```c
/*Pointers: Use to traverse the trajectory array.
Arrays: Store trajectory points (x, y, z) at discrete time intervals.
Functions:
void calculate_trajectory(const double *parameters, double *trajectory,
int size): Takes the initial velocity, angle, and an array to store
trajectory points.
void print_trajectory(const double *trajectory, int size): Prints the
stored trajectory points.*/
#include <stdio.h>
#include <math.h>
#define GRAVITY 9.81
void calculate_trajectory(const double *parameters, double *trajectory,
int size) {
    double initial_velocity = parameters[0];
    double angle = parameters[1];
    double time_step = parameters[2];

    // Convert angle to radians
    double angle_rad = angle * M_PI / 180.0;

    for (int i = 0; i < size; i++) {
        double t = i * time_step;

        double x = initial_velocity * cos(angle_rad) * t;
        double y = initial_velocity * sin(angle_rad) * t - 0.5 * GRAVITY *
t * t;
        double z = 0;

        trajectory[i * 3] = x;
        trajectory[i * 3 + 1] = y;
        trajectory[i * 3 + 2] = z;
        if (y < 0) {
            for (int j = i; j < size; j++) {
                trajectory[j * 3] = 0;
                trajectory[j * 3 + 1] = 0;
                trajectory[j * 3 + 2] = 0;
            }
            break;
        }
    }
```

```
}
void print_trajectory(const double *trajectory, int size) {
    printf("Trajectory Points:\n");

    for (int i = 1; i < size; i++)
     {
        double x = trajectory[i * 3];
        double y = trajectory[i * 3 + 1];
        double z = trajectory[i * 3 + 2];

        if (x == 0 && y == 0 && z == 0) break;
        printf("Point %d: (x: %f, y: %f, z: %f)\n", i + 1, x, y, z);
    }
}

int main() {
    double parameters[] = {50.0, 45.0, 0.1};
    int size = 100;
    double trajectory[size * 3];
    calculate_trajectory(parameters, trajectory, size);
    print_trajectory(trajectory, size);

    return 0;
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($?) {
Trajectory Points:
Point 2: (x: 3.535534, y: 3.486484, z: 0.000000)
Point 3: (x: 7.071068, y: 6.874868, z: 0.000000)
Point 4: (x: 10.606602, y: 10.165152, z: 0.000000)
Point 5: (x: 14.142136, y: 13.357336, z: 0.000000)
Point 6: (x: 17.677670, y: 16.451420, z: 0.000000)
Point 7: (x: 21.213203, y: 19.447403, z: 0.000000)
Point 8: (x: 24.748737, y: 22.345287, z: 0.000000)
Point 9: (x: 28.284271, y: 25.145071, z: 0.000000)
Point 10: (x: 31.819805, y: 27.846755, z: 0.000000)
Point 11: (x: 35.355339, y: 30.450339, z: 0.000000)
Point 12: (x: 38.890873, y: 32.955823, z: 0.000000)
Point 13: (x: 42.426407, y: 35.363207, z: 0.000000)
Point 14: (x: 45.961941, y: 37.672491, z: 0.000000)
Point 15: (x: 49.497475, y: 39.883675, z: 0.000000)
Point 16: (x: 53.033009, y: 41.996759, z: 0.000000)
Point 17: (x: 56.568542, y: 44.011742, z: 0.000000)
Point 18: (x: 60.104076, y: 45.928626, z: 0.000000)
Point 19: (x: 63.639610, y: 47.747410, z: 0.000000)
Point 20: (x: 67.175144, y: 49.468094, z: 0.000000)
Point 21: (x: 70.710678, y: 51.090678, z: 0.000000)
Point 22: (x: 74.246212, y: 52.615162, z: 0.000000)
Point 23: (x: 77.781746, y: 54.041546, z: 0.000000)
Point 24: (x: 81.317280, y: 55.369830, z: 0.000000)
Point 25: (x: 84.852814, y: 56.600014, z: 0.000000)
Point 26: (x: 88.388348, y: 57.732098, z: 0.000000)
```

```c
#include<stdio.h>
void update_position(const double *velocity, double *position, int size);
void simulate_orbit(const double *initial_conditions, double *positions,
int steps);
void print_positions(const double *positions, int steps);
void main()
{
    double initial[]={1000,2000,1500,50,60,70 };
    int steps =10;
    double position[steps*3];
    simulate_orbit(initial,position,steps);
```

```
    print_positions(position,steps);

}
void simulate_orbit(const double *initial_conditions, double *positions,
int steps) {
    const double *initial_position = initial_conditions;
