---Arrays and pointers--

1

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

#define G 9.81

void calculate_trajectory(double initial_velocity, double angle_degrees, double trajectory[][3], int size) {
    double angle_radians = angle_degrees * M_PI / 180.0;
    double dt = 0.1;
    for (int i = 0; i < size; ++i) {
        double t = i * dt;
        double x = initial_velocity * cos(angle_radians) * t;
        double y = initial_velocity * sin(angle_radians) * t - 0.5 * G * t * t;
        trajectory[i][0] = x;
        trajectory[i][1] = y;
        trajectory[i][2] = 0;
    }
}

void print_trajectory(double trajectory[][3], int size) {
    for (int i = 0; i < size; ++i) {
        printf("Time step %d: (x = %.2f, y = %.2f, z = %.2f)\n", i, trajectory[i][0], trajectory[i][1], trajectory[i][2]);
    }
}
```

```
int main() {
    double initial_velocity = 20.0;
    double angle_degrees = 45.0;
    int size = 10;
    double trajectory[size][3];
    calculate_trajectory(initial_velocity, angle_degrees, trajectory, size);
    print_trajectory(trajectory, size);

    return 0;
}
```

2

```
#include <stdio.h>
void update_position(const double *velocity, double *position, double dt) {
    for (int i = 0; i < 3; ++i) {
        position[i] += velocity[i] * dt;
    }
}

void simulate_orbit(const double *initial_conditions, double positions[][3], int steps, double dt) {
    double position[3] = {initial_conditions[0], initial_conditions[1], initial_conditions[2]};
    double velocity[3] = {initial_conditions[3], initial_conditions[4], initial_conditions[5]};
    for (int i = 0; i < 3; ++i) {
        positions[0][i] = position[i];
    }

    for (int step = 1; step < steps; ++step) {
        update_position(velocity, position, dt);
        for (int i = 0; i < 3; ++i) {
            positions[step][i] = position[i];
        }
    }
}

void print_trajectory(double positions[][3], int steps) {
    for (int i = 0; i < steps; ++i) {
        printf("Time step %d: (x = %.2f, y = %.2f, z = %.2f)\n", i, positions[i][0], positions[i][1], positions[i][2]);
    }
}

int main() {
    double initial_conditions[6] = {7000e3, 0, 0, 0, 7.5e3, 0};
    int steps = 10;
    double dt = 10.0;
    double positions[steps][3];
    simulate_orbit(initial_conditions, positions, steps, dt);
    print_trajectory(positions, steps);

    return 0;
}
```

3

```
#include<stdio.h>

void display_weather_data(int*data,int size){
    printf("Hourly weather data \n");
    for(int i=0;i<size;i+=3){
        printf("Hour %d :Temp: %dc ,windspeed: %dkm/hr,Pressure:
%dHpa\n",(i/3+1),data[i],data[i+1],data[i+2]);
    }
}
void calc_average(int *data,double *averages,int size){
    for(int i=0;i<3;i++){
        double sum=0.0;
        for(int j=0;j<size;j+=3){
            sum+=data[j+i];
        }
        averages[i] = sum / (size / 3);
    }
}

int main(){
    int weather_data[12]={
        20,15,1021,
        21, 14, 1011,
        19, 13, 1010,
        18, 12, 1013,
    };
    int datasize = sizeof(weather_data) / sizeof(weather_data[0]);
    double averages[3];
    display_weather_data(weather_data, datasize);
    calc_average(weather_data,averages,datasize);
    printf("\nDaily Averages:\n");
    printf("Average Temperature: %.2f°C\n", averages[0]);
    printf("Average Wind Speed: %.2f km/h\n", averages[1]);
    printf("Average Pressure: %.2f hPa\n", averages[2]);
    return 0;
}
```

4

