Structure Function

Set 1:

Problem 1: Vehicle Fleet Management System

Requirements:

Create a structure Vehicle with the following members:

char registrationNumber[15]

char model[30]

int yearOfManufacture

float mileage

float fuelEfficiency

Implement functions to:

Add a new vehicle to the fleet.

Update the mileage and fuel efficiency for a vehicle.

Display all vehicles manufactured after a certain year.

Find the vehicle with the highest fuel efficiency.

Use dynamic memory allocation to manage the fleet of vehicles.

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Vehicle {

   char registrationNumber[15];

   char model[30];

   int yearOfManufacture;

   float mileage;

   float fuelEfficiency;

};
```

```
void addNew(struct Vehicle** fleet, int* size);
void update(struct Vehicle* fleet, int size, char* reg);
void display(struct Vehicle* fleet, int size, int year);
void search(struct Vehicle* fleet, int size);

int main() {
    struct Vehicle* fleet = NULL;
    int size = 0;
    int choice;
    char reg[15];
    int year;

    while (1) {
        printf("\n Vehicle Fleet Management System \n");
        printf("1 => Add new vehicle\n");
        printf("2 => Update details\n");
        printf("3 => Display vehicle manufactured after a certain year\n");
        printf("4 => Find the vehicle with highest fuel efficiency\n");
        printf("5 => Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addNew(&fleet, &size);
                break;
            case 2:
                printf("Enter the registration number to be updated: ");
                scanf("%s", reg);
                update(fleet, size, reg);
                break;
            case 3:
                printf("Enter year: ");
                scanf("%d", &year);
                display(fleet, size, year);
                break;
            case 4:
                search(fleet, size);
                break;
            case 5:
                printf("------Exiting ------------\n");
                free(fleet);
                printf("Memory dellocated");
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

void addNew(struct Vehicle** fleet, int* size) {
    (*size)++;
    *fleet = realloc(*fleet, (*size) * sizeof(struct Vehicle));
    if (*fleet == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    printf("Enter the Registration number: ");
    scanf("%s", (*fleet)[*size - 1].registrationNumber);
    printf("Enter model: ");
    scanf("%s", (*fleet)[*size - 1].model);
    printf("Enter year of manufacture: ");
    scanf("%d", &(*fleet)[*size - 1].yearOfManufacture);
    printf("Enter mileage: ");
    scanf("%f", &(*fleet)[*size - 1].mileage);
    printf("Enter fuel efficiency: ");
    scanf("%f", &(*fleet)[*size - 1].fuelEfficiency);
}

void update(struct Vehicle* fleet, int size, char* reg) {
    for (int i = 0; i < size; i++) {
        if (strcmp(fleet[i].registrationNumber, reg) == 0) {
            printf("Enter new mileage: ");
            scanf("%f", &fleet[i].mileage);
            printf("Enter new fuel efficiency: ");
            scanf("%f", &fleet[i].fuelEfficiency);
            return;
        }
    }
    printf("Vehicle not found!\n");
}

void display(struct Vehicle* fleet, int size, int year) {
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (fleet[i].yearOfManufacture > year) {
            printf("\nRegistration Number: %s\n", fleet[i].registrationNumber);
            printf("Model: %s\n", fleet[i].model);
            printf("Year of Manufacture: %d\n", fleet[i].yearOfManufacture);
            printf("Mileage: %.2f\n", fleet[i].mileage);
            printf("Fuel Efficiency: %.2f\n", fleet[i].fuelEfficiency);
            count = 1;
        }
    }
    if (!count) {
        printf("No vehicles found manufactured after %d.\n", year);
    }
}

void search(struct Vehicle* fleet, int size) {
    if (size == 0) {
        printf("No vehicles in the fleet.\n");
        return;
    }

    int best = 0;
    for (int i = 1; i < size; i++) {
        if (fleet[i].fuelEfficiency > fleet[best].fuelEfficiency) {
            best = i;
        }
    }

    printf("\nVehicle with the highest fuel efficiency:\n");
    printf("Registration Number: %s\n", fleet[best].registrationNumber);
    printf("Model: %s\n", fleet[best].model);
    printf("Year of Manufacture: %d\n", fleet[best].yearOfManufacture);
    printf("Mileage: %.2f\n", fleet[best].mileage);
    printf("Fuel Efficiency: %.2f\n", fleet[best].fuelEfficiency);
}
```

Problem 2: Car Rental Reservation System

Requirements:

Define a structure CarRental with members:

char carID[10]

char customerName[50]

char rentalDate[11] (format: YYYY-MM-DD)

char returnDate[11]

float rentalPricePerDay

Write functions to:

Book a car for a customer by inputting necessary details.

Calculate the total rental price based on the number of rental days.

Display all current rentals.

Search for rentals by customer name.

Implement error handling for invalid dates and calculate the number of rental days.

