----------weekend tasks----------

1(Temperature monitoring by various sensors)

```
#include<stdio.h>
#include<stdlib.h>
#define sensors 3
#define critical_temp 40

void temperature_exceed(int matrix[sensors][3]){
   for(int i=0;i<sensors;i++){
      for(int j=0;j<3;j++){
         if(matrix[i][j]>critical_temp){
            printf("ALERT:The temperature of sensor %d at hour %d is critical\n",i+1,j+1);
         }
      }
   }
}

void avg_temp(int matrix[sensors][3]){
   static float daily_avg[sensors];
   for(int i=0;i<sensors;i++){
      float sum=0.0;
      for(int j=0;j<3;j++){
         sum+=matrix[i][j];
      }
      daily_avg[i]=sum/3.0;
   }
   for(int i=0;i<sensors;i++){
      printf("The daily avaerage of sensor %d = %.2f\n",i+1,daily_avg[i]);
   }
```

```
}

int main(){
   int temperature[sensors][3];
   for(int i=0;i<sensors;i++){
      for(int j=0;j<3;j++){
         printf("Enter the data %d for sensor %d : ",j+1,i+1);
         scanf("%d",&temperature[i][j]);
      }
   }
   avg_temp(temperature);
   temperature_exceed(temperature);
   return 0;
}
```

2 (LED toggling)

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#define m 4
#define n 4
#define size 4
#define ON 1
#define OFF 0
void toggle_bits(int matrix[m][n],int row,int col,int *count){
   if(row>=0 && row<size && col>=0 && col<size){
      if(matrix[row][col]==OFF){
         matrix[row][col]=ON;
         (*count)+=1;
```

```
        }
        else if(matrix[row][col]==ON){
            matrix[row][col]=OFF;
            (*count)-=1;
        }
    }else{
        printf("Invalid column\n");
    }
}
void print_matrix(int matrix[m][n]){
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            printf("%d",matrix[i][j]);
        }
        printf("\n");
    }
}
int main(){
    int led[m][n]={{0}};
    print_matrix(led);
    bool is_on=true;
    static int count=0;
    while(is_on){
        int user_input;
        printf("Enter '1' to display count,'2' to toggle bits,'3' to Exit");
        scanf("%d",&user_input);
        if(user_input==3){
            is_on=false;
        }else if(user_input==2){
```

```
            int row,col;
            printf("enter the row :");
            scanf("%d",&row);
            printf("enter the column :");
            scanf("%d",&col);
            toggle_bits(led,row,col,&count);
            print_matrix(led);
        }else if(user_input==1){
            printf("No of led's ON = %d\n",count);
        }else{
            printf("Invalid comment!");
        }
    }
}
```

3(Robot Game)

```
#include <stdio.h>
#include <string.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 x = 0, y = 0;
    grid[x][y] = 1;
    char direction[10];
    while (1) {
        printGrid(grid);
        printf("Enter direction (UP, DOWN, LEFT, RIGHT) or 'exit' to quit: ");
        scanf("%s", direction);
        if (strcmp(direction, "exit") == 0) {
            break;
        }
        int prev_x = x, prev_y = y;
        if (strcmp(direction, "UP") == 0) {
            if (x > 0) x--;
        } else if (strcmp(direction, "DOWN") == 0) {
            if (x < M - 1) x++;
        } else if (strcmp(direction, "LEFT") == 0) {
            if (y > 0) y--;
        } else if (strcmp(direction, "RIGHT") == 0) {
            if (y < N - 1) y++;
        } else {
            printf("Invalid direction. Please enter UP, DOWN, LEFT, or RIGHT.\n");
            continue;
        }
        grid[x][y] = 1;
        if (grid[x][y] == 1 && (x != prev_x || y != prev_y)) {
            printf("Revisited position: (%d, %d)\n", x, y);
        }
```

```
    }
    printf("Final state of the grid:\n");
    printGrid(grid);

    return 0;
}
```

4(LED Patterns)

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#define size 4
void clearcube(int cube[size][size][size]){
    for(int i=0;i<size;i++){
        for(int j=0;j<size;j++){
            for(int k=0;k<size;k++){
                cube[i][j][k]=0;
            }
        }
    }
}
void print_cube(int cube[size][size][size]){
    for(int i=0;i<size;i++){
        for(int j=0;j<size;j++){
            for(int k=0;k<size;k++){
                printf("%d",cube[i][j][k]);
            }
            printf("\n");
        }
```

```
        printf("\n");
    }
}
void simple_animation(int matrix[size][size][size],int frame){
    clearcube(matrix);
    int x=frame%size;
    int y=(frame/size)%size;
    int z=(frame/(size*size))%size;
    matrix[x][y][z]=1;
}
int main(){
    int cube[size][size][size]={{0}};
    int frame=0;
    while(1){
        simple_animation(cube,frame);
        print_cube(cube);
        usleep(500000);
        frame+=1;
        if(frame>=size*size*size){
            frame=0;
        }
        system("clear");
    }
    return 0;



}
```

