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.

Sol:
```
#include <stdio.h>

#define N 5

#define CRITICAL_THRESHOLD 75.0

void logTemperatureData(float temperatures[N][24]);

void analyzeTemperatureData(float temperatures[N][24]);


int main() {

   float temperatures[N][24] = {0};

   logTemperatureData(temperatures);

   analyzeTemperatureData(temperatures);

   return 0;

}

void logTemperatureData(float temperatures[N][24]) {

   printf("Logging temperature data for %d sensors over 24 hours:\n", N);

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

void analyzeTemperatureData(float temperatures[N][24]) {
    printf("\nAnalyzing temperature data:\n");
    static float dailyAverage[N] = {0};
    for (int i = 0; i < N; i++) {
        float sum = 0;
        for (int j = 0; j < 24; j++) {
            sum += temperatures[i][j];
        }
        dailyAverage[i] = sum / 24;
        printf("Sensor %d - Daily Average Temperature: %.2f\n", i + 1,
dailyAverage[i]);
    }

    printf("\nSensors exceeding the critical threshold (%.2f):\n",
CRITICAL_THRESHOLD);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < 24; j++) {
            if (temperatures[i][j] > CRITICAL_THRESHOLD) {
                printf("  Sensor %d exceeded the threshold at Hour %d with
Temperature: %.2f\n", i + 1, j + 1, temperatures[i][j]);
            }
        }
    }
}
```

O/p:

```
Logging temperature data for 1 sensors over 24 hours:
Enter temperature readings for Sensor 1:
  Hour 1: 70
  Hour 2: 80
  Hour 3: 90
  Hour 4: 87
  Hour 5: 89
  Hour 6: 56
  Hour 7: 45
  Hour 8: 65
  Hour 9: 76
  Hour 10: 60
  Hour 11: 89
  Hour 12: 98
  Hour 13: 85
  Hour 14: 34
  Hour 15: 54
  Hour 16: 23
  Hour 17: 54
  Hour 18: 78
  Hour 19: 95
  Hour 20: 93
  Hour 21: 45
  Hour 22: 53
  Hour 23: 31
  Hour 24: 51

Analyzing temperature data:
Sensor 1 - Daily Average Temperature: 66.71

Sensors exceeding the critical threshold (75.00):
  Sensor 1 exceeded the threshold at Hour 2 with Temperature: 80.00
  Sensor 1 exceeded the threshold at Hour 3 with Temperature: 90.00
  Sensor 1 exceeded the threshold at Hour 4 with Temperature: 87.00
  Sensor 1 exceeded the threshold at Hour 5 with Temperature: 89.00
  Sensor 1 exceeded the threshold at Hour 9 with Temperature: 76.00
  Sensor 1 exceeded the threshold at Hour 11 with Temperature: 89.00
  Sensor 1 exceeded the threshold at Hour 12 with Temperature: 98.00
  Sensor 1 exceeded the threshold at Hour 13 with Temperature: 85.00
  Sensor 1 exceeded the threshold at Hour 18 with Temperature: 78.00
  Sensor 1 exceeded the threshold at Hour 19 with Temperature: 95.00
  Sensor 1 exceeded the threshold at Hour 20 with Temperature: 93.00
```