```
#include <stdio.h>

#include <string.h>

#include <time.h>


struct Car {

    char carID[10];

    char customerName[50];

    char rentalDate[11];

    char returnDate[11];

    float rentalPricePerDay;

};


void Bookcar(struct Car *rent, int *count);

int calculateRentalDays(char *rentalDate, char *returnDate);

void display(struct Car *rent, int count);

int totalPrice(int rentalDays, float pricePerDay);

void search(struct Car *rent, int count, char *customerName);
```

```
int isValid(char *date);

int main() {
    struct Car rent[100];
    int count = 0;
    int choice;
    char customerName[50];

    while (1) {
        printf("\nCar Reservation System\n");
        printf("1 => Book a Car\n");
        printf("2 => Display All Rentals\n");
        printf("3 => Search Rentals by Customer Name\n");
        printf("4 => Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                Bookcar(rent, &count);
                break;
            case 2:
                display(rent, count);
                break;
            case 3:
                printf("Enter Customer name: ");
                scanf("%s", customerName);
                search(rent, count, customerName);
                break;
            case 4:
                printf("======== Exiting ========\n");
            return 0;
        default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}

void Bookcar(struct Car *rent, int *count) {
    printf("\nEnter car ID: ");
    scanf("%s", rent[*count].carID);
    printf("Enter the customer name: ");
    scanf("%s", rent[*count].customerName);

    while (1) {
        printf("Enter rental date (YYYY-MM-DD): ");
        scanf("%s", rent[*count].rentalDate);
        if (isValid(rent[*count].rentalDate))
            break;
        else
            printf("Invalid rental date format. Please try again.\n");
    }

    while (1) {
        printf("Enter return date (YYYY-MM-DD): ");
        scanf("%s", rent[*count].returnDate);
        if (isValid(rent[*count].returnDate))
            break;
        else
            printf("Invalid return date format. Please try again.\n");
    }
    printf("Enter rental price per day: ");
    scanf("%f", &rent[*count].rentalPricePerDay);

    int rentalDays = calculateRentalDays(rent[*count].rentalDate, rent[*count].returnDate);
    if (rentalDays < 0) {
        printf("Error: Return date cannot be earlier than rental date.\n");
    } else {
        float total = totalPrice(rentalDays, rent[*count].rentalPricePerDay);
        printf("Total rental price: %.2f\n", total);
    }
    (*count)++;
}

int calculateRentalDays(char *rentalDate, char *returnDate) {
    int rentYear, rentMonth, rentDay;
    int returnYear, returnMonth, returnDay;

    sscanf(rentalDate, "%d-%d-%d", &rentYear, &rentMonth, &rentDay);
    sscanf(returnDate, "%d-%d-%d", &returnYear, &returnMonth, &returnDay);

    struct tm rent = {0}, returnD = {0};
    rent.tm_year = rentYear - 1900; // tm_year is year since 1900
    rent.tm_mon = rentMonth - 1;    // tm_mon is month 0-11
    rent.tm_mday = rentDay;

    returnD.tm_year = returnYear - 1900;
    returnD.tm_mon = returnMonth - 1;
    returnD.tm_mday = returnDay;

    time_t rentTime = mktime(&rent);
    time_t returnTime = mktime(&returnD);

    if (rentTime == -1 || returnTime == -1) {
        return -1;  // Invalid date
    }

    double diff = difftime(returnTime, rentTime);
    return diff / (60 * 60 * 24);  // Return the difference in days
}

void display(struct Car *rent, int count) {
    for (int i = 0; i < count; i++) {
        printf("\nCar ID: %s\n", rent[i].carID);
        printf("Customer Name: %s\n", rent[i].customerName);
        printf("Rental Date: %s\n", rent[i].rentalDate);
        printf("Return Date: %s\n", rent[i].returnDate);
        printf("Rental Price per Day: %.2f\n", rent[i].rentalPricePerDay);
    }
}