    const double *velocity = initial_conditions + 3;
    for (int i = 0; i < 3; i++) {
        positions[i] = initial_position[i];
    }
    for (int step = 1; step < steps; step++) {
        double *current_position = positions + step * 3;
        const double *previous_position = positions + (step - 1) * 3;
        for (int i = 0; i < 3; i++) {
            current_position[i] = previous_position[i];
        }
        update_position(velocity, current_position, 3);
    }
}

void update_position(const double *velocity, double *position, int size) {
    for (int i = 0; i < size; i++) {
        position[i] += velocity[i];
    }
}
void print_positions(const double *positions, int steps) {
    printf("Satellite Positions:\n");
    for (int step = 0; step < steps; step++) {
        const double *position = positions + step * 3;
        printf("Step %d: (x: %.2f, y: %.2f, z: %.2f)\n", step,
position[0], position[1], position[2]);
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\"
Satellite Positions:
Step 0: (x: 1000.00, y: 2000.00, z: 1500.00)
Step 1: (x: 1050.00, y: 2060.00, z: 1570.00)
Step 2: (x: 1100.00, y: 2120.00, z: 1640.00)
Step 3: (x: 1150.00, y: 2180.00, z: 1710.00)
Step 4: (x: 1200.00, y: 2240.00, z: 1780.00)
Step 5: (x: 1250.00, y: 2300.00, z: 1850.00)
Step 6: (x: 1300.00, y: 2360.00, z: 1920.00)
Step 7: (x: 1350.00, y: 2420.00, z: 1990.00)
Step 8: (x: 1400.00, y: 2480.00, z: 2060.00)
Step 9: (x: 1450.00, y: 2540.00, z: 2130.00)
PS D:\projects\quest\C>
```

```c
/*Pointers: Traverse weather data arrays efficiently.
Arrays: Store hourly temperature, wind speed, and pressure.
Functions:
void calculate_daily_averages(const double *data, int size, double
*averages): Computes daily averages for each parameter.
void display_weather_data(const double *data, int size): Displays data for
monitoring purposes.
Pass Arrays as Pointers: Pass weather data as pointers to processing
functions.*/
#include<stdio.h>
void calculate_daily_averages(const double *data, int size, double
*averages);
void display_weather_data(const double *data, int size);
void main()
{
    int size=5;
    double data[15]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    double avg[3];
    double *const p1=data;
    double *const p2=avg;
    calculate_daily_averages(p1,size,p2);
    display_weather_data(data,size);
```

```
}
void calculate_daily_averages(const double *data, int size, double
*averages)
{
    double sumt=0,sums=0,sump=0;
    for(int i=0;i<5;i++)
    {
        sumt+=data[i*3];
        sums+=data[i*3+1];
        sump+=data[i*3+2];
    }
averages[0]=sumt/size;
averages[0]=sums/size;
averages[0]=sump/size;

}
void display_weather_data(const double *data, int size) {
    printf("\nWeather Data:\n");
    printf("Day\tTemperature\tSpeed\tPressure\n");
    for (int i = 0; i < size; i++) {
        printf("%d\t%.2f\t\t%.2f\t%.2f\n", i + 1, data[i * 3], data[i * 3
+ 1], data[i * 3 + 2]);
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C

Weather Data:
Day     Temperature     Speed   Pressure
1       1.00            2.00    3.00
2       4.00            5.00    6.00
3       7.00            8.00    9.00
4       10.00           11.00   12.00
5       13.00           14.00   15.00
PS D:\projects\quest\C>
```

```c
/*Pointers: Traverse and manipulate error values in arrays.
Arrays: Store historical error values for proportional, integral, and
derivative calculations.
Functions:
double compute_pid(const double *errors, int size, const double *gains):
Calculates control output using PID logic.
void update_errors(double *errors, double new_error): Updates the error
array with the latest value.
Pass Arrays as Pointers: Use pointers for the errors array and the gains
array.*/
#include<stdio.h>
double compute_pid(const double *errors, int size, const double *gains);
void update_errors(double *errors, double new_error);