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).

```
Sol: #include <stdio.h>

#define SIZE 8


void displayMatrix(int matrix[SIZE][SIZE]);

void toggleLED(int matrix[SIZE][SIZE], int row, int col);

void countONLEDs(int matrix[SIZE][SIZE]);


int main() {

    int ledMatrix[SIZE][SIZE] = {0};     int row, col, choice;


    while (1) {

        printf("\n1. Toggle an LED\n2. Display Matrix\n3. Count ON LEDs\n4. Exit\nEnter your choice: ");

        scanf("%d", &choice);


        if (choice == 1) {

            printf("Enter row (0-7) and column (0-7): ");
            scanf("%d %d", &row, &col);
            if (row >= 0 && row < SIZE && col >= 0 && col < SIZE) {
                toggleLED(ledMatrix, row, col);
            } else {
                printf("Invalid indices. Please enter values between 0 and 7.\n");
            }
        } else if (choice == 2) {
            displayMatrix(ledMatrix);
        } else if (choice == 3) {
            countONLEDs(ledMatrix);
        } else if (choice == 4) {
            break;
        } else {
            printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}

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

void toggleLED(int matrix[SIZE][SIZE], int row, int col) {
    matrix[row][col] = !matrix[row][col];
    printf("Toggled LED at (%d, %d). New state: %d\n", row, col,
matrix[row][col]);
}

void countONLEDs(int matrix[SIZE][SIZE]) {
    static int onCount = 0;
    onCount = 0;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (matrix[i][j] == 1) {
                onCount++;
            }
        }
    }
    printf("Number of LEDs that are ON: %d\n", onCount);
}
```

O/p:

```
1. Toggle an LED
```

```
2. Display Matrix
3. Count ON LEDs
4. Exit
Enter your choice: 1
Enter row (0-7) and column (0-7): 1 2 3
Toggled LED at (2, 3). New state: 1

1. Toggle an LED
2. Display Matrix
3. Count ON LEDs
4. Exit
Enter your choice: 2

LED Matrix:
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

1. Toggle an LED
2. Display Matrix
```

3. Count ON LEDs

4. Exit

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.

Sol:
```
#include <stdio.h>

#define M 5

#define N 5


void displayGrid(int grid[M][N]);

void moveRobot(int grid[M][N], int *x, int *y, char direction);


int main() {

    int grid[M][N] = {0};

    int x = 0, y = 0;

    char direction;


    grid[x][y] = 1;

    printf("Robot Path Mapping\n");

    printf("Commands: W (UP), S (DOWN), A (LEFT), D (RIGHT), Q (QUIT)\n");
    while (1) {
        displayGrid(grid);
        printf("Enter direction (W/A/S/D) or Q to quit: ");
        scanf(" %c", &direction);

        if (direction == 'Q' || direction == 'q') {
            printf("Exiting program.\n");
            break;
        }

        moveRobot(grid, &x, &y, direction);
    }

    return 0;
}

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

```
}

void moveRobot(int grid[M][N], int *x, int *y, char direction) {
    int newX = *x, newY = *y;

    if (direction == 'W' || direction == 'w') newX--;
    else if (direction == 'S' || direction == 's') newX++;
    else if (direction == 'A' || direction == 'a') newY--;
    else if (direction == 'D' || direction == 'd') newY++;
    else {
        printf("Invalid direction! Use W/A/S/D to move.\n");
        return;
    }

    if (newX >= 0 && newX < M && newY >= 0 && newY < N) {
        *x = newX;
        *y = newY;
        grid[*x][*y] = 1; // Mark the new position as visited
    } else {
        printf("Move out of bounds! Try a different direction.\n");
    }
}
```

O/p:
```
Robot Path Mapping
Commands: W (UP), S (DOWN), A (LEFT), D (RIGHT), Q (QUIT)
```

```
Grid State:

1 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

Enter direction (W/A/S/D) or Q to quit: S


Grid State:

1 0 0 0 0

1 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

Enter direction (W/A/S/D) or Q to quit:
```

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).

Sol:
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define X 3
#define Y 3
#define Z 3
#define CRITICAL_TEMP 50

void populateSensorData(float sensors[X][Y][Z]);
void analyzeCriticalValues(float sensors[X][Y][Z]);
void calculateLayerAverages(float sensors[X][Y][Z]);

int main() {
    float sensors[X][Y][Z];
    srand(time(0));
    printf("Populating sensor data...\n");
    populateSensorData(sensors);

    printf("\nAnalyzing critical values...\n");
    analyzeCriticalValues(sensors);

    printf("\nCalculating layer averages...\n");
    calculateLayerAverages(sensors);
    return 0;
}

void populateSensorData(float sensors[X][Y][Z]) {
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                sensors[i][j][k] = 20 + (rand() % 61);
                printf("Sensor[%d][%d][%d] = %.2f°C\n", i, j, k, sensors[i][j][k]);
            }
        }
    }
}

void analyzeCriticalValues(float sensors[X][Y][Z]) {
    int criticalCount = 0;
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                if (sensors[i][j][k] > CRITICAL_TEMP) {
                    criticalCount++;
                    printf("Critical: Sensor[%d][%d][%d] = %.2f°C\n", i, j, k,
sensors[i][j][k]);
                }
            }
        }
    }

    printf("Total critical readings: %d\n", criticalCount);
}

void calculateLayerAverages(float sensors[X][Y][Z]) {
    static float layerAverages[Z] = {0};
    for (int k = 0; k < Z; k++) {
        float layerSum = 0;

        for (int i = 0; i < X; i++) {
            for (int j = 0; j < Y; j++) {
                layerSum += sensors[i][j][k];
            }
        }

        layerAverages[k] = layerSum / (X * Y);
        printf("Average for layer %d: %.2f°C\n", k, layerAverages[k]);
    }
}
```

```
O/p: Populating sensor data...
Sensor[0][0][0] = 79.00°C
Sensor[0][0][1] = 61.00°C
Sensor[0][0][2] = 38.00°C
Sensor[0][1][0] = 68.00°C
```

```
Sensor[0][1][1] = 45.00°C
Sensor[0][1][2] = 29.00°C
Sensor[0][2][0] = 75.00°C
Sensor[0][2][1] = 69.00°C
Sensor[0][2][2] = 64.00°C
Sensor[1][0][0] = 28.00°C
Sensor[1][0][1] = 48.00°C
Sensor[1][0][2] = 80.00°C
Sensor[1][1][0] = 51.00°C
Sensor[1][1][1] = 21.00°C
Sensor[1][1][2] = 42.00°C
Sensor[1][2][0] = 40.00°C
Sensor[1][2][1] = 47.00°C
Sensor[1][2][2] = 51.00°C
Sensor[2][0][0] = 30.00°C
Sensor[2][0][1] = 51.00°C
Sensor[2][0][2] = 39.00°C
Sensor[2][1][0] = 20.00°C
Sensor[2][1][1] = 21.00°C
Sensor[2][1][2] = 20.00°C
Sensor[2][2][0] = 69.00°C
Sensor[2][2][1] = 64.00°C
Sensor[2][2][2] = 39.00°C

Analyzing critical values...
```

```
Critical: Sensor[0][0][0] = 79.00°C
Critical: Sensor[0][0][1] = 61.00°C
Critical: Sensor[0][1][0] = 68.00°C
Critical: Sensor[0][2][0] = 75.00°C
Critical: Sensor[0][2][1] = 69.00°C
Critical: Sensor[0][2][2] = 64.00°C
Critical: Sensor[1][0][2] = 80.00°C
Critical: Sensor[1][1][0] = 51.00°C
Critical: Sensor[1][2][2] = 51.00°C
Critical: Sensor[2][0][1] = 51.00°C
Critical: Sensor[2][2][0] = 69.00°C
Critical: Sensor[2][2][1] = 64.00°C
Total critical readings: 12

Calculating layer averages...
Average for layer 0: 51.11°C
Average for layer 1: 47.44°C
Average for layer 2: 44.67°C
```

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.