int totalPrice(int rentalDays, float pricePerDay) {
    return rentalDays * pricePerDay;
}

void search(struct Car *rent, int count, char *customerName) {
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(rent[i].customerName, customerName) == 0) {
            printf("\nCar ID: %s\n", rent[i].carID);
            printf("Customer Name: %s\n", rent[i].customerName);
            printf("Rental Date: %s\n", rent[i].rentalDate);
            printf("Return Date: %s\n", rent[i].returnDate);
            printf("Rental Price per Day: %.2f\n", rent[i].rentalPricePerDay);
            found = 1;
        }
    }
    if (!found) {
        printf("No rentals found for customer: %s\n", customerName);
    }
}

int isValid(char *date) {
    int year, month, day;
    if (sscanf(date, "%d-%d-%d", &year, &month, &day) != 3) {
        return 0;
    }
    if (month < 1 || month > 12 || day < 1 || day > 31) {
        return 0;
    }
    return 1;
}
```

Problem 3: Autonomous Vehicle Sensor Data Logger

Requirements:

Create a structure SensorData with fields:

int sensorID

char timestamp[20] (format: YYYY-MM-DD HH:MM:SS)

float speed

float latitude

float longitude

Functions to:

Log new sensor data.

Display sensor data for a specific time range.

Find the maximum speed recorded.

Calculate the average speed over a specific time period.

Store sensor data in a dynamically allocated array and resize it as needed.

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

// Structure to store sensor data
struct SensorData {
    int sensorID;
    char timestamp[20];  // (format: YYYY-MM-DD HH:MM:SS)
    float speed;
    float latitude;
    float longitude;
};

// Function declarations
void logSensorData(struct SensorData **data, int *count, int *size);
void display(struct SensorData *data, int count, char *startTime, char *endTime);
float maxSpeed(struct SensorData *data, int count);
float averageSpeed(struct SensorData *data, int count, char *startTime, char *endTime);
void resizeArray(struct SensorData **data, int *size);
int isValidTimestamp(char *timestamp);

int main() {
    struct SensorData *data = (struct SensorData *)malloc(10 * sizeof(struct SensorData));
    int count = 0;
    int size = 10;
    int choice;
```

```
while (1) {
    printf("\nAutonomous Vehicle Sensor Data Logger\n");
    printf("1 => Log New Sensor Data\n");
    printf("2 => Display Sensor Data for a Specific Time Range\n");
    printf("3 => Find Maximum Speed Recorded\n");
    printf("4 => Calculate Average Speed Over a Time Period\n");
    printf("5 => Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    // Clear the newline left by scanf
    while (getchar() != '\n');

    switch (choice) {
        case 1:
            logSensorData(&data, &count, &size);  // Passing reference of data
            break;
        case 2: {
            char startTime[20], endTime[20];
            // Inside the case for displaying data
            printf("Enter Start Time (YYYY-MM-DD HH:MM:SS): ");
            fgets(startTime, 20, stdin);
            startTime[strcspn(startTime, "\n")] = '\0';  // Remove newline character

            // Clear the input buffer to handle any residual characters after fgets
            while(getchar() != '\n');  // This ensures the buffer is cleared before reading the next input

            printf("Enter End Time (YYYY-MM-DD HH:MM:SS): ");
            fgets(endTime, 20, stdin);
            endTime[strcspn(endTime, "\n")] = '\0';  // Remove newline character
            // Proceed with display function
            display(data, count, startTime, endTime);


            break;
          }
          case 3:
            printf("Maximum Speed Recorded: %.2f\n", maxSpeed(data, count));
            break;
          case 4: {
            char startTime[20], endTime[20];
            printf("Enter Start Time (YYYY-MM-DD HH:MM:SS): ");
            fgets(startTime, 20, stdin);
            startTime[strcspn(startTime, "\n")] = '\0';  // Remove newline character
            printf("Enter End Time (YYYY-MM-DD HH:MM:SS): ");
            fgets(endTime, 20, stdin);
            endTime[strcspn(endTime, "\n")] = '\0';  // Remove newline character
            printf("Average Speed between %s and %s: %.2f\n", startTime, endTime,
averageSpeed(data, count, startTime, endTime));
            break;
          }
          case 5:
            printf("======== Exiting =================\n");
            free(data);
            return 0;
          default:
            printf("Invalid choice.\n");
        }
    }

    return 0;
```

```
}

// Function to validate the timestamp format
int isValidTimestamp(char *timestamp) {
    int y, m, d, h, min, sec;

    // Ensure there are no extra newline characters