```
#include <stdio.h>

#define MAX_HISTORY 100

double compute_pid(const double *errors, int size, const double *gains) {
    double proportional = errors[size - 1];
    double integral = 0.0;
    double derivative = 0.0;
    for (int i = 0; i < size; i++) {
        integral += errors[i];
    }

    if (size > 1) {

        derivative = errors[size - 1] - errors[size - 2];
    }


    double pid_output = (gains[0] * proportional) + (gains[1] * integral) + (gains[2] * derivative);

    return pid_output;
}

void update_errors(double *errors, int *size, double new_error) {
    if (*size < MAX_HISTORY) {
        errors[*size] = new_error;
        (*size)++;
    } else {
        for (int i = 1; i < MAX_HISTORY; i++) {
            errors[i - 1] = errors[i];
        }
        errors[MAX_HISTORY - 1] = new_error;
    }
}

int main() {
    double errors[MAX_HISTORY] = {0};
    int error_size = 0;
    double gains[3] = {1.0, 0.5, 0.2};
    double new_error = 0.5;
    update_errors(errors, &error_size, new_error);
    double control_output = compute_pid(errors, error_size, gains);
    printf("PID Control Output: %.2f\n", control_output);

    return 0;
}
```

5

```
#include <stdio.h>
void fuse_data(const double *sensor1, const double *sensor2, double *result, int size) {
    for (int i = 0; i < size; i++) {
        result[i] = (sensor1[i] + sensor2[i]) / 2.0;
    }
}
void calibrate_data(double *data, int size, double calibration_factor) {
    for (int i = 0; i < size; i++) {

        data[i] = data[i] * calibration_factor;
    }
}

int main() {
    double sensor1_data[] = {25.0, 26.5, 24.0, 23.5, 22.0};
    double sensor2_data[] = {24.5, 27.0, 23.5, 23.0, 21.5};
    int size = sizeof(sensor1_data) / sizeof(sensor1_data[0]);

    double fused_data[size];
    double calibration_factor = 1.05;
    fuse_data(sensor1_data, sensor2_data, fused_data, size);
    printf("Fused Data (Before Calibration):\n");
    for (int i = 0; i < size; i++) {
        printf("Fused[%d]: %.2f\n", i, fused_data[i]);
    }
    calibrate_data(fused_data, size, calibration_factor);
    printf("\nFused Data (After Calibration):\n");
    for (int i = 0; i < size; i++) {
        printf("Calibrated[%d]: %.2f\n", i, fused_data[i]);
    }

    return 0;
}
```

```
//6

#include <stdio.h>

#define MAX_FLIGHTS 100

typedef struct {
    int id;
    double altitude;
    double latitude;
    double longitude;
} flight_t;

void add_flight(flight_t *flights, int *flight_count, const flight_t *new_flight) {
    if (*flight_count < MAX_FLIGHTS) {
        flights[*flight_count] = *new_flight;
        (*flight_count)++;
        printf("Flight %d added.\n", new_flight->id);
    } else {
        printf("Error: Cannot add more flights. Max capacity reached.\n");
    }
}

void remove_flight(flight_t *flights, int *flight_count, int flight_id) {
    int found = 0;
    for (int i = 0; i < *flight_count; i++) {
        if (flights[i].id == flight_id) {
            for (int j = i; j < *flight_count - 1; j++) {
                flights[j] = flights[j + 1];
            }
            (*flight_count)--;
            printf("Flight %d removed.\n", flight_id);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Error: Flight ID %d not found.\n", flight_id);
    }
}

void display_flights(const flight_t *flights, int flight_count) {
    if (flight_count == 0) {
        printf("No active flights.\n");
        return;
    }
    printf("Active Flights:\n");
    for (int i = 0; i < flight_count; i++) {
        printf("Flight ID: %d, Altitude: %.2f feet, Coordinates: (%.2f, %.2f)\n",
               flights[i].id, flights[i].altitude, flights[i].latitude, flights[i].longitude);
    }
}

int main() {
    flight_t flights[MAX_FLIGHTS];
    int flight_count = 0;

    flight_t flight1 = {101, 35000.0, 37.7749, -122.4194};
    flight_t flight2 = {102, 28000.0, 34.0522, -118.2437};

    add_flight(flights, &flight_count, &flight1);
    add_flight(flights, &flight_count, &flight2);

    display_flights(flights, flight_count);

    remove_flight(flights, &flight_count, 101);

    display_flights(flights, flight_count);

    return 0;
}

//7
#include <stdio.h>
#include <math.h>

#define MAX_DATA_POINTS 100

void analyze_telemetry(const double *data, int size) {
    double sum = 0.0;
    double mean, variance = 0.0, stddev;