Sol:
```
#include <stdio.h>
#include <stdlib.h>

#define H 5
#define W 5
#define THRESHOLD 100

void applyEdgeDetection(int input[H][W], int output[H][W]);
void displayMatrix(const char *title, int matrix[H][W]);

int main() {

    int image[H][W] = {
        {10, 20, 30, 40, 50},
        {20, 30, 40, 50, 60},
        {30, 40, 50, 60, 70},
        {40, 50, 60, 70, 80},
        {50, 60, 70, 80, 90}
    };

    static int edges[H][W] = {0};

    printf("Image Processing: Edge Detection\n");
    displayMatrix("Original Image", image);
    applyEdgeDetection(image, edges);
    displayMatrix("Edge Detected Image", edges);

    return 0;
}

void applyEdgeDetection(int input[H][W], int output[H][W]) {
    int gx, gy;

    for (int i = 1; i < H - 1; i++) {
        for (int j = 1; j < W - 1; j++) {

            gx = (input[i - 1][j + 1] + 2 * input[i][j + 1] + input[i + 1][j + 1]) -
                 (input[i - 1][j - 1] + 2 * input[i][j - 1] + input[i + 1][j - 1]);

            gy = (input[i + 1][j - 1] + 2 * input[i + 1][j] + input[i + 1][j + 1]) -
                 (input[i - 1][j - 1] + 2 * input[i - 1][j] + input[i - 1][j + 1]);
            int magnitude = abs(gx) + abs(gy);

            output[i][j] = (magnitude > THRESHOLD) ? 255 : 0;
        }
    }
}
```

```
void displayMatrix(const char *title, int matrix[H][W]) {
    printf("\n%s:\n", title);
    for (int i = 0; i < H; i++) {
        for (int j = 0; j < W; j++) {
            printf("%3d ", matrix[i][j]);
        }
        printf("\n");
    }
}
```

O/p:

```
Image Processing: Edge Detection

Original Image:
 10  20  30  40  50
 20  30  40  50  60
 30  40  50  60  70
 40  50  60  70  80
 50  60  70  80  90

Edge Detected Image:
  0   0   0   0   0
  0 255 255 255   0
  0 255 255 255   0
  0 255 255 255   0
  0   0   0   0   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).

Sol:
```c
#include <stdio.h>

#define ROADS 4

#define LIGHTS 3


void displayTrafficLights(int trafficLights[ROADS][LIGHTS]) {

    for (int i = 0; i < ROADS; i++) {

        printf("Road %d: Red=%d, Yellow=%d, Green=%d\n", i + 1, 
trafficLights[i][0], trafficLights[i][1], trafficLights[i][2]);

    }

    printf("\n");

}


int main() {

    int trafficLights[ROADS][LIGHTS] = {0};

    static int stateCycle = 0;
    for (int time = 0; time < 6; time++) {
        printf("Time %d:\n", time + 1);
        for (int i = 0; i < ROADS; i++) {
            trafficLights[i][0] = 0;
            trafficLights[i][1] = 0;
            trafficLights[i][2] = 0;
        }
        for (int i = 0; i < ROADS; i++) {
            if (stateCycle == 0) trafficLights[i][0] = 1;
            else if (stateCycle == 1) trafficLights[i][1] = 1;
            else trafficLights[i][2] = 1;
        }

        displayTrafficLights(trafficLights);
        stateCycle = (stateCycle + 1) % 3;
    }

    return 0;
}
```

O/p:

```
Time 1:
Road 1: Red=1, Yellow=0, Green=0
Road 2: Red=1, Yellow=0, Green=0
Road 3: Red=1, Yellow=0, Green=0
Road 4: Red=1, Yellow=0, Green=0
```

```
Time 2:
Road 1: Red=0, Yellow=1, Green=0
Road 2: Red=0, Yellow=1, Green=0
Road 3: Red=0, Yellow=1, Green=0
Road 4: Red=0, Yellow=1, Green=0

Time 3:
Road 1: Red=0, Yellow=0, Green=1
Road 2: Red=0, Yellow=0, Green=1
Road 3: Red=0, Yellow=0, Green=1
Road 4: Red=0, Yellow=0, Green=1

Time 4:
Road 1: Red=1, Yellow=0, Green=0
Road 2: Red=1, Yellow=0, Green=0
Road 3: Red=1, Yellow=0, Green=0
Road 4: Red=1, Yellow=0, Green=0