void main()
{
    double gains[3] = {1.0, 0.5, 0.1};
    double errors[3] = {0.0, 0.0, 0.0};
    double error_inputs[] = {5.0, 4.0, 3.0, 2.0, 1.0};
    int num_inputs = sizeof(error_inputs) / sizeof(error_inputs[0]);
    printf("PID Control Simulation:\n");
    for (int i = 0; i < num_inputs; i++)
    {
        update_errors(errors, error_inputs[i]);
        double output = compute_pid(errors, 3, gains);
```

```
        printf("Step %d: Error = %.2f, PID Output = %.2f\n", i + 1,
error_inputs[i], output);
    }
}
void update_errors(double *errors, double new_error) {
    errors[2] = errors[1];
    errors[1] += errors[0];
    errors[0] = new_error;
}
double compute_pid(const double *errors, int size, const double *gains) {
    if (size < 3) {
        printf("Error: Insufficient size for PID calculation.\n");
        return 0.0;
    }
    double proportional = gains[0] * errors[0];
    double integral = gains[1] * errors[1];
    double derivative = gains[2] * (errors[0] - errors[2]);
    return proportional + integral + derivative;
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ;
PID Control Simulation:
Step 1: Error = 5.00, PID Output = 5.50
Step 2: Error = 4.00, PID Output = 6.90
Step 3: Error = 3.00, PID Output = 7.30
Step 4: Error = 2.00, PID Output = 7.30
Step 5: Error = 1.00, PID Output = 6.90
PS D:\projects\quest\C>
```

```
/*Pointers: Handle sensor readings and fusion results.
Arrays: Store data from multiple sensors.
Functions:
void fuse_data(const double *sensor1, const double *sensor2, double
*result, int size): Merges two sensor datasets into a single result array.
void calibrate_data(double *data, int size): Adjusts sensor readings based
on calibration data.
```

```c
/*Pass Arrays as Pointers: Pass sensor arrays as pointers to fusion and calibration functions.*/
#include<stdio.h>
void fuse_data(const double *sensor1, const double *sensor2, double *result, int size);
void calibrate_data(double *data, int size);
void main()
{
    int size=5;
    double sensor1[5]={1,2,3,4,5};
    double sensor2[5]={6,7,8,9,10};
    double result[5];
    double *p1 =sensor1;
    double *p2 =sensor2;
    double *r=result;
    fuse_data(p1,p2,r,size);
    printf("Sensor1  Sensor2 Result\n");
    for(int i=0;i<5;i++)
    {
        printf("%.2f\t %.2f\t %.2f\n",sensor1[i],sensor2[i],result[i]);
    }
    calibrate_data(result,size);
    printf("Calibrated data\n");
    for(int i=0;i<size;i++)
    printf("%.2f\n",result[i]);

}
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);
}
void calibrate_data(double *data, int size)
{
    double calibration=10;
    for(int i=0;i<size;i++)
    {
        if(data[i]>calibration)
        data[i]=1;
```

```
        else
        data[i]=0;
    }

}
```

```
PS D:\projects\quest\C> cd "d:\
Sensor1  Sensor2 Result
1.00     6.00    7.00
2.00     7.00    9.00
3.00     8.00    11.00
4.00     9.00    13.00
5.00     10.00   15.00
Calibrated data
0.00
0.00
1.00
1.00
1.00
PS D:\projects\quest\C>
```

```
/*Pointers: Traverse the array of flight structures.
Arrays: Store details of active flights (e.g., ID, altitude, coordinates).
Functions:
void add_flight(flight_t *flights, int *flight_count, const flight_t
*new_flight): Adds a new flight to the system.
void remove_flight(flight_t *flights, int *flight_count, int flight_id):
Removes a flight by ID.
Pass Arrays as Pointers: Use pointers to manipulate the array of flight
structures.*/
#include <stdio.h>
#include <string.h>
#define MAX_FLIGHTS 100
typedef struct {
    int id;
    double altitude;
    double x, y;
```

```
} flight_t;
void add_flight(flight_t *flights, int *flight_count, const flight_t
*new_flight) {
    if (*flight_count >= MAX_FLIGHTS) {
        printf("Error: Maximum flight capacity reached.\n");
        return;
    }
    flights[*flight_count] = *new_flight;
    (*flight_count)++;