    for (int i = 0; i < size; i++) {
        sum += data[i];
    }
    mean = sum / size;
    for (int i = 0; i < size; i++) {
        variance += pow(data[i] - mean, 2);
    }
    variance /= size;
    stddev = sqrt(variance);

    printf("Telemetry Analysis:\n");
    printf("Mean: %.2f\n", mean);
    printf("Variance: %.2f\n", variance);
    printf("Standard Deviation: %.2f\n", stddev);
}

void filter_outliers(double *data, int size) {
    double mean = 0.0, sum = 0.0, stddev, threshold;
    for (int i = 0; i < size; i++) {
        sum += data[i];
    }
    mean = sum / size;

    sum = 0.0;
    for (int i = 0; i < size; i++) {
        sum += pow(data[i] - mean, 2);
    }
    stddev = sqrt(sum / size);
    threshold = 2.0 * stddev;

    int valid_size = 0;
    for (int i = 0; i < size; i++) {
        if (fabs(data[i] - mean) <= threshold) {
            data[valid_size++] = data[i];
        }
    }

    printf("Filtered Telemetry Data (Outliers Removed):\n");
    for (int i = 0; i < valid_size; i++) {
        printf("%.2f ", data[i]);
    }
    printf("\n");
}

int main() {
    double telemetry_data[MAX_DATA_POINTS] = {45.6, 46.3, 47.2, 48.1, 1000.0, 46.5, 47.9, 46.2, 48.0, 46.8};
    int size = 10;

    analyze_telemetry(telemetry_data, size);

    filter_outliers(telemetry_data, size);

    return 0;
}

//8

#include <stdio.h>

#define MAX_STAGES 5
```

```
double compute_total_thrust(const double *stages, int size) {
    double total_thrust = 0.0;
    for (int i = 0; i < size; i++) {
        total_thrust += stages[i];
    }
    return total_thrust;
}

void update_stage_thrust(double *stages, int stage, double new_thrust) {
    if (stage >= 0 && stage < MAX_STAGES) {
        stages[stage] = new_thrust;
        printf("Updated thrust for stage %d to %.2f N\n", stage + 1, new_thrust);
    } else {
        printf("Invalid stage number!\n");
    }
}

int main() {
    double thrust_stages[MAX_STAGES] = {500000.0, 1000000.0, 1500000.0, 2000000.0, 2500000.0};
    int size = 5;

    double total_thrust = compute_total_thrust(thrust_stages, size);
    printf("Total thrust: %.2f N\n", total_thrust);

    update_stage_thrust(thrust_stages, 2, 1600000.0);  // Update thrust for the 3rd stage
    total_thrust = compute_total_thrust(thrust_stages, size);
    printf("Updated total thrust: %.2f N\n", total_thrust);

    return 0;
```

```
}

//9

#include <stdio.h>

#define MAX_SECTIONS 10

void compute_stress_distribution(const double *forces, double *stress, int size) {
    for (int i = 0; i < size; i++) {
        stress[i] = forces[i] / (i + 1);  // Stress is assumed to be force per unit section index
    }
}

void display_stress(const double *stress, int size) {
    printf("Wing Stress Distribution:\n");
    for (int i = 0; i < size; i++) {
        printf("Section %d: Stress = %.2f N/m²\n", i + 1, stress[i]);
    }
}

int main() {
    double forces[MAX_SECTIONS] = {200.0, 450.0, 350.0, 500.0, 600.0, 400.0, 700.0, 550.0, 800.0, 900.0};
    double stress[MAX_SECTIONS];
    int size = 10;

    compute_stress_distribution(forces, stress, size);
    display_stress(stress, size);
    return 0;
}