5(Traffic lights)

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

#define light 3
#define roads 4
#define red 0
#define yellow 1
#define green 2

void print_light(int lights[roads][light]) {
   const char *colors[] = {"Red", "Yellow", "Green"};
   for (int i = 0; i < roads; i++) {
      printf("Road %d: ", i + 1);
      for (int j = 0; j < light; j++) {
         if (lights[i][j] == 1) {
            printf("%s ", colors[j]);
         }
      }
      printf("\n");
   }
}

void update_light(int lights[roads][light], int *cycle) {
   for (int i = 0; i < roads; i++) {
      for (int j = 0; j < light; j++) {
         lights[i][j] = 0;
      }
   }
```

```
    for (int i = 0; i < roads; i++) {
        if (*cycle == i) {
            lights[i][green] = 1;
        } else if (*cycle == (i + 1) % roads) {
            lights[i][yellow] = 1;
        } else {
            lights[i][red] = 1;
        }
    }

    *cycle = (*cycle + 1) % roads;
}

int main() {
    int lights[roads][light] = {0};
    int cycle = 0;

    for (int i = 0; i < 10; i++) {
        printf("The light for cycle %d:\n", i);
        update_light(lights, &cycle);
        print_light(lights);
        sleep(4);
    }

    return 0;
}
```

6 (weather monitoring)

```
#include <stdio.h>

#define D 5
#define L 2
#define H 6

int main() {
    float weatherData[D][L][H] = {
        {
            {30.2, 31.5, 32.0, 30.8, 29.5, 28.2},
            {25.0, 24.5, 25.1, 24.7, 23.5, 23.1}
        },
        {
            {32.2, 33.1, 34.0, 33.5, 32.0, 30.9},
            {27.0, 26.5, 27.1, 26.8, 26.0, 25.4}
        },
        {
            {33.5, 34.2, 35.0, 34.8, 34.2, 33.1},
            {28.2, 28.7, 29.1, 28.8, 28.4, 28.0}
        },
        {
            {35.2, 36.0, 36.5, 35.8, 35.2, 34.8},
            {29.1, 29.5, 29.8, 29.3, 28.9, 28.6}
        },
        {
            {31.5, 32.0, 32.3, 32.1, 31.8, 31.4},
            {26.7, 27.2, 27.5, 27.3, 27.0, 26.6}
        }
    };
```

```
    static float highestTemp = -1000.0;
    static int highestTempLocation = -1;
    static int highestTempDay = -1;

    for (int d = 0; d < D; d++) {
        for (int l = 0; l < L; l++) {
            float dailySum = 0.0;
            for (int h = 0; h < H; h++) {
                dailySum += weatherData[d][l][h];
            }
            float dailyAvg = dailySum / H;
            printf("Location %d, Day %d: Average Temperature = %.2f\n", l + 1, d + 1, dailyAvg);
            if (dailyAvg > highestTemp) {
                highestTemp = dailyAvg;
                highestTempLocation = l;
                highestTempDay = d;
            }
        }
    }

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

    return 0;
}
```

7 (Signal processsing)

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

```
#define X 5
#define Y 4
#define Z 6

void applyFilter(float signal[X][Y][Z], float 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++) {
            float sum = 0.0;
            int 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;
                     int ny = y + dy;
                     int 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;
         }
      }
   }
```

```
}

int main() {
    float signal[X][Y][Z] = {
        {
            {1, 2, 3, 4, 5, 6},
            {2, 3, 4, 5, 6, 7},
            {3, 4, 5, 6, 7, 8},
            {4, 5, 6, 7, 8, 9}
        },
        {
            {1, 1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1, 1}
        },
        {
            {9, 8, 7, 6, 5, 4},
            {8, 7, 6, 5, 4, 3},
            {7, 6, 5, 4, 3, 2},
            {6, 5, 4, 3, 2, 1}
        },
        {
            {4, 5, 6, 7, 8, 9},
            {5, 6, 7, 8, 9, 10},
            {6, 7, 8, 9, 10, 11},
            {7, 8, 9, 10, 11, 12}
        },
        {
```

```
        {1, 2, 3, 4, 5, 6},
        {2, 3, 4, 5, 6, 7},
        {3, 4, 5, 6, 7, 8},
        {4, 5, 6, 7, 8, 9}
    }
};

float result[X][Y][Z];

applyFilter(signal, result);

printf("Filtered Signal Data:\n");
for (int x = 0; x < X; x++) {
    for (int y = 0; y < Y; y++) {
        for (int z = 0; z < Z; z++) {
            printf("result[%d][%d][%d] = %.2f\n", x, y, z, result[x][y][z]);
        }
    }
}

return 0;
}
```

8(Inventory)

