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>
#include <stdlib.h>
#include <time.h>

#define N 5
#define CT 80

int main()
{
   const int h = 24;
   float t[N][h], s;
   static float a[N];
   int i, j;
   srand(time(0));
   for ( i = 0; i < N; i++)
```

```
    {
        s=0.0;
        for ( j = 0; j < h; j++)
        {
            t[i][j] = (rand() % 500) / 10.0;
            s += t[i][j];
        }
        a[i] = s / h;
    }
    printf("Temperature readings (°C):\n");
    for (i = 0; i < N; i++)
    {
        printf("Sensor %d: ", i + 1);
        for ( j = 0; j < h; j++)
            printf("%.2f ", t[i][j]);
        printf("\n");
    }
    printf("\nDaily average temperatures (°C):\n");
    for (i = 0; i < N; i++)
    {
        printf("Sensor %d: %.2f", i + 1, a[i]);
        if (a[i] > CT)
            printf(" Exceeds critical threshold\n");
        printf("\n");
    }
    return 0;
}
```

O/P:

Temperature readings (°C):

Sensor 1: 33.20 32.60 48.60 43.30 36.50 7.90 48.90 12.30 46.70 6.40 32.80 7.10 21.20 37.50 24.70 29.60 38.30 26.10 17.20 4.20 38.40 32.70 11.30 23.50

Sensor 2: 39.30 31.90 13.20 45.70 0.00 24.20 45.50 42.40 20.70 28.10 27.40 11.00 39.00 30.20 23.70 42.60 1.50 33.20 15.80 15.90 12.60 11.40 31.50 18.50

Sensor 3: 18.00 10.70 0.00 5.60 35.80 28.60 40.00 44.30 11.80 45.60 25.60 28.20 37.70 33.40 8.40 20.00 30.10 44.10 38.90 27.50 23.20 21.50 38.60 46.50

Sensor 4: 3.40 22.40 30.70 38.80 13.20 49.70 47.10 7.30 32.50 29.10 32.00 43.80 14.10 24.30 2.90 35.10 21.20 6.90 31.10 39.30 4.40 37.80 0.80 36.60

Sensor 5: 9.00 7.10 0.00 31.20 7.50 29.90 29.60 12.20 0.10 33.20 49.30 5.90 5.50 16.00 33.70 38.80 15.30 15.70 9.70 13.10 43.30 45.10 47.00 36.10

Daily average temperatures (°C):

Sensor 1: 27.54

Sensor 2: 25.22

Sensor 3: 27.67

Sensor 4: 25.19

Sensor 5: 22.26

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 S 8

int main()
{
   int l[S][S];
   static int count = 0;
   int r, c, i, j;
   char ch;
   for (i = 0; i < S; i++)
   {
      for (j = 0; j < S; j++)
         l[i][j] = 0;
   }
   while (1)
   {
      printf("State of LED matrix:\n");
      for (i = 0; i < S; i++)
      {
         for (j = 0; j < S; j++)
            printf("%d ", l[i][j]);
         printf("\n");
```

```
    }
    printf("Current ON LEDs: %d\n", count);
    printf("Enter command - t to toggle, q to quit: ");
    scanf(" %c", &ch);
    if (ch == 'q')
        break;
    else if (ch == 't')
    {
        printf("Enter row and column to toggle (0-7): ");
        scanf("%d %d", &r, &c);
        if (r >= 0 && r < S && c >= 0 && c < S)
        {
            l[r][c] = !l[r][c];
            count += (l[r][c] == 1) ? 1 : -1;
        }
        else
            printf("Invalid row or column\n");
    }
    else
        printf("Invalid command\n");
}

printf("Final state of the matrix:\n");
for (i = 0; i < S; i++)
{
    for (j = 0; j < S; j++)
        printf("%d ", l[i][j]);
```

```
            printf("\n");
        }
        printf("Final ON LEDs: %d\n", count);
        return 0;
    }
```

O/P:

State of LED matrix:

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 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

Current ON LEDs: 0

Enter command - t to toggle, q to quit: t

Enter row and column to toggle (0-7): 4 6

State of LED matrix:

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 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

Current ON LEDs: 1

Enter command - t to toggle, q to quit: t

Enter row and column to toggle (0-7): 8 2

Invalid row or column

State of LED matrix:

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 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

Current ON LEDs: 1

Enter command - t to toggle, q to quit: t

Enter row and column to toggle (0-7): 2 3

State of 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 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

Current ON LEDs: 2

Enter command - t to toggle, q to quit: q

Final state of the 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 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