    timestamp[strcspn(timestamp, "\n")] = '\0';  // Remove newline character

    int result = sscanf(timestamp, "%d-%d-%d %d:%d:%d", &y, &m, &d, &h, &min, &sec);
      // Debugging line to check the sscanf result

    if (result != 6) {
        printf("Invalid timestamp format: %s\n", timestamp);  // Debugging line to print invalid
timestamp
        return 0;  // Invalid format
    }

    // Validate the ranges for each part of the timestamp
    if (m < 1 || m > 12 || d < 1 || d > 31 || h < 0 || h > 23 || min < 0 || min > 59 || sec < 0 || sec > 59) {
        printf("Invalid range: %d-%d-%d %d:%d:%d\n", y, m, d, h, min, sec);  // Debugging line to print
invalid range
        return 0;  // Invalid values in the timestamp
    }

    // Validate days in February, considering leap years
    if (m == 2) {
        int isLeapYear = (y % 4 == 0 && (y % 100 != 0 || y % 400 == 0));
        if ((isLeapYear && d > 29) || (!isLeapYear && d > 28)) {
            printf("Invalid day for February: %d\n", d);  // Debugging line to print invalid day
            return 0;  // Invalid day for February
        }
    }

    // Validate months with 30 days
    if ((m == 4 || m == 6 || m == 9 || m == 11) && d > 30) {
        printf("Invalid day for month %d: %d\n", m, d);  // Debugging line to print invalid day for
months with 30 days
        return 0;  // Invalid day for a month with 30 days
    }

    return 1;  // Valid timestamp
}

// Function to log new sensor data
void logSensorData(struct SensorData **data, int *count, int *size) {
    if (*count >= *size) {
        resizeArray(data, size);  // Pass the pointer to update it
    }

    printf("Enter Sensor ID: ");
    scanf("%d", &(*data)[*count].sensorID);

    // Clear the newline left by scanf
    while (getchar() != '\n');

    // Validate timestamp format
    while (1) {
        printf("Enter Timestamp (YYYY-MM-DD HH:MM:SS): ");
        fgets((*data)[*count].timestamp, 20, stdin);
        (*data)[*count].timestamp[strcspn((*data)[*count].timestamp, "\n")] = '\0';  // Remove newline
character
```

```
        if (isValidTimestamp((*data)[*count].timestamp)) break;
        else printf("Invalid timestamp format. Please try again.\n");
    }

    printf("Enter Speed: ");
    scanf("%f", &(*data)[*count].speed);

    printf("Enter Latitude: ");
    scanf("%f", &(*data)[*count].latitude);

    printf("Enter Longitude: ");
    scanf("%f", &(*data)[*count].longitude);

    (*count)++;
}

// Function to display sensor data for a specific time range
void display(struct SensorData *data, int count, char *startTime, char *endTime) {
    // Check if endTime is empty or invalid
    if (endTime == NULL || strlen(endTime) == 0) {
        printf("Error: End Time is empty. Please enter a valid end time.\n");
        return;
    }

    printf("Displaying Sensor Data between %s and %s:\n", startTime, endTime);

    for (int i = 0; i < count; i++) {
        // Compare the timestamps with the given range
        if (strcmp(data[i].timestamp, startTime) >= 0 && strcmp(data[i].timestamp, endTime) <= 0) {
            printf("Sensor ID: %d\n", data[i].sensorID);
            printf("Timestamp: %s\n", data[i].timestamp);
            printf("Speed: %.2f\n", data[i].speed);
            printf("Latitude: %.2f\n", data[i].latitude);
            printf("Longitude: %.2f\n", data[i].longitude);
        }
    }
    printf("No sensors present ");
}

// Function to find the maximum speed recorded
float maxSpeed(struct SensorData *data, int count) {
    if (count == 0) {
        printf("No data available.\n");
        return -1.0;  // Return an error value
    }

    float max = data[0].speed;
    for (int i = 1; i < count; i++) {
        if (data[i].speed > max) {
            max = data[i].speed;
        }
    }
    return max;
}

// Function to calculate the average speed over a time period
float averageSpeed(struct SensorData *data, int count, char *startTime, char *endTime) {
    float totalSpeed = 0.0;
    int validCount = 0;

    for (int i = 0; i < count; i++) {
        if (strcmp(data[i].timestamp, startTime) >= 0 && strcmp(data[i].timestamp, endTime) <= 0) {
            totalSpeed += data[i].speed;
            validCount++;
        }
    }

    if (validCount == 0) {
        printf("No data found for the specified time range.\n");
        return -1.0;  // Return an error value
    }

    return totalSpeed / validCount;
}

// Function to resize the array if needed
void resizeArray(struct SensorData **data, int *size) {
    *size *= 2;  // Double the size
    *data = (struct SensorData *)realloc(*data, (*size) * sizeof(struct SensorData));
    if (*data == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}
```