    printf("Flight %d added successfully.\n", new_flight->id);
}
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) {
            found = 1;
            for (int j = i; j < *flight_count - 1; j++) {
                flights[j] = flights[j + 1];
            }
            (*flight_count)--;
            printf("Flight %d removed successfully.\n", flight_id);
            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("\nActive Flights:\n");
    printf("ID\tAltitude\tCoordinates (x, y)\n");
    for (int i = 0; i < flight_count; i++) {
        printf("%d\t%.2f\t\t(%.2f, %.2f)\n", flights[i].id,
flights[i].altitude, flights[i].x, flights[i].y);
    }
}
```

```c
int main() {
    flight_t flights[MAX_FLIGHTS];
    int flight_count = 0;
    flight_t flight1 = {101, 30000.0, 50.5, 60.5};
    flight_t flight2 = {102, 35000.0, 70.0, 80.0};
    flight_t flight3 = {103, 40000.0, 90.0, 100.0};
    add_flight(flights, &flight_count, &flight1);
    add_flight(flights, &flight_count, &flight2);
    add_flight(flights, &flight_count, &flight3);
    display_flights(flights, flight_count);
    remove_flight(flights, &flight_count, 102);
    display_flights(flights, flight_count);

    return 0;
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ;
Flight 101 added successfully.
Flight 102 added successfully.
Flight 103 added successfully.

Active Flights:
ID      Altitude        Coordinates (x, y)
101     30000.00                (50.50, 60.50)
102     35000.00                (70.00, 80.00)
103     40000.00                (90.00, 100.00)
Flight 102 removed successfully.

Active Flights:
ID      Altitude        Coordinates (x, y)
101     30000.00                (50.50, 60.50)
103     40000.00                (90.00, 100.00)
PS D:\projects\quest\C>
```

```c
/*Pointers: Traverse telemetry data arrays.
Arrays: Store telemetry parameters (e.g., power, temperature, voltage).
Functions:
```

```
void analyze_telemetry(const double *data, int size): Computes statistical
metrics for telemetry data.
void filter_outliers(double *data, int size): Removes outliers from the
telemetry data array.
Pass Arrays as Pointers: Pass telemetry data arrays to both functions.*/
#include<stdio.h>
void analyze_telemetry(const double *data, int size);
void filter_outliers(double *data, int size);
void main()
{
    int size=5;
    double data[15]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    double *const ptr= data;
    analyze_telemetry(ptr,size);
    filter_outliers(ptr,size);
}
void analyze_telemetry(const double *data, int size)
{
    double tp=0,tt=0,tv=0;
    for(int i=0;i<size;i++)
    {
        tp+=*(data+i*3);
        tt+=*(data+i*3+1);
        tv+=*(data+i*3+2);

    }
    printf("Total power is %.2f \n",tp);
    printf("Total temperature is %.2f \n",tt);
    printf("Total voltage is %.2f \n",tv);
}
void filter_outliers(double *data, int size)
{
    double tp=0,tt=0,tv=0;
    for(int i=0;i<size;i++)
    {
        tp+=*(data+i*3);
        tt+=*(data+i*3+1);
        tv+=*(data+i*3+2);

    }
```

```
    for(int i=0;i<size;i++)
    {

        if(data[i*3]<(tp/5) && data[i*3+1]<(tt/5) && data[i*3+2]<(tv/5))
       {
         printf("%.2f %.2f %.2f\n",data[i*3],data[i*3+1],data[i*3+2]);
       }
    }
}
```

```
PS D:\projects\quest\C> cd "d:\project
Total power is 35.00
Total temperature is 40.00
Total voltage is 45.00
1.00 2.00 3.00
4.00 5.00 6.00
PS D:\projects\quest\C>
```

```
/*Pointers: Traverse thrust arrays.
Arrays: Store thrust values for each stage of the rocket.
Functions:
double compute_total_thrust(const double *stages, int size): Calculates
cumulative thrust across all stages.
void update_stage_thrust(double *stages, int stage, double new_thrust):
Updates thrust for a specific stage.
Pass Arrays as Pointers: Use pointers for thrust arrays.*/
#include<stdio.h>
double compute_total_thrust(const double *stages, int size);