Final ON LEDs: 2

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 3
#define N 3
```

```
int main()
{
    int g[M][N];
    int x = 0, y = 0, i, j;
    char d;
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
            g[i][j] = 0;
    }
    g[x][y] = 1;
    printf("Initial Position: (%d, %d)\n", x, y);
    while (1)
    {
        printf("Grid State:\n");
        for (i = 0; i < M; i++)
        {
            for (j = 0; j < N; j++)
                printf("%d ", g[i][j]);
            printf("\n");
        }
            printf("Enter direction - U=UP, D=DOWN, L=LEFT,
R=RIGHT, Q=QUIT): ");
        scanf(" %c", &d);
        if (d == 'Q' || d == 'q')
            break;
        if (d == 'U' || d == 'u')
        {
```

```
    if (x > 0)
        x--;
    else
        printf("Cannot move UP\n");
}
else if (d == 'D' || d == 'd')
{
    if (x < M - 1)
        x++;
    else
        printf("Cannot move DOWN\n");
}
else if (d == 'L' || d == 'l')
{
    if (y > 0)
        y--;
    else
        printf("Cannot move LEFT\n");
}
else if (d == 'R' || d == 'r')
{
    if (y < N - 1)
        y++;
    else
        printf("Cannot move RIGHT\n");
}
else
```

```
            printf("Invalid direction\n");
        g[x][y] = 1;
        printf("Robot moved to position: (%d, %d)\n", x, y);
    }
    printf("Final state of the grid:\n");
    printf("Grid State:\n");
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
            printf("%d ", g[i][j]);
        printf("\n");
    }
    return 0;
}
```

O/P:

```
Initial Position: (0, 0)
Grid State:
1 0 0
0 0 0
0 0 0
Enter direction - U=UP, D=DOWN, L=LEFT, R=RIGHT, Q=QUIT): u
Cannot move UP
Robot moved to position: (0, 0)
Grid State:
1 0 0
0 0 0
```

0 0 0

Enter direction - U=UP, D=DOWN, L=LEFT, R=RIGHT, Q=QUIT): d

Robot moved to position: (1, 0)

Grid State:

1 0 0

1 0 0

0 0 0

Enter direction - U=UP, D=DOWN, L=LEFT, R=RIGHT, Q=QUIT): r

Robot moved to position: (1, 1)

Grid State:

1 0 0

1 1 0

0 0 0

Enter direction - U=UP, D=DOWN, L=LEFT, R=RIGHT, Q=QUIT): d

Robot moved to position: (2, 1)

Grid State:

1 0 0

1 1 0

0 1 0

Enter direction - U=UP, D=DOWN, L=LEFT, R=RIGHT, Q=QUIT): l

Robot moved to position: (2, 0)

Grid State:

1 0 0

1 1 0

1 1 0

Enter direction - U=UP, D=DOWN, L=LEFT, R=RIGHT, Q=QUIT): q

Final state of the grid:

Grid State:

1 0 0

1 1 0

1 1 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 2
#define CT 70.0

int main()
{
   float s[X][Y][Z], s1=0.0, r;
   static float a[Z];
```

```
int i, j, k, c=0, sc=0;
printf("Sensor data\n");
for (i = 0; i < X; i++)
{
   for (j = 0; j < Y; j++)
   {
      for (k = 0; k < Z; k++)
      {
         s[i][j][k] = 20.0 + i * 10.0 + j * 5.0 + k * 2.0;
         printf("Sensor[%d][%d][%d] = %.2f\n", i, j, k, s[i][j][k]);
      }
   }
}
for (k = 0; k < Z; k++)
{
   for (i = 0; i < X; i++)
   {
      for (j = 0; j < Y; j++)
      {
         r = s[i][j][k];
         s1 += r;
         sc++;
         if (r > CT)
         {
            c++;
                printf("Critical sensor at [%d][%d][%d] with value:
%.2f\n", i, j, k, r);
         }
```

```
            }
        }
        a[k] = s1 / sc;
        printf("Layer %d - Average Reading: %.2f\n", k, a[k]);
    }
    printf("\nTotal critical sensors: %d\n", c);
    return 0;
}
```