//10
#include <stdio.h>
#include <math.h>

#define MAX_WAYPOINTS 100

typedef struct {
    double x;
    double y;
} waypoint_t;

double calculate_distance(const waypoint_t *a, const waypoint_t *b) {
    return sqrt(pow(b->x - a->x, 2) + pow(b->y - a->y, 2));
}

double optimize_path(const waypoint_t *waypoints, int size) {
    double total_distance = 0.0;
    for (int i = 0; i < size - 1; i++) {
        total_distance += calculate_distance(&waypoints[i], &waypoints[i + 1]);
    }
    return total_distance;
}

void add_waypoint(waypoint_t *waypoints, int *size, double x, double y) {
    if (*size < MAX_WAYPOINTS) {
        waypoints[*size].x = x;
        waypoints[*size].y = y;
        (*size)++;
        printf("Waypoint added at (%.2f, %.2f)\n", x, y);
    } else {
        printf("Cannot add more waypoints, max limit reached.\n");
    }
}

int main() {
    waypoint_t waypoints[MAX_WAYPOINTS] = {{0.0, 0.0}, {1.0, 2.0}, {4.0, 6.0}};
    int size = 3;

    double total_distance = optimize_path(waypoints, size);
    printf("Total path length: %.2f units\n", total_distance);

    add_waypoint(waypoints, &size, 7.0, 8.0);
    total_distance = optimize_path(waypoints, size);
    printf("Updated total path length: %.2f units\n", total_distance);

    return 0;
}

//11
#include <stdio.h>
#include <math.h>

#define QUATERNION_SIZE 4

void update_attitude(const double *quaternion, double *new_attitude) {
    for (int i = 0; i < QUATERNION_SIZE; i++) {
        new_attitude[i] = quaternion[i];
    }
}

void normalize_quaternion(double *quaternion) {
    double norm = 0.0;
    for (int i = 0; i < QUATERNION_SIZE; i++) {
        norm += quaternion[i] * quaternion[i];
    }
    norm = sqrt(norm);

    if (norm > 0.0) {
        for (int i = 0; i < QUATERNION_SIZE; i++) {
            quaternion[i] /= norm;
        }
    }
}

void display_quaternion(const double *quaternion) {
    printf("Quaternion: [%.4f, %.4f, %.4f, %.4f]\n", quaternion[0], quaternion[1], quaternion[2], quaternion[3]);
}

int main() {
    double attitude[QUATERNION_SIZE] = {1.0, 0.0, 0.0, 0.0};  // Initial quaternion (no rotation)
    double new_attitude[QUATERNION_SIZE];

    display_quaternion(attitude);
    update_attitude(attitude, new_attitude);
    printf("Updated Attitude: ");
    display_quaternion(new_attitude);

    attitude[1] = 1.0;  // Modify quaternion to simulate a change in attitude
    normalize_quaternion(attitude);
    printf("Normalized Attitude: ");
    display_quaternion(attitude);

    return 0;
}

//12
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_POINTS 10

void simulate_heat_transfer(const double *material_properties, double *temperatures, int size) {
    for (int i = 1; i < size - 1; i++) {
        temperatures[i] += material_properties[0] * (temperatures[i - 1] - 2 * temperatures[i] +
temperatures[i + 1]);
    }
}

void display_temperatures(const double *temperatures, int size) {
    for (int i = 0; i < size; i++) {
        printf("Point %d: Temperature = %.2f°C\n", i + 1, temperatures[i]);
    }
}

int main() {
    double material_properties[3];
    double temperatures[MAX_POINTS];
    int size;

    printf("Enter 3 material property values (e.g., thermal conductivity, etc.): ");
    for (int i = 0; i < 3; i++) {
        scanf("%lf", &material_properties[i]);
    }

    printf("Enter the number of temperature points (up to %d): ", MAX_POINTS);
    scanf("%d", &size);

    if (size > MAX_POINTS) {
        printf("Exceeds maximum allowed points. Setting to %d.\n", MAX_POINTS);
        size = MAX_POINTS;
    }

    printf("Enter initial temperatures for %d points:\n", size);
    for (int i = 0; i < size; i++) {
        scanf("%lf", &temperatures[i]);
    }

    display_temperatures(temperatures, size);
    simulate_heat_transfer(material_properties, temperatures, size);

    display_temperatures(temperatures, size);

    return 0;
}

//13
#include<stdio.h>
void overall_fuel_efficiency(float *arr,int arr_size){
    double sum,avg;
    for(int i=0;i<arr_size;i++){
        sum+=*(arr+i);
    }
    avg=sum/arr_size;
    printf("The overall fuel efficiency is = %.2f",avg);
}

void print_data(float *arr,int arr_size){
    printf("Updated array values are :\n");
    for(int i=0;i<arr_size;i++){
        printf("value at position %d is :%.2f\n",i+1,*(arr+i));
    }
}

