Weekend 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 SENSORS 5
#define HOURS 4
 void Readings(float temp[SENSORS][HOURS])
 {
    static int hour = 0;
    int sensor;

 printf("Logging temperature readings for hour %d:\n", hour + 1);
   for(sensor = 0; sensor < SENSORS; sensor++)
   {
      printf("Enter temperature for Sensor %d: ", sensor + 1);
      scanf("%f", &temp[sensor][hour]);
   }

   hour++;
   if (hour == HOURS)
   {
      hour = 0;
    }
}
void displayReadings(float temp[SENSORS][HOURS])
{
    int sensor, hour;

    printf("\nTemperature readings for 24 hours:\n");
    for(sensor = 0; sensor < SENSORS; sensor++) {
        printf("Sensor %d: ", sensor + 1);
        for(hour = 0; hour < HOURS; hour++) {
            printf("%.2f ", temp[sensor][hour]);
        }
        printf("\n");
    }
}

int main()

{
    float temperatureReadings[SENSORS][HOURS] = {0};
    int i;


    for(i = 0; i < HOURS; i++) {
        Readings(temperatureReadings);
    }


    displayReadings(temperatureReadings);


    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

void toggleLED(int matrix[SIZE][SIZE], int row, int col, int *onCount)
{
  if (row >= 0 && row < SIZE && col >= 0 && col < SIZE) {
    if (matrix[row][col] == 1)
    {
      matrix[row][col] = 0;
      (*onCount)--;
    } else {
      matrix[row][col] = 1;
      (*onCount)++;
    }
  } else {
    printf("Invalid command: row and column indices must be between 0 and 7.\n");
  }
}
```

```
void printMatrix(int matrix[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int matrix[SIZE][SIZE] = {0};
    static int onCount = 0;
    int row, col;
    char command;

    printf("LED Matrix Control\n");
    printf("Enter commands to toggle LEDs (format: T row col, Q to quit):\n");

    while (1) {
        printf("> ");
        scanf(" %c", &command);

        if (command == 'Q' || command == 'q') {
            break;
        } else if (command == 'T' || command == 't') {
            scanf("%d %d", &row, &col);
            toggleLED(matrix, row, col, &onCount);
            printf("Number of LEDs ON: %d\n", onCount);
        } else {
            printf("Invalid command. Use 'T row col' to toggle or 'Q' to quit.\n");
        }

        printMatrix(matrix);
    }

    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



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 robotX = 0, robotY = 0;
    char direction;
    int visitedCount = 0;

    grid[robotX][robotY] = 1;
    visitedCount++;

    printf("Robot Movement\n");
    printf("Use commands: W (UP), S (DOWN), A (LEFT), D (RIGHT), Q (QUIT)\n");

    while (1)
    {
        printf("\nEnter direction: ");
        scanf(" %c", &direction);

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



        if (direction == 'W' || direction == 'w') {
            if (robotX > 0) robotX--;
        } else if (direction == 'S' || direction == 's') {
            if (robotX < M - 1) robotX++;
        } else if (direction == 'A' || direction == 'a') {
            if (robotY > 0) robotY--;
        } else if (direction == 'D' || direction == 'd') {
            if (robotY < N - 1) robotY++;
        } else {
            printf("Invalid direction. Use W, S, A, D for directions or Q to quit.\n");
            continue;
        }



        if (grid[robotX][robotY] == 0) {
            grid[robotX][robotY] = 1;
            visitedCount++;
        }

        printf("Current position: (%d, %d)\n", robotX, robotY);
        printf("Visited cells: %d\n", visitedCount);
        printGrid(grid);
    }

    printf("Final position: (%d, %d)\n", robotX, robotY);
    printf("Total visited cells: %d\n", visitedCount);

    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 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. 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 5
#define W 5
#define THRESHOLD 100



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

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


int abs(int value) {
    return value < 0 ? -value : value;
}


void applySobelFilter(int image[H][W], int output[H][W]) {
    int x, y, i, j;
    for (x = 1; x < H - 1; x++) {
        for (y = 1; y < W - 1; y++) {
            int gx = 0;
            int gy = 0;

            for (i = -1; i <= 1; i++) {
                for (j = -1; j <= 1; j++) {
                    gx += image[x + i][y + j] * Gx[i + 1][j + 1];
                    gy += image[x + i][y + j] * Gy[i + 1][j + 1];
                }
            }

            int magnitude = abs(gx) + abs(gy);
            if (magnitude > THRESHOLD) {
                output[x][y] = 255;
            } else {
                output[x][y] = 0;
            }
        }
    }
}


void printImage(int image[H][W]) {
    for (int i = 0; i < H; i++) {
        for (int j = 0; j < W; j++) {
            printf("%d ", image[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int image[H][W] = {
        {10, 10, 10, 10, 10},
        {10, 50, 50, 50, 10},
        {10, 50, 100, 50, 10},
        {10, 50, 50, 50, 10},
        {10, 10, 10, 10, 10}
    };
    static int output[H][W] = {0};

    printf("Original Image:\n");
    printImage(image);
    applySobelFilter(image, output);


    printf("\nEdge Detected Image:\n");
    printImage(output);