Time 5:
Road 1: Red=0, Yellow=1, Green=0
Road 2: Red=0, Yellow=1, Green=0
Road 3: Red=0, Yellow=1, Green=0
Road 4: Red=0, Yellow=1, Green=0
```

```
Time 6:
Road 1: Red=0, Yellow=0, Green=1
Road 2: Red=0, Yellow=0, Green=1
Road 3: Red=0, Yellow=0, Green=1
Road 4: Red=0, Yellow=0, Green=1
```

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.

Sol:
```c
#include <stdio.h>
#define SIZE 4
void displayLEDCube(int cube[SIZE][SIZE][SIZE]) {
    for (int z = 0; z < SIZE; z++) {
        printf("Layer %d:\n", z + 1);
        for (int y = 0; y < SIZE; y++) {
            for (int x = 0; x < SIZE; x++) {
                printf("%d ", cube[z][y][x]);
            }
            printf("\n");
        }
        printf("\n");
    }
}

int main() {
    int cube[SIZE][SIZE][SIZE] = {0};
    static int frameCounter = 0;
    for (int frame = 0; frame < 4; frame++) {
        printf("Frame %d:\n", frame + 1);
        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;
                }
            }
        }
        for (int i = 0; i < SIZE; i++) {
            cube[frame][i][i] = 1;
        }

        displayLEDCube(cube);
        frameCounter++;
    }

    return 0;
}
```

```
O/p: Frame 1:
Layer 1:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

Layer 2:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Layer 3:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Layer 4:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
```

Frame 2:

Layer 1:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 2:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

Layer 3:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 4:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Frame 3:

Layer 1:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 2:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 3:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

Layer 4:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Frame 4:

Layer 1:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 2:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 3:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 4:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

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.

Sol:
```c
#include <stdio.h>
#define PRODUCTS 2
#define ROWS 3
#define COLS 3
void displayInventory(int inventory[PRODUCTS][ROWS][COLS], int
total[PRODUCTS]);
void updateInventory(int inventory[PRODUCTS][ROWS][COLS], int
*lowStockCount);

int main() {
    int inventory[PRODUCTS][ROWS][COLS] = {0};
    int totalInventory[PRODUCTS] = {0};
    int lowStockCount = 0;

    printf("Warehouse Inventory Tracking\n");
    updateInventory(inventory, &lowStockCount);
    displayInventory(inventory, totalInventory);
    printf("\nLow-stock locations: %d\n", lowStockCount);


    return 0;
}


void displayInventory(int inventory[PRODUCTS][ROWS][COLS], int
total[PRODUCTS]) {
    for (int p = 0; p < PRODUCTS; p++) {
        printf("Product %d Inventory:\n", p + 1);
        total[p] = 0;
        for (int r = 0; r < ROWS; r++) {
            for (int c = 0; c < COLS; c++) {
                printf("%d ", inventory[p][r][c]);
                total[p] += inventory[p][r][c];
            }
            printf("\n");
        }
        printf("Total Inventory for Product %d: %d\n", p + 1, total[p]);
    }
}


void updateInventory(int inventory[PRODUCTS][ROWS][COLS], int
*lowStockCount) {
    for (int p = 0; p < PRODUCTS; p++) {
        for (int r = 0; r < ROWS; r++) {
            for (int c = 0; c < COLS; c++) {
                inventory[p][r][c] = (p + 1) * (r + 1) * (c + 1);
                if (inventory[p][r][c] < 5) {
                    (*lowStockCount)++;
                }
            }
        }
    }
}
```

```
O/p: Warehouse Inventory Tracking
Product 1 Inventory:
1 2 3
2 4 6
3 6 9
Total Inventory for Product 1: 36
Product 2 Inventory:
2 4 6
4 8 12
6 12 18
Total Inventory for Product 2: 72