void update_stage_thrust(double *stages, int stage, double new_thrust);
void main()
{
    double thrust[5]={50,40,30,20,10};
    int size = 5;
    double *const ptr=thrust;
    for(int i=0;i<5;i++)
    printf("Thrust at stage %d : %.2f\n",i,thrust[i]);
    printf("\n");
```

```
    compute_total_thrust(ptr,size);
    update_stage_thrust(ptr,2,60);
    update_stage_thrust(ptr,4,90);
    update_stage_thrust(ptr,1,160);
    for(int i=0;i<5;i++)
    printf("Thrust at stage %d : %.2f\n",i,thrust[i]);

}
double compute_total_thrust(const double *stages, int size)
{
    double sum=0;
    for(int i=0;i<size;i++)
    sum+=stages[i];
    printf("Total thrust is %.2f\n",sum);
}
void update_stage_thrust(double *stages, int stage, double new_thrust)
{
    stages[stage]=new_thrust;
}
```

```
PS D:\projects\quest\C> cd "d:\proje
Thrust at stage 0 : 50.00
Thrust at stage 1 : 40.00
Thrust at stage 2 : 30.00
Thrust at stage 3 : 20.00
Thrust at stage 4 : 10.00

Total thrust is 150.00
Thrust at stage 0 : 50.00
Thrust at stage 1 : 160.00
Thrust at stage 2 : 60.00
Thrust at stage 3 : 20.00
Thrust at stage 4 : 90.00
PS D:\projects\quest\C>
```

```
/*Pointers: Access stress values at various points.
Arrays: Store stress values for discrete wing sections.
Functions:
```

```c
/*void compute_stress_distribution(const double *forces, double *stress, int size): Computes stress values based on applied forces.
void display_stress(const double *stress, int size): Displays the stress distribution.
Pass Arrays as Pointers: Pass stress arrays to computation functions.*/
#include<stdio.h>
void compute_stress_distribution(const double *forces, double *stress, int size);
void display_stress(const double *stress, int size);
void main()
{
double stress[5]={10,20,30,40,50};
double force[5]={20,30,40,50,60};
int size = 5;
double *const s=stress;
double *const f=force;
compute_stress_distribution(f,s,size);
display_stress(s,size);
}
void compute_stress_distribution(const double *forces, double *stress, int size)
{
    for(int i=0;i<size;i++)
    {
        stress[i]=stress[i]+forces[i];
    }
}
void display_stress(const double *stress, int size)
{
    for (int  i = 0; i < size; i++)
    {
        printf("Stress %d is %.2f\n",i,stress[i]);

    }

}
```

```
PS D:\projects\quest\C> cd "d:\pr
Stress 0 is 30.00
Stress 1 is 50.00
Stress 2 is 70.00
Stress 3 is 90.00
Stress 4 is 110.00
PS D:\projects\quest\C>
```

```c
/*Pointers: Traverse waypoint arrays.
Arrays: Store coordinates of waypoints.
Functions:
double optimize_path(const double *waypoints, int size): Reduces the total path length.
void add_waypoint(double *waypoints, int *size, double x, double y): Adds a new waypoint.
Pass Arrays as Pointers: Use pointers to access and modify waypoints.*/
#include <stdio.h>
#include <math.h>
#define MAX_WAYPOINTS 100
double optimize_path(const double *waypoints, int size);
void add_waypoint(double *waypoints, int *size, double x, double y);
double calculate_distance(double x1, double y1, double x2, double y2);
void main() {
    double waypoints[MAX_WAYPOINTS * 2];
    int size = 0;
    add_waypoint(waypoints, &size, 0.0, 0.0);
    add_waypoint(waypoints, &size, 3.0, 4.0);
    add_waypoint(waypoints, &size, 7.0, 1.0);
    printf("Waypoints:\n");
    for (int i = 0; i < size; i++) {
        printf("(%.2f, %.2f)\n", waypoints[i * 2], waypoints[i * 2 + 1]);
    }
    double total_length = optimize_path(waypoints, size);
    printf("Total Path Length: %.2f\n", total_length);


}
double calculate_distance(double x1, double y1, double x2, double y2) {
```

```
    return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}
double optimize_path(const double *waypoints, int size) {
    double total_length = 0.0;
    for (int i = 0; i < size - 1; i++) {
        double x1 = *(waypoints + i * 2);
        double y1 = *(waypoints + i * 2 + 1);
