, 110},
            {111, 112, 113, 114, 115},
            {116, 117, 118, 119, 120},
            {121, 122, 123, 124, 125}
        }
    };

    double output[X][Y][Z];

    applyFilter(input, output);

    printf("Filtered signal data:\n");
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("%.2f ", output[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }

    return 0;
}
```

10.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 5
#define L 3
#define H 24

static int highestTemp = -1000;
static int highestTempDay = -1;
static int highestTempLocation = -1;

int main() {
    float temperatures[D][L][H] = {
        {{15.0, 16.5, 17.0, 18.5, 19.0, 20.0, 21.5, 22.0, 23.0, 24.5, 25.0, 26.0, 27.0, 28.0, 28.5, 29.0, 29.5,
30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0},
         {14.0, 15.5, 16.0, 17.5, 18.0, 19.0, 20.5, 21.0, 22.0, 23.5, 24.0, 25.0, 26.0, 27.0, 27.5, 28.0, 28.5,
29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0},
         {13.0, 14.5, 15.0, 16.5, 17.0, 18.0, 19.5, 20.0, 21.0, 22.5, 23.0, 24.0, 25.0, 26.0, 26.5, 27.0, 27.5,
28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0}},
        {{16.0, 17.5, 18.0, 19.5, 20.0, 21.0, 22.5, 23.0, 24.0, 25.5, 26.0, 27.0, 28.0, 29.0, 29.5, 30.0, 30.5,
31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0},
         {14.5, 16.0, 16.5, 18.0, 18.5, 19.5, 21.0, 21.5, 22.5, 24.0, 24.5, 25.5, 26.5, 27.5, 28.0, 28.5, 29.0,
29.5, 30.5, 31.5, 32.0, 33.0, 34.0, 35.0},
         {12.5, 14.0, 14.5, 16.0, 16.5, 17.5, 19.0, 19.5, 20.5, 22.0, 22.5, 23.5, 24.5, 25.5, 26.0, 26.5, 27.0,
27.5, 28.5, 29.5, 30.0, 31.0, 32.0, 33.0}},
        {{15.5, 16.5, 17.0, 18.0, 19.5, 20.5, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 30.5, 31.0,
31.5, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0},
         {14.0, 15.0, 15.5, 17.0, 18.0, 19.0, 20.0, 21.5, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 28.5, 29.0,
30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0},
         {13.5, 14.5, 15.0, 16.0, 16.5, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0,
30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0}}
    };

    float dailyAvgTemp[L];
    float totalTemp;

    for (int day = 0; day < D; day++) {
        for (int loc = 0; loc < L; loc++) {
            totalTemp = 0.0;
            for (int hour = 0; hour < H; hour++) {
                totalTemp += temperatures[day][loc][hour];
            }
            dailyAvgTemp[loc] = totalTemp / H;
            printf("Day %d, Location %d: Average Daily Temperature = %.2f°C\n", day + 1, loc + 1,
dailyAvgTemp[loc]);

            for (int hour = 0; hour < H; hour++) {
                if (temperatures[day][loc][hour] > highestTemp) {
                    highestTemp = temperatures[day][loc][hour];
                    highestTempDay = day;
                    highestTempLocation = loc;
                }
            }
        }
```

```
    }

    printf("\nHighest Temperature Recorded: %.2f°C\n", highestTemp);
    printf("Occurred on Day %d, Location %d\n", highestTempDay + 1, highestTempLocation + 1);

    return 0;
}
```