Problem 4: Engine Performance Monitoring System

Requirements:

Define a structure EnginePerformance with members:

char engineID[10]

float temperature

float rpm

float fuelConsumptionRate

float oilPressure

Functions to:

Add performance data for a specific engine.

Display all performance data for a specific engine ID.

Calculate the average temperature and RPM for a specific engine.

Identify any engine with abnormal oil pressure (above or below specified thresholds).

Use array to store and manage performance data entries.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Define the EnginePerformance structure

struct EnginePerformance {

    char engineID[10];

    float temperature;

    float rpm;

    float fuelConsumptionRate;

    float oilPressure;

};


// Function declarations

void addEnginePerformance(struct EnginePerformance *data, int *count);

void displayEnginePerformance(struct EnginePerformance *data, int count, char *engineID);

void calculateAverage(struct EnginePerformance *data, int count, char *engineID);

void identifyAbnormalOilPressure(struct EnginePerformance *data, int count, float minOilPressure,
float maxOilPressure);

void resizeArray(struct EnginePerformance **data, int *size);


int main() {

    int count = 0;
    int size = 10;
    struct EnginePerformance *data = (struct EnginePerformance *)malloc(size * sizeof(struct
EnginePerformance));
    int choice;

    while (1) {
        printf("\nEngine Performance Monitoring System\n");
        printf("1 => Add Engine Performance Data\n");
        printf("2 => Display Engine Performance Data for a Specific Engine ID\n");
        printf("3 => Calculate Average Temperature and RPM for a Specific Engine\n");
        printf("4 => Identify Engines with Abnormal Oil Pressure\n");
        printf("5 => Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addEnginePerformance(data, &count);
                break;
            case 2: {
                char engineID[10];
                printf("Enter Engine ID: ");
                scanf("%s", engineID);
                displayEnginePerformance(data, count, engineID);
                break;
            }
            case 3: {
                char engineID[10];
                printf("Enter Engine ID: ");
                scanf("%s", engineID);
                calculateAverage(data, count, engineID);
                break;
            }
            case 4: {
                float minOilPressure, maxOilPressure;
                printf("Enter minimum oil pressure threshold: ");
                scanf("%f", &minOilPressure);
                printf("Enter maximum oil pressure threshold: ");
                scanf("%f", &maxOilPressure);
                identifyAbnormalOilPressure(data, count, minOilPressure, maxOilPressure);
                break;
            }
            case 5:
                printf("======== Exiting =================\n");
                free(data);
                return 0;
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}

// Function to add performance data for a specific engine
void addEnginePerformance(struct EnginePerformance *data, int *count) {
    if (*count >= 10) {
        printf("Maximum data entries reached.\n");
        return;
    }

    printf("Enter Engine ID: ");
    scanf("%s", data[*count].engineID);

    printf("Enter Temperature: ");
    scanf("%f", &data[*count].temperature);

    printf("Enter RPM: ");
    scanf("%f", &data[*count].rpm);

    printf("Enter Fuel Consumption Rate: ");
    scanf("%f", &data[*count].fuelConsumptionRate);

    printf("Enter Oil Pressure: ");
    scanf("%f", &data[*count].oilPressure);