        double x2 = *(waypoints + (i + 1) * 2);
        double y2 = *(waypoints + (i + 1) * 2 + 1);

        total_length += calculate_distance(x1, y1, x2, y2);
    }
    return total_length;
}
void add_waypoint(double *waypoints, int *size, double x, double y) {
    if (*size >= MAX_WAYPOINTS) {
        printf("Error: Maximum waypoint capacity reached.\n");
        return;
    }

    *(waypoints + (*size) * 2) = x;
    *(waypoints + (*size) * 2 + 1) = y;
    (*size)++;
    printf("Waypoint (%.2f, %.2f) added successfully.\n", x, y);
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; i
Waypoint (0.00, 0.00) added successfully.
Waypoint (3.00, 4.00) added successfully.
Waypoint (7.00, 1.00) added successfully.
Waypoints:
(0.00, 0.00)
(3.00, 4.00)
(7.00, 1.00)
Total Path Length: 10.00
PS D:\projects\quest\C>
```

```c
/*Pointers: Manipulate quaternion arrays.
Arrays: Store quaternion values for attitude control.
Functions:
void update_attitude(const double *quaternion, double *new_attitude):
Updates the satellite's attitude.
void normalize_quaternion(double *quaternion): Ensures quaternion
normalization.
Pass Arrays as Pointers: Pass quaternion arrays as pointers.*/
#include<stdio.h>
#include<math.h>
#define MAX 100
void update_attitude(const double *quaternion, double *new_attitude);
void normalize_quaternion(double *quaternion);
void main()
{
    double quaternion[MAX][4]={
         {0.707, 0.0, 0.707, 0.0},
        {0.923, 0.0, 0.0, 0.383}
    };
    int quaternion_count = 2;
    double new_attitude[4];
     for (int i = 0; i < quaternion_count; i++) {
```

```
        normalize_quaternion(quaternion[i]);
        printf("Normalized Quaternion %d: (%.3f, %.3f, %.3f, %.3f)\n", i +
1, quaternion[i][0], quaternion[i][1], quaternion[i][2],
quaternion[i][3]);
    }
     update_attitude(quaternion[0], new_attitude);
     printf("\nUpdated Attitude Quaternion: (%.3f, %.3f, %.3f, %.3f)\n",
           new_attitude[0],
           new_attitude[1],
           new_attitude[2],
           new_attitude[3]);
}
void normalize_quaternion(double *quaternion) {
    double magnitude = sqrt(quaternion[0] * quaternion[0] +quaternion[1] *
quaternion[1] +quaternion[2] * quaternion[2] +quaternion[3] *
quaternion[3]);

    if (magnitude > 0.0) {
        quaternion[0] /= magnitude;
        quaternion[1] /= magnitude;
        quaternion[2] /= magnitude;
        quaternion[3] /= magnitude;
    }
}
void update_attitude(const double *quaternion, double *new_attitude) {
    for (int i = 0; i < 4; i++) {
        new_attitude[i] = quaternion[i];
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($?) { gc
Normalized Quaternion 1: (0.707, 0.000, 0.707, 0.000)
Normalized Quaternion 2: (0.924, 0.000, 0.000, 0.383)

Updated Attitude Quaternion: (0.707, 0.000, 0.707, 0.000)
PS D:\projects\quest\C>
```

```c
/*Pointers: Access temperature arrays for computation.
Arrays: Store temperature values at discrete points.
Functions:
void simulate_heat_transfer(const double *material_properties, double
*temperatures, int size): Simulates heat transfer across the material.
void display_temperatures(const double *temperatures, int size): Outputs
temperature distribution.
Pass Arrays as Pointers: Use pointers for temperature arrays.*/
#include<stdio.h>
void simulate_heat_transfer(const double *material_properties, double
*temperatures, int size);
void display_temperatures(const double *temperatures, int size);
void main()
{
    double temp[5]={10,15,20,25,30};
    double material[5]={5,11,6,5,15};
    int size=5;
    double *const t=temp;
    double *const m=material;
    simulate_heat_transfer(m,t,size);
    display_temperatures(t,size);
}
void simulate_heat_transfer(const double *material_properties, double
*temperatures, int size)
{
    for(int i=0;i<size;i++)
    {
        temperatures[i]/=material_properties[i];
```