void update_fuel_consumption(float *f_c,int n){
    int val,new_val;
    printf("Enter the value to update :");
    scanf("%d",&val);
    printf("Enter the new value for intrevel %d :",val);
    scanf("%d",&new_val);
    f_c[val-1]=new_val;
}

int main(){
    int size=10;
    float fuel_consumption[size];
    printf("Enter the fuel consumption at different intervels :");
    for(int i=0;i<size;i++){
        scanf("%f",&fuel_consumption[i]);
    }
    update_fuel_consumption(fuel_consumption,size);
    print_data(fuel_consumption,size);
    overall_fuel_efficiency(fuel_consumption,size);
    return 0;
}

//14
#include <stdio.h>

#define MAX_PARAMETERS 10

double compute_link_budget(const double *parameters, int size) {
    double link_budget = 0.0;
    for (int i = 0; i < size; i++) {
        link_budget += parameters[i];
    }
    return link_budget;
```

```
}

void update_parameters(double *parameters, int index, double value) {
    if (index >= 0) {
        parameters[index] = value;
    }
}

int main() {
    double parameters[MAX_PARAMETERS];
    int size, index;
    double value;

    printf("Enter the number of communication parameters (up to %d): ", MAX_PARAMETERS);
    scanf("%d", &size);
    if (size > MAX_PARAMETERS) {
        size = MAX_PARAMETERS;
    }

    printf("Enter %d communication parameters (e.g., power, losses, etc.):\n", size);
    for (int i = 0; i < size; i++) {
        scanf("%lf", &parameters[i]);
    }

    printf("Initial communication parameters:\n");
    for (int i = 0; i < size; i++) {
        printf("Parameter %d: %.2f\n", i + 1, parameters[i]);
    }
    printf("Enter the index to update (0 to %d): ", size - 1);
    scanf("%d", &index);
    printf("Enter the new value for parameter %d: ", index);
    scanf("%lf", &value);

    update_parameters(parameters, index, value);

    printf("Updated communication parameters:\n");
    for (int i = 0; i < size; i++) {
        printf("Parameter %d: %.2f\n", i + 1, parameters[i]);
    }

    double link_budget = compute_link_budget(parameters, size);
    printf("Total Link Budget: %.2f dB\n", link_budget);

    return 0;
}

//15
#include <stdio.h>
#include <math.h>

#define MAX_SIZE 10

void detect_turbulence(const double *accelerations, int size, double *output) {
    for (int i = 1; i < size - 1; i++) {
        double diff1 = accelerations[i] - accelerations[i - 1];
        double diff2 = accelerations[i + 1] - accelerations[i];
        if (fabs(diff1) > 0.5 && fabs(diff2) > 0.5) {
            output[i] = 1.0; // Turbulence detected
        } else {
            output[i] = 0.0; // No turbulence
        }
    }
}

void log_turbulence(double *turbulence_log, const double *detection_output, int size) {
    for (int i = 0; i < size; i++) {
        if (detection_output[i] == 1.0) {
            turbulence_log[i] = 1.0; // Log turbulence event
        } else {
            turbulence_log[i] = 0.0; // No turbulence
        }
    }
}

int main() {
    double accelerations[MAX_SIZE];
    double turbulence_output[MAX_SIZE];
    double turbulence_log[MAX_SIZE];
    int size;

    printf("Enter the number of acceleration data points (up to %d): ", MAX_SIZE);
    scanf("%d", &size);
    if (size > MAX_SIZE) {
        size = MAX_SIZE;
    }
    printf("Enter the acceleration data for %d points:\n", size);
    for (int i = 0; i < size; i++) {
        scanf("%lf", &accelerations[i]);
    }

    detect_turbulence(accelerations, size, turbulence_output);

    printf("Turbulence detection results:\n");
    for (int i = 0; i < size; i++) {
        printf("Point %d: %.2f\n", i + 1, turbulence_output[i]);
    }

    log_turbulence(turbulence_log, turbulence_output, size);

    printf("Logged turbulence events:\n");
    for (int i = 0; i < size; i++) {
        if (turbulence_log[i] == 1.0) {
            printf("Turbulence detected at point %d\n", i + 1);
        }
    }

    return 0;
}
```