O/P:

```
Sensor data
Sensor[0][0][0] = 20.00
Sensor[0][0][1] = 22.00
Sensor[0][1][0] = 25.00
Sensor[0][1][1] = 27.00
Sensor[0][2][0] = 30.00
Sensor[0][2][1] = 32.00
Sensor[1][0][0] = 30.00
Sensor[1][0][1] = 32.00
Sensor[1][1][0] = 35.00
Sensor[1][1][1] = 37.00
Sensor[1][2][0] = 40.00
Sensor[1][2][1] = 42.00
Sensor[2][0][0] = 40.00
Sensor[2][0][1] = 42.00
Sensor[2][1][0] = 45.00
Sensor[2][1][1] = 47.00
```

Sensor[2][2][0] = 50.00

Sensor[2][2][1] = 52.00

Layer 0 - Average Reading: 35.00

Layer 1 - Average Reading: 36.00

Total critical sensors: 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>
#include <math.h>

#define H 5
#define W 5
#define ET 100

void main()
{
```

```
int inp[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}    };
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}    };
static int out[H][W];
int i, j, x, y, m;
printf("Input Image:\n");
for (i = 0; i < H; i++)
{
   for (j = 0; j < W; j++)
      printf("%4d", inp[i][j]);
   printf("\n");
}
for (i = 1; i < H - 1; i++)
{
   for (j = 1; j < W - 1; j++)
   {
      int gX = 0, gY = 0;
      for (x = -1; x <= 1; x++)
      {
```

```
                    for (y = -1; y <= 1; y++)
                    {
                        gX += inp[i + x][j + y] * Gx[x + 1][y + 1];
                        gY += inp[i + x][j + y] * Gy[x + 1][y + 1];
                    }
                }
                m = (int)sqrt(gX * gX + gY * gY);
                out[i][j] = (m > ET) ? 255 : 0;
            }
        }
        printf("\nEdge Detected Image:\n");
        for (i = 0; i < H; i++)
        {
            for (j = 0; j < W; j++)
                printf("%4d", out[i][j]);
            printf("\n");
        }
    }
```

O/P:

Input Image:

```
  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
```

Edge Detected Image:

0 0 0 0 0

0 255 255 255 0

0 255 0 255 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).

```
#include <stdio.h>
#include <unistd.h>

#define R 0
#define Y 1
#define G 2
#define R1 4
#define L 3
```

```
void fun(int t[R1][L], int *c)
{
   int k;
   k= *c % R1;
   t[k][G] = 0;
   t[k][Y] = 1;
   sleep(1);
   t[k][Y] = 0;
   t[k][R] = 1;
   sleep(1);
   t[k][R] = 0;
   t[k][G] = 1;
   (*c)++;
}

void main()
{
   int t[R1][L], i, j, t1;
   static int c = 0;
   for (i = 0; i < R1; i++)
   {
      t[i][R] = 1;
      t[i][Y] = 0;
      t[i][G] = 0;
   }
   while(1)
   {
```

```
            printf("\nCycle %d:\n", c);
            printf("State of Traffic Lights (RED, YELLOW, GREEN):\n");
            for (i = 0; i < R1; i++)
            {
               printf("Road %d: ", i + 1);
               for (j = 0; j < L; j++)
                  printf("%d ", t[i][j]);
               printf("\n");
            }
            fun(t, &c);
         }
      }
```

O/P:

Cycle 0:

State of Traffic Lights (RED, YELLOW, GREEN):

Road 1: 1 0 0

Road 2: 1 0 0

Road 3: 1 0 0

Road 4: 1 0 0

Cycle 1:

State of Traffic Lights (RED, YELLOW, GREEN):

Road 1: 0 0 1

Road 2: 1 0 0

Road 3: 1 0 0

Road 4: 1 0 0

Cycle 2:

State of Traffic Lights (RED, YELLOW, GREEN):

Road 1: 0 0 1

Road 2: 0 0 1

Road 3: 1 0 0

Road 4: 1 0 0

Cycle 3:

State of Traffic Lights (RED, YELLOW, GREEN):

Road 1: 0 0 1

Road 2: 0 0 1

Road 3: 0 0 1

Road 4: 1 0 0

Cycle 4:

State of Traffic Lights (RED, YELLOW, GREEN):

Road 1: 0 0 1

Road 2: 0 0 1

Road 3: 0 0 1

Road 4: 0 0 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.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define S 4
#define D 500

int main()
{
   printf("Animation on a 4x4x4 LED cube\n");
   static int c[S][S][S];
   static int f = 0;
   int x, y, z, i;
   while (f < 10)
   {
      for (z = 0; z < S; z++)
      {
         for (y = 0; y < S; y++)
         {
            for (x = 0; x < S; x++)
            c[z][y][x] = 0;
         }
      }
```

```
if (f % 3 == 0)
{
    for (i = 0; i < S; i++)
        c[i][i][i] = 1;
}
else if (f % 3 == 1)
{
    for (y = 0; y < S; y++)
    {
        for (x = 0; x < S; x++)
            c[S / 2][y][x] = 1;
    }
}
else
{
    for (z = 0; z < S; z++)
    {
        for (x = 0; x < S; x++)
            c[z][x][(x + f) % S] = 1;
    }
}
for (z = 0; z < S; z++)
{
    printf("Layer %d:\n", z);
    for (y = 0; y < S; y++)
    {
        for (x = 0; x < S; x++)
```

```
                printf("%d ", c[z][y][x]);
            printf("\n");
        }
        printf("\n");
    }
    printf("-------------------------\n");
    usleep(D);
    f++;
  }
  return 0;
}
```