    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



void printLights(int lights[ROADS][LIGHTS]) {
    const char *lightColors[] = {"Red", "Yellow", "Green"};
    for (int i = 0; i < ROADS; i++) {
        printf("Road %d: ", i + 1);
        for (int j = 0; j < LIGHTS; j++) {
            printf("%s: %d ", lightColors[j], lights[i][j]);
        }
        printf("\n");
    }
}


int main() {
    int lights[ROADS][LIGHTS] = {0};
    static int cycle = 0;
    int i, j;



    for (i = 0; i < ROADS; i++)
    {
        lights[i][0] = 1;
    }

    while (cycle < 9) {

        printf("Cycle %d:\n", cycle + 1);
        printLights(lights);



        for (i = 0; i < ROADS; i++) {
            if (cycle % 3 == 0) {

                lights[i][0] = 1;
                lights[i][1] = 0;
                lights[i][2] = 0;
            } else if (cycle % 3 == 1) {

                lights[i][0] = 0;
                lights[i][1] = 1;
                lights[i][2] = 0;
            } else {

                lights[i][0] = 0;
                lights[i][1] = 0;
                lights[i][2] = 1;
            }
        }


        for (i = 0; i < ROADS; i++) {
            for (j = 0; j < ROADS; j++) {
                if (lights[i][2] == 1 && lights[j][0] == 1) {
                    printf("Invalid state detected! Green and Red overlap on road %d and road %d\n", i + 1, j
+ 1);
                    return 1;
                }
            }
        }