    (*count)++;
}

// Function to display all performance data for a specific engine ID
void displayEnginePerformance(struct EnginePerformance *data, int count, char *engineID) {
    printf("Displaying performance data for engine %s:\n", engineID);

    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(data[i].engineID, engineID) == 0) {
            printf("Temperature: %.2f\n", data[i].temperature);
            printf("RPM: %.2f\n", data[i].rpm);
            printf("Fuel Consumption Rate: %.2f\n", data[i].fuelConsumptionRate);
            printf("Oil Pressure: %.2f\n", data[i].oilPressure);
            found = 1;
        }
    }
    if (!found) {
        printf("Engine ID %s not found.\n", engineID);
    }
}

// Function to calculate the average temperature and RPM for a specific engine
void calculateAverage(struct EnginePerformance *data, int count, char *engineID) {
    float totalTemperature = 0.0, totalRPM = 0.0;
    int validEntries = 0;

    for (int i = 0; i < count; i++) {
        if (strcmp(data[i].engineID, engineID) == 0) {
            totalTemperature += data[i].temperature;
            totalRPM += data[i].rpm;
            validEntries++;
        }
    }

    if (validEntries > 0) {
        printf("Average Temperature: %.2f\n", totalTemperature / validEntries);
        printf("Average RPM: %.2f\n", totalRPM / validEntries);
    } else {
        printf("Engine ID %s not found.\n", engineID);
    }
}

// Function to identify engines with abnormal oil pressure
void identifyAbnormalOilPressure(struct EnginePerformance *data, int count, float minOilPressure,
float maxOilPressure) {
    printf("Identifying engines with abnormal oil pressure:\n");
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (data[i].oilPressure < minOilPressure || data[i].oilPressure > maxOilPressure) {
            printf("Engine ID: %s\n", data[i].engineID);
            printf("Oil Pressure: %.2f (Abnormal)\n", data[i].oilPressure);
            found = 1;
        }
    }

    if (!found) {
        printf("No engines with abnormal oil pressure found.\n");
    }
}
```

Problem 5: Vehicle Service History Tracker

Requirements:

Create a structure ServiceRecord with the following:

char serviceID[10]

char vehicleID[15]

char serviceDate[11]

char description[100]

float serviceCost

Functions to:

Add a new service record for a vehicle.

Display all service records for a given vehicle ID.

Calculate the total cost of services for a vehicle.

Sort and display service records by service date.

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

```
#include <stdlib.h>
#include <string.h>

struct ServiceRecord {
    char serviceID[10];
    char vehicleID[15];
    char serviceDate[11];  // Date in YYYY-MM-DD format
    char description[100];
    float serviceCost;
};

void addServiceRecord(struct ServiceRecord *data, int *count);
void displayServiceRecords(struct ServiceRecord *data, int count, char *vehicleID);
void calculateTotalCost(struct ServiceRecord *data, int count, char *vehicleID);
void sortServiceRecordsByDate(struct ServiceRecord *data, int count);

int main() {
    int count = 0;
    int size = 10;
    struct ServiceRecord *data = (struct ServiceRecord *)malloc(size * sizeof(struct ServiceRecord));
    int choice;

    while (1) {
        printf("\nVehicle Service History Tracker\n");
        printf("1 => Add Service Record\n");
        printf("2 => Display Service Records for a Specific Vehicle ID\n");
        printf("3 => Calculate Total Service Cost for a Vehicle\n");
        printf("4 => Sort and Display Service Records by Service Date\n");
        printf("5 => Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
    switch (choice) {
        case 1:
            addServiceRecord(data, &count);
            break;
        case 2: {
            char vehicleID[15];
            printf("Enter Vehicle ID: ");
            scanf("%s", vehicleID);
            displayServiceRecords(data, count, vehicleID);
            break;
        }
        case 3: {
            char vehicleID[15];
            printf("Enter Vehicle ID: ");
            scanf("%s", vehicleID);
            calculateTotalCost(data, count, vehicleID);
            break;
        }
        case 4:
            sortServiceRecordsByDate(data, count);
            break;
        case 5:
            printf("======== Exiting =================\n");
            free(data);
            return 0;
        default:
            printf("Invalid choice.\n");
    }
}
```