O/P:

Animation on a 4x4x4 LED cube

Layer 0:

1 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 1:

0 0 0 0

0 1 0 0

0 0 0 0

0 0 0 0

Layer 2:

0 0 0 0

0 0 0 0

0 0 1 0

0 0 0 0

Layer 3:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 1

Layer 0:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 1:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 2:

1 1 1 1

1 1 1 1

1 1 1 1

1 1 1 1

Layer 3:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Layer 0:

0 0 1 0

0 0 0 1

1 0 0 0

0 1 0 0

Layer 1:

0 0 1 0

0 0 0 1

1 0 0 0

0 1 0 0

Layer 2:

0 0 1 0

0 0 0 1

1 0 0 0

0 1 0 0

Layer 3:

0 0 1 0

0 0 0 1

1 0 0 0

0 1 0 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 3
#define R 4
#define C 5
#define T 10

void update(int inv[P][R][C], int p, int r, int c, int s)
{
   static int t[P] = {0};
```

```
    if (p < P && r < R && c < C)
    {
        inv[p][r][c] += s;
        t[p] += s;
         printf("Updated inventory for Product %d at (%d, %d): %d
units (Total: %d units)\n", p, r, c, inv[p][r][c], t[p]);
    }
    else
        printf("Invalid product or location\n");
}

int main()
{
    int inv[P][R][C] = {0}, p, r, c;
    update(inv, 0, 2, 1, 40);
    update(inv, 1, 0, 3, 60);
    update(inv, 2, 2, 3, 8);
    printf("\nCurrent Inventory Levels:\n");
    for (p = 0; p < P; p++)
    {
        printf("Product %d:\n", p);
        for (r = 0; r < R; r++)
        {
            for (c = 0; c < C; c++)
                printf("%4d ", inv[p][r][c]);
            printf("\n");
        }
        printf("\n");
```

```
    }
    printf("\nChecking for low stock levels:\n");
    for (p = 0; p < P; p++)
    {
        for (r = 0; r < R; r++)
        {
            for (c = 0; c < C; c++)
            {
                if (inv[p][r][c] > 0 && inv[p][r][c] < T)
                    printf("Low stock for product %d at (%d, %d): %d
units\n", p, r, c, inv[p][r][c]);
            }
        }
    }
    return 0;
}
```

O/P:

```
Updated inventory for Product 0 at (2, 1): 40 units (Total: 40 units)
Updated inventory for Product 1 at (0, 3): 60 units (Total: 60 units)
Updated inventory for Product 2 at (2, 3): 8 units (Total: 8 units)

Current Inventory Levels:
Product 0:
  0   0   0   0   0
  0   0   0   0   0
  0  40   0   0   0
  0   0   0   0   0
```

Product 1:

0 0 0 60 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

Product 2:

0 0 0 0 0

0 0 0 0 0

0 0 0 8 0

0 0 0 0 0

Checking for low stock levels:

Low stock for product 2 at (2, 3): 8 units

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

// Function to print a 3D matrix
void printMatrix(int matrix[X][Y][Z]) {
    for (int i = 0; i < X; i++) {
        printf("Layer %d:\n", i+1);
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("%d ", matrix[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
}

int main()
{
    int s[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}}
    };
    int f[X][Y][Z] = {0}, i, j, k, sum, c, di, dj, dk, ni, nj, nk;
    for (i = 0; i < X; i++)
    {
        for (j = 0; j < Y; j++)
        {
            for (k = 0; k < Z; k++)
            {
                sum = 0;
                c = 0;
                for (di = -1; di <= 1; di++)
                {
                    for (dj = -1; dj <= 1; dj++)
                    {
                        for (dk = -1; dk <= 1; dk++)
                        {
                            ni = i + di;
                            nj = j + dj;
                            nk = k + dk;
                            if (ni >= 0 && ni < X && nj >= 0 && nj < Y &&
nk >= 0 && nk < Z)
                            {
                                sum += s[ni][nj][nk];
                                c++;
                            }
                        }
```

```
            }
        }
        f[i][j][k] = sum / c;
    }
  }
}
printf("Original Signal:\n");
for (i = 0; i < X; i++)
{
  printf("Layer %d:\n", i+1);
  for (j = 0; j < Y; j++)
  {
    for (k = 0; k < Z; k++)
      printf("%d ", s[i][j][k]);
    printf("\n");
  }
  printf("\n");
}
printf("Filtered Signal:\n");
for (i = 0; i < X; i++)
{
  printf("Layer %d:\n", i+1);
  for (j = 0; j < Y; j++)
  {
    for (k = 0; k < Z; k++)
      printf("%d ", f[i][j][k]);
    printf("\n");
```

```
            }
            printf("\n");
        }
        return 0;
    }
```