        for (long int delay = 0; delay < 100000000; delay++) {

        }
        cycle++;
    }

    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.

```c
#include <stdio.h>

#define SIZE 4
#define FRAMES 10

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

void animateLED(int cube[SIZE][SIZE][SIZE], int frame) {

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

    if (frame % 2 == 0) {

        for (int i = 0; i < SIZE; i++) {
            cube[i][i][i] = 1;
        }
    } else {

        for (int i = 0; i < SIZE; i++) {
            cube[SIZE - 1 - i][i][i] = 1;
        }
    }
}

int main() {
    int cube[SIZE][SIZE][SIZE] = {0};
    static int frame = 0;
    int animationProgress = 0;

    while (animationProgress < FRAMES) {
        animateLED(cube, frame);
        printf("Frame %d:\n", frame);
        printCube(cube);



        for (long int delay = 0; delay < 100000000; delay++) {
        }

        frame++;
        animationProgress++;
    }

    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 PRODUCTS 3
#define ROWS 4
#define COLUMNS 5
#define LOW_STOCK_THRESHOLD 10


void updateInventory(int inventory[PRODUCTS][ROWS][COLUMNS]) {
    for (int p = 0; p < PRODUCTS; p++) {
        for (int r = 0; r < ROWS; r++) {
            for (int c = 0; c < COLUMNS; c++) {
                printf("Enter shipment quantity for Product %d at position (%d, %d): ", p + 1, r, c);
                scanf("%d", &inventory[p][r][c]);
            }
        }
    }
}


void analyzeInventory(int inventory[PRODUCTS][ROWS][COLUMNS]) {
    static int totalInventory[PRODUCTS] = {0};
    for (int p = 0; p < PRODUCTS; p++)
    {
        totalInventory[p] = 0;
        for (int r = 0; r < ROWS; r++) {
            for (int c = 0; c < COLUMNS; c++) {
                totalInventory[p] += inventory[p][r][c];
                if (inventory[p][r][c] < LOW_STOCK_THRESHOLD) {
                    printf("Low stock detected for Product %d at position (%d, %d) with quantity %d\n", p + 1,
r, c, inventory[p][r][c]);
                }
            }
        }
        printf("Total inventory for Product %d: %d\n", p + 1, totalInventory[p]);
    }
}

int main() {
    int inventory[PRODUCTS][ROWS][COLUMNS] = {0};



    updateInventory(inventory);
    analyzeInventory(inventory);

    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 4
#define Y 4
#define Z 4


void applyFilter(float input[X][Y][Z], float output[X][Y][Z]) {
    int i, j, k, ii, jj, kk;
    int neighbors;
    float sum;

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

                for (ii = i - 1; ii <= i + 1; ii++) {
                    for (jj = j - 1; jj <= j + 1; jj++) {
                        for (kk = k - 1; kk <= k + 1; kk++) {
                            if (ii >= 0 && ii < X && jj >= 0 && jj < Y && kk >= 0 && kk < Z) {
                                sum += input[ii][jj][kk];
                                neighbors++;
                            }
                        }
                    }
                }

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

void printMatrix(float matrix[X][Y][Z]) {
    for (int i = 0; i < X; i++) {
        printf("Layer %d:\n", i);
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("%.2f ", matrix[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
}

int main() {

    float input[X][Y][Z] = {
        {
            {1.0, 2.0, 3.0, 4.0},
            {5.0, 6.0, 7.0, 8.0},
            {9.0, 10.0, 11.0, 12.0},
            {13.0, 14.0, 15.0, 16.0}
        },
        {
            {17.0, 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},
            {37.0, 38.0, 39.0, 40.0},
            {41.0, 42.0, 43.0, 44.0},
            {45.0, 46.0, 47.0, 48.0}
        },
        {
            {49.0, 50.0, 51.0, 52.0},
            {53.0, 54.0, 55.0, 56.0},
            {57.0, 58.0, 59.0, 60.0},
            {61.0, 62.0, 63.0, 64.0}
        }
    };
    static float output[X][Y][Z] = {0};


    printf("Original Matrix:\n");
    printMatrix(input);



    applyFilter(input, output);


    printf("Filtered Matrix:\n");
    printMatrix(output);


    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 3 // Number of days
#define L 2 // Number of locations
#define H 4 // Number of hours per day
```

```
// Function to calculate average daily temperatures
void calculateAverageDailyTemperature(float data[D][L][H], float avgTemps[L]) {
    static float locationAverages[L]; // Static array to store average temperatures for each location
    for (int loc = 0; loc < L; loc++) {
        float totalTemp = 0.0;
        int count = 0;
        for (int day = 0; day < D; day++) {
            for (int hour = 0; hour < H; hour++) {
                totalTemp += data[day][loc][hour];
                count++;
            }
        }
        locationAverages[loc] = totalTemp / count; // Calculate average
        avgTemps[loc] = locationAverages[loc];
    }
}

// Function to find the day and location with the highest temperature
void findMaxTemperature(float data[D][L][H], int *maxDay, int *maxLocation, float *maxTemp) {
    *maxTemp = data[0][0][0];
    *maxDay = 0;
    *maxLocation = 0;

    for (int day = 0; day < D; day++) {
        for (int loc = 0; loc < L; loc++) {
            for (int hour = 0; hour < H; hour++) {
                if (data[day][loc][hour] > *maxTemp) {
                    *maxTemp = data[day][loc][hour];
                    *maxDay = day;
                    *maxLocation = loc;
                }
            }
        }
    }
}

int main() {
    // 3D array to store temperature readings
    float temperatures[D][L][H] = {
        {
            {20.5, 21.0, 19.8, 22.3}, // Day 1, Location 1
            {25.0, 24.5, 26.1, 23.8}  // Day 1, Location 2
        },
        {
            {21.5, 22.1, 20.2, 23.5}, // Day 2, Location 1
            {26.2, 25.1, 27.0, 24.4}  // Day 2, Location 2
        },
        {
            {22.0, 23.0, 21.1, 24.0}, // Day 3, Location 1
            {27.0, 26.5, 28.2, 25.3}  // Day 3, Location 2
        }
    };

    float avgTemps[L]; // Array to store average daily temperatures for each location
    int maxDay, maxLocation;
    float maxTemp;

    // Calculate average daily temperatures for each location
    calculateAverageDailyTemperature(temperatures, avgTemps);

    // Find the day and location with the highest temperature
    findMaxTemperature(temperatures, &maxDay, &maxLocation, &maxTemp);
    // Display the results
    printf("Average daily temperatures for each location:\n");
    for (int loc = 0; loc < L; loc++) {
        printf("Location %d: %.2f\n", loc + 1, avgTemps[loc]);
    }

    printf("\nDay and location with the highest temperature:\n");
    printf("Day: %d, Location: %d, Temperature: %.2f\n", maxDay + 1, maxLocation + 1, maxTemp);

    return 0;
}
```