Low-stock locations: 9
```

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.

Sol:
```
#include <stdio.h>
#define X 3
#define Y 3
#define Z 3
void applyFilter(int signal[X][Y][Z], int result[X][Y][Z]);

int main() {
    int signal[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}}
    };
    int result[X][Y][Z] = {0};
    printf("Original Signal:\n");
    for (int x = 0; x < X; x++) {
        for (int y = 0; y < Y; y++) {
            for (int z = 0; z < Z; z++) {
                printf("%d ", signal[x][y][z]);
            }
            printf("\n");
        }
        printf("\n");
    }

    applyFilter(signal, result);

    printf("Filtered Signal:\n");
    for (int x = 0; x < X; x++) {
        for (int y = 0; y < Y; y++) {
            for (int z = 0; z < Z; z++) {
                printf("%d ", result[x][y][z]);
            }
            printf("\n");
        }
        printf("\n");
    }

    return 0;
}

void applyFilter(int signal[X][Y][Z], int result[X][Y][Z]) {
    for (int x = 0; x < X; x++) {
        for (int y = 0; y < Y; y++) {
            for (int z = 0; z < Z; z++) {
                int sum = 0, count = 0;
                for (int dx = -1; dx <= 1; dx++) {
                    for (int dy = -1; dy <= 1; dy++) {
                        for (int dz = -1; dz <= 1; dz++) {
                            int nx = x + dx, ny = y + dy, nz = z + dz;
                            if (nx >= 0 && nx < X && ny >= 0 && ny < Y && nz >= 0 &&
nz < Z) {
                                sum += signal[nx][ny][nz];
                                count++;
                            }
                        }
                    }
                }
                result[x][y][z] = sum / count;            }
        }
    }
}
```

O/p: Original Signal:

```
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

Filtered Signal:
7 8 8
9 9 10
10 11 11

12 12 13
13 14 14
15 15 16

16 17 17
18 18 19
19 20 20
```

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.

Sol: #include <stdio.h>

```
#define DAYS 2
#define LOCATIONS 3
#define HOURS 4
void analyzeWeather(float data[DAYS][LOCATIONS][HOURS]);

int main() {
    float data[DAYS][LOCATIONS][HOURS] = {
        {{30.1, 32.2, 31.8, 30.5}, {29.5, 28.9, 30.0, 31.2}, {33.0, 34.1, 32.7, 31.9}},
        {{29.8, 30.0, 29.5, 30.1}, {28.7, 27.9, 29.0, 29.8}, {30.5, 31.0, 30.2, 32.0}}
    };

    analyzeWeather(data);

    return 0;
}

void analyzeWeather(float data[DAYS][LOCATIONS][HOURS]) {
    float dailyAvg[LOCATIONS] = {0};
    float maxTemp = -1.0;
    int maxDay = 0, maxLoc = 0;

    for (int l = 0; l < LOCATIONS; l++) {
        dailyAvg[l] = 0;
        for (int d = 0; d < DAYS; d++) {
            float dailySum = 0;
            for (int h = 0; h < HOURS; h++) {
                dailySum += data[d][l][h];
                if (data[d][l][h] > maxTemp) {
                    maxTemp = data[d][l][h];
                    maxDay = d;
                    maxLoc = l;
                }
            }
            dailyAvg[l] += dailySum / HOURS;
        }
        dailyAvg[l] /= DAYS;
    }
    printf("Average Daily Temperature by Location:\n");
    for (int l = 0; l < LOCATIONS; l++) {
        printf("Location %d: %.2f°C\n", l + 1, dailyAvg[l]);
    }
    printf("\nHighest Temperature: %.2f°C on Day %d at Location %d\n",
maxTemp, maxDay + 1, maxLoc + 1);
}
```

O/p:
```
Average Daily Temperature by Location:
Location 1: 30.50°C
Location 2: 29.38°C
Location 3: 31.92°C

Highest Temperature: 34.10°C on Day 1 at Location 3
```