O/P:

Original Signal:

Layer 1:

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

Layer 2:

17 18 19 20

21 22 23 24

25 26 27 28

29 30 31 32

Layer 3:

33 34 35 36

37 38 39 40

41 42 43 44

45 46 47 48

Layer 4:

49 50 51 52

53 54 55 56

57 58 59 60

61 62 63 64

Filtered Signal:

Layer 1:

11 12 13 13

13 14 15 15

17 18 19 19

19 20 21 21

Layer 2:

19 20 21 21

21 22 23 23

25 26 27 27

27 28 29 29

Layer 3:

35 36 37 37

37 38 39 39

41 42 43 43

43 44 45 45

Layer 4:

43 44 45 45

45 46 47 47

49 50 51 51

51 52 53 53

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
#define L 4
#define H 24

void temp(int w[D][L][H], int *mT, int *mTDay, float *max)
{
   *max = -1000;
   int d, l, h;
   for (d = 0; d < D; d++)
   {
      for (l = 0; l < L; l++)
```

```
        {
            for (h = 0; h < H; h++)
            {
                if (w[d][l][h] > *max)
                {
                    *max = w[d][l][h];
                    *mT = l;
                    *mTDay = d;
                }
            }
        }
    }
}

void average(int w[D][L][H], float avg[L])
{
    int d, l, h;
    for (l = 0; l < L; l++)
    {
        float t = 0;
        for (d = 0; d < D; d++)
        {
            for (h = 0; h < H; h++)
                t += w[d][l][h];
        }
        avg[l] = t / (D * H);
    }
```

```
}

int main()
{
   int w[D][L][H] = {
      {
         {20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43},
         {25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48},
         {30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53},
         {22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 43, 44, 45}
      },
      {
         {21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44},
         {26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49},
         {31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
47, 48, 49, 50, 51, 52, 53, 54},
         {23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
39, 40, 41, 42, 43, 44, 45, 46}
      },
      {
         {22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 43, 44, 45},
         {27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 47, 48, 49, 50},
```

```
        {32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
48, 49, 50, 51, 52, 53, 54, 55},
        {24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47}
    }
};
float avg[L] = {0};
int mT = 0, mTDay = 0, d, l, h;
float max = 0;
printf("Weather Data:\n");
for (d = 0; d < D; d++)
{
    printf("Day %d:\n", d + 1);
    for (l = 0; l < L; l++)
    {
        printf("Location %d: ", l + 1);
        for (h = 0; h < H; h++)
            printf("%d ", w[d][l][h]);
        printf("\n");
    }
    printf("\n");
}
average(w, avg);
printf("Average Daily Temperatures for Each Location:\n");
for (l = 0; l < L; l++)
    printf("Location %d: %.2f\n", l + 1, avg[l]);
temp(w, &mT, &mTDay, &max);
```

```
    printf("\nLocation %d on day %d had the highest temperature: %.2f\n", mT + 1, mTDay + 1, max);

    return 0;

}
```

O/P:

Weather Data:

Day 1:

Location 1: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43

Location 2: 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48

Location 3: 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53

Location 4: 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45

Day 2:

Location 1: 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44

Location 2: 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49

Location 3: 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54

Location 4: 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46

Day 3:

Location 1: 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45

Location 2: 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Location 3: 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

Location 4: 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

Average Daily Temperatures for Each Location:

Location 1: 32.50

Location 2: 37.50

Location 3: 42.50

Location 4: 34.50

Location 3 on day 3 had the highest temperature: 55.00