```
#include <stdio.h>

#define P 3
#define R 4
```

```
#define C 5
void updateInventory(int inventory[P][R][C], int productID, int row, int col, int quantity) {
    if (productID < 0 || productID >= P || row < 0 || row >= R || col < 0 || col >= C) {
        printf("Invalid location or product ID.\n");
        return;
    }
    inventory[productID][row][col] += quantity;
}

void checkLowStock(int inventory[P][R][C], int threshold) {
    for (int p = 0; p < P; p++) {
        for (int r = 0; r < R; r++) {
            for (int c = 0; c < C; c++) {
                if (inventory[p][r][c] < threshold) {
                    printf("Low stock detected: Product %d at location (%d, %d, %d) with quantity %d\n",
                        p + 1, r + 1, c + 1, inventory[p][r][c]);
                }
            }
        }
    }
}

void displayInventory(int inventory[P][R][C]) {
    for (int p = 0; p < P; p++) {
        printf("\nInventory for Product %d:\n", p + 1);
        for (int r = 0; r < R; r++) {
            for (int c = 0; c < C; c++) {
                printf("Location (%d, %d): %d units\n", r + 1, c + 1, inventory[p][r][c]);
            }
```

```
        }
    }
}

int main() {
    int inventory[P][R][C] = {{{0}}};
    int totalInventory[P] = {0};

    updateInventory(inventory, 0, 0, 0, 10);
    updateInventory(inventory, 0, 1, 1, 15);
    updateInventory(inventory, 1, 2, 2, 20);
    updateInventory(inventory, 2, 3, 4, 30);

    for (int p = 0; p < P; p++) {
        totalInventory[p] = 0;
        for (int r = 0; r < R; r++) {
            for (int c = 0; c < C; c++) {
                totalInventory[p] += inventory[p][r][c];
            }
        }
    }

    for (int p = 0; p < P; p++) {
        printf("\nTotal inventory for Product %d: %d units\n", p + 1, totalInventory[p]);
    }

    displayInventory(inventory);

    int threshold = 5;
```

```
    checkLowStock(inventory, threshold);

    return 0;
}
```

9

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

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

void applySobelFilter(int image[H][W], int output[H][W]) {
    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 sumX, sumY, magnitude;

    for (int i = 1; i < H - 1; i++) {
        for (int j = 1; j < W - 1; j++) {
            sumX = 0;
            sumY = 0;

            for (int k = -1; k <= 1; k++) {
                for (int l = -1; l <= 1; l++) {
                    sumX += image[i + k][j + l] * Gx[k + 1][l + 1];
                    sumY += image[i + k][j + l] * Gy[k + 1][l + 1];
                }
            }
```

```
            magnitude = (int)sqrt(sumX * sumX + sumY * sumY);
            if (magnitude > THRESHOLD) {
                output[i][j] = 255;
            } else {
                output[i][j] = 0;
            }
        }
    }
}

void displayImage(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] = {
        {255, 255, 255, 255, 255},
        {255, 0, 0, 0, 255},
        {255, 0, 255, 0, 255},
        {255, 0, 0, 0, 255},
        {255, 255, 255, 255, 255}
    };
```

```
    int output[H][W] = {{0}};

    applySobelFilter(image, output);
    displayImage(output);

    return 0;
}
```

10(sensor data aggregation)

```
#include <stdio.h>

#define X 3
#define Y 4
#define Z 5
#define CRITICAL_THRESHOLD 50

void populateData(int data[X][Y][Z]) {
    for (int x = 0; x < X; x++) {
        for (int y = 0; y < Y; y++) {
            for (int z = 0; z < Z; z++) {
                data[x][y][z] = rand() % 100;  // Random data between 0 and 99
            }
        }
    }
}

void analyzeData(int data[X][Y][Z], int *criticalCount, float *averagePerLayer) {
    int totalSum = 0;
    int totalCount = 0;
```

```
    for (int x = 0; x < X; x++) {
        int layerSum = 0;
        int layerCount = 0;
        for (int y = 0; y < Y; y++) {
            for (int z = 0; z < Z; z++) {
                if (data[x][y][z] > CRITICAL_THRESHOLD) {
                    (*criticalCount)++;
                }
                layerSum += data[x][y][z];
                layerCount++;
            }
        }
        totalSum += layerSum;
        totalCount += layerCount;
        averagePerLayer[x] = (float)layerSum / layerCount;
    }

    printf("Total Critical Sensors: %d\n", *criticalCount);
    printf("Overall Average Reading: %.2f\n", (float)totalSum / totalCount);
}

void displayResults(int data[X][Y][Z], float averagePerLayer[X]) {
    for (int x = 0; x < X; x++) {
        printf("\nLayer %d:\n", x + 1);
        for (int y = 0; y < Y; y++) {
            for (int z = 0; z < Z; z++) {
                printf("%d ", data[x][y][z]);
            }
```

```
            printf("\n");
        }
        printf("Average reading for Layer %d: %.2f\n", x + 1, averagePerLayer[x]);
    }
}

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

    populateData(data);
    analyzeData(data, &criticalCount, averagePerLayer);
    displayResults(data, averagePerLayer);

    return 0;
}
```