```
    return 0;
}

void addServiceRecord(struct ServiceRecord *data, int *count) {
    if (*count >= 10) {
        printf("Maximum number of records reached.\n");
        return;
    }

    printf("Enter Service ID: ");
    scanf("%s", data[*count].serviceID);

    printf("Enter Vehicle ID: ");
    scanf("%s", data[*count].vehicleID);

    printf("Enter Service Date (YYYY-MM-DD): ");
    scanf("%s", data[*count].serviceDate);

    printf("Enter Service Description: ");
    scanf(" %[^\n]s", data[*count].description);  // To read spaces in description

    printf("Enter Service Cost: ");
    scanf("%f", &data[*count].serviceCost);

    (*count)++;
}

void displayServiceRecords(struct ServiceRecord *data, int count, char *vehicleID) {
    printf("Displaying service records for vehicle %s:\n", vehicleID);

    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(data[i].vehicleID, vehicleID) == 0) {
            printf("\nService ID: %s\n", data[i].serviceID);
            printf("Service Date: %s\n", data[i].serviceDate);
            printf("Description: %s\n", data[i].description);
            printf("Service Cost: %.2f\n", data[i].serviceCost);
            found = 1;
        }
    }

    if (!found) {
        printf("No service records found for vehicle ID %s.\n", vehicleID);
    }
}

void calculateTotalCost(struct ServiceRecord *data, int count, char *vehicleID) {
    float totalCost = 0.0;

    for (int i = 0; i < count; i++) {
        if (strcmp(data[i].vehicleID, vehicleID) == 0) {
            totalCost += data[i].serviceCost;
        }
    }

    printf("Total service cost for vehicle ID %s: %.2f\n", vehicleID, totalCost);
}

int compareDates(const void *a, const void *b) {
    return strcmp(((struct ServiceRecord *)a)->serviceDate, ((struct ServiceRecord *)b)->serviceDate);
}
```

```
void sortServiceRecordsByDate(struct ServiceRecord *data, int count) {
    qsort(data, count, sizeof(struct ServiceRecord), compareDates);