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

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; i
Temperature distribution for material 0 : 2.00
Temperature distribution for material 1 : 1.36
Temperature distribution for material 2 : 3.33
Temperature distribution for material 3 : 5.00
Temperature distribution for material 4 : 2.00
PS D:\projects\quest\C>
```

```
/*Pointers: Traverse fuel consumption arrays.
Arrays: Store fuel consumption at different time intervals.
Functions:
double compute_efficiency(const double *fuel_data, int size): Calculates
overall fuel efficiency.
void update_fuel_data(double *fuel_data, int interval, double
consumption): Updates fuel data for a specific interval.
Pass Arrays as Pointers: Pass fuel data arrays as pointers*/
#include<stdio.h>
double compute_efficiency(const double *fuel_data, int size);
void update_fuel_data(double *fuel_data, int interval, double
consumption);
void main()
{
    int size =5;
    double fuel[5]={10,20,30,40,50};
    double *const f=fuel;
    for(int i=0;i<size;i++)
```

```
    printf("Consumption at point %d is %.2f\n",i,fuel[i]);
    compute_efficiency(f,size);
    update_fuel_data(f,3,60);
    for(int i=0;i<size;i++)
    printf("Consumption at point %d is %.2f\n",i,fuel[i]);

}
double compute_efficiency(const double *fuel_data, int size)
{
    double sum=0;
    for(int i=0;i<size;i++)
    {
        sum+=fuel_data[i];
    }
    printf("Efficiency is %.2f\n",sum/size);
}
void update_fuel_data(double *fuel_data, int interval, double consumption)
{
    fuel_data[interval]=consumption;
}
```

```
PS D:\projects\quest\C> cd "d:\projects
Consumption at point 0 is 10.00
Consumption at point 1 is 20.00
Consumption at point 2 is 30.00
Consumption at point 3 is 40.00
Consumption at point 4 is 50.00
Efficiency is 30.00
Consumption at point 0 is 10.00
Consumption at point 1 is 20.00
Consumption at point 2 is 30.00
Consumption at point 3 is 60.00
Consumption at point 4 is 50.00
PS D:\projects\quest\C>
```

```
/*Pointers: Handle parameter arrays for computation.
```

```c
/*Arrays: Store communication parameters like power and losses.
Functions:
double compute_link_budget(const double *parameters, int size): Calculates
the total link budget.
void update_parameters(double *parameters, int index, double value):
Updates a specific parameter.
Pass Arrays as Pointers: Pass parameter arrays as pointers.*/
#include<stdio.h>
double compute_link_budget(const double *parameters, int size);
void update_parameters(double *parameters, int index, double value);
void main()
{
    double parameters[5]={10,-24,36,98,-28};
    int size=5;
    double *const p=parameters;
    for(int i=0;i<size;i++)
    printf("parameter %d is %.2f\n",i,parameters[i]);
    compute_link_budget(p,size);
    update_parameters(p,3,60);
     for(int i=0;i<size;i++)
    printf("parameter %d is %.2f\n",i,parameters[i]);

}
double compute_link_budget(const double *parameters, int size)
{
    double sum=0;
    for(int i=0;i<size;i++)
    sum+=parameters[i];
    printf("Total link budget is %.2f",sum);
}
void update_parameters(double *parameters, int index, double value)
{
    parameters[index]=value;
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ;
parameter 0 is 10.00
parameter 1 is -24.00
parameter 2 is 36.00
parameter 3 is 98.00
parameter 4 is -28.00
Total link budget is 92.00parameter 0 is 10.00
parameter 1 is -24.00
parameter 2 is 36.00
parameter 3 is 60.00
parameter 4 is -28.00
PS D:\projects\quest\C>
```

```c
/*Pointers: Traverse acceleration arrays.
Arrays: Store acceleration data from sensors.
Functions:
void detect_turbulence(const double *accelerations, int size, double
*output): Detects turbulence based on frequency analysis.
void log_turbulence(double *turbulence_log, const double
*detection_output, int size): Logs detected turbulence events.
Pass Arrays as Pointers: Pass acceleration and log arrays to functions.*/
#include <stdio.h>
void detect_turbulence(const double *accelerations, int size, double
*output);
void log_turbulence(double *turbulence_log, const double
*detection_output, int size);
int main() {
    double accelerations[10] = {10, 20, 30, 40, 50, 70, 60, 40, 60, 70};
    double turbulence[10] = {0};
    double detection_output[10] = {0};
    int size = 10;
    detect_turbulence(accelerations, size, detection_output);
    log_turbulence(turbulence, detection_output, size);
    printf("Acceleration Data:\n");
    for (int i = 0; i < size; i++) {
```

```
        printf("%.2f ", accelerations[i]);
    }
    printf("\n\nTurbulence Detected:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2f ", detection_output[i]);
    }
    printf("\n\nTurbulence Log:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2f ", turbulence[i]);
    }
    printf("\n");

    return 0;
}
void detect_turbulence(const double *accelerations, int size, double
*output) {
    for (int i = 0; i < size; i++) {
        if (accelerations[i] > 50.0) {
            output[i] = accelerations[i];
        } else {
            output[i] = 0.0;
        }
    }
}

void log_turbulence(double *turbulence_log, const double
*detection_output, int size) {
    for (int i = 0; i < size; i++) {
        if (detection_output[i] > 0.0) {
            turbulence_log[i] = detection_output[i];
        } else {
            turbulence_log[i] = 0.0;
        }
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($?) { gcc t
Acceleration Data:
10.00 20.00 30.00 40.00 50.00 70.00 60.00 40.00 60.00 70.00

Turbulence Detected:
0.00 0.00 0.00 0.00 0.00 70.00 60.00 0.00 60.00 70.00

Turbulence Log:
0.00 0.00 0.00 0.00 0.00 70.00 60.00 0.00 60.00 70.00
PS D:\projects\quest\C>
```