    printf("Service records sorted by service date:\n");
    for (int i = 0; i < count; i++) {
        printf("\nService ID: %s\n", data[i].serviceID);
        printf("Vehicle ID: %s\n", data[i].vehicleID);
        printf("Service Date: %s\n", data[i].serviceDate);
        printf("Description: %s\n", data[i].description);
        printf("Service Cost: %.2f\n", data[i].serviceCost);
    }
}
```

Set 2:

Problem 1: Player Statistics Management

Requirements:

- Define a structure Player with the following members:

o char name[50]

o int age

o char team[30]

o int matchesPlayed

o int totalRuns

o int totalWickets

- Functions to:
- Add a new player to the system.
- Update a player’s statistics after a match.
- Display the details of players from a specific team.
- Find the player with the highest runs and the player with the most wickets.

• Use dynamic memory allocation to store player data in an array and expand it as needed.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Player{

   char name[50];

   int age;

   char team[30];

   int matchesPlayed;

   int totalRuns;

   int totalWickets;


};


void addNew(struct Player **player,int *count, int *limit);

void update(struct Player *player,int count);

void display(struct Player *player,int count,char *team);

void highest(struct Player *player,int count);


int main(){


   struct Player *player =NULL;

   int count =0,limit =2;

   int choice;

   player = (struct Player *)malloc(limit * sizeof(struct Player));
```

```
while(1){
     printf("\n--- Player Statistics Management ---\n");
    printf("1. Add New Player\n");
    printf("2. Update Player Statistics\n");
    printf("3. Display Players by Team\n");
    printf("4. Find Top Players (Highest Runs & Most Wickets)\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice){
       case 1:
          addNew(&player,&count,&limit);
       break;
       case 2:
          update(player,count);
       break;
       case 3:
       {
          char team[50];
          printf("Enter Team:  ");
          scanf("%s",team);
          display(player,count,team);
       break;
       }
       case 4:
          highest(player,count);
```

```
            break;
            case 5:
               printf("======Exiting==========");
               return 0;
            break;
            default:
            printf("Invalid choice ");

        }

    }
    free(player);
    return 0;
}

void addNew(struct Player **player, int *count, int *limit) {
    if (*count == *limit) {
        *limit *= 2;
        *player = (struct Player *)realloc(*player, *limit * sizeof(struct Player));
        if (*player == NULL) {
            printf("Memory allocation failed.\n");
            exit(1);
        }
    }

    printf("Enter the player details:\n");
    printf("Name: ");
    scanf("%s", (*player)[*count].name);
    printf("Age: ");
    scanf("%d", &(*player)[*count].age);
    printf("Team: ");
    scanf("%s", (*player)[*count].team);
    printf("Matches Played: ");
    scanf("%d", &(*player)[*count].matchesPlayed);
    printf("Total Runs: ");
    scanf("%d", &(*player)[*count].totalRuns);
    printf("Total Wickets: ");
    scanf("%d", &(*player)[*count].totalWickets);

    (*count)++;
    printf("Player added successfully.\n");
}

void update(struct Player *player,int count){
    char name[100];
    printf("Enter the player name to be updated ");
    scanf("%s",name);
    for(int i=0;i<count;i++){
        if(strcmp(player[i].name,name)==0){
            printf("Enter the new Statistics: \n");
            printf("Matched played: ");
            scanf("%d",&player[i].matchesPlayed);
            printf("Total Runs : ");
            scanf("%d",&player[i].totalRuns);
            printf("total wickets : ");
            scanf("%d",&player[i].totalWickets);
            return;
        }
    }
    printf("Player not found\n");


}
void display(struct Player *player,int count,char *team){
    printf("Player from team %s \n",team);
    for(int i=0;i<count;i++){
        if(strcmp(player[i].team,team)==0){
            printf("Name = %s \n  Age = %d \n Matches playes = %d \n Total runs = %d \n Total
wickets = %d
\n",player[i].name,player[i].age,player[i].matchesPlayed,player[i].totalRuns,player[i].tot
alWickets);


        }
    }
    printf("Team not found \n");


}
void highest(struct Player *player, int count) {
    if (count == 0) {
        printf("No players on board.\n");
        return;
    }


    int highestRunIndex = 0;    // Index of the player with the highest runs
    int highestWicketIndex = 0; // Index of the player with the most wickets
```

```
  for (int i = 1; i < count; i++) { // Start from 1 because 0 is already assumed
    if (player[i].totalRuns > player[highestRunIndex].totalRuns) {
      highestRunIndex = i;
    }
    if (player[i].totalWickets > player[highestWicketIndex].totalWickets) {
      highestWicketIndex = i;
    }
  }

  printf("\nPlayer with the highest runs:\n");
  printf("Name: %s, Runs: %d\n", player[highestRunIndex].name,
player[highestRunIndex].totalRuns);

  printf("\nPlayer with the most wickets:\n");
  printf("Name: %s, Wickets: %d\n", player[highestWicketIndex].name,
player[highestWicketIndex].totalWickets);
}
```

Problem 2: Tournament Fixture Scheduler

Requirements:

- Create a structure Match with members:
  - o char team1[30]
  - o char team2[30]
  - o char date[11] (format: YYYY-MM-DD)
  - o char venue[50]
- Functions to:

- Schedule a new match between two teams.
- Display all scheduled matches.
- Search for matches scheduled on a specific date.
- Cancel a match by specifying both team names and the date.
- Ensure that the match schedule is stored in an array, with the ability to dynamically adjust its size.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Match {

   char team1[30];

   char team2[30];

   char date[11];

   char venue[50];

};


// Function prototypes

void schedule(struct Match **match, int *count, int *limit);

void display(struct Match *match, int count);

void search(struct Match *match, int count, char *date);

void cancel(struct Match **match, int *count, char *team1, char *team2, char *date);


int main() {

   struct Match *match = NULL;

   int count = 0, limit = 2;

   match = (struct Match *)malloc(limit * sizeof(struct Match));
```

```
int choice;

while (1) {
    printf("\n--- Tournament Fixture Scheduler ---\n");
    printf("1. Schedule a New Match\n");
    printf("2. Display All Scheduled Matches\n");
    printf("3. Search Matches by Date\n");
    printf("4. Cancel a Match\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            schedule(&match, &count, &limit);
            break;
        case 2:
            display(match, count);
            break;
        case 3: {
            char date[11];
            printf("Enter date (YYYY-MM-DD): ");
            scanf("%s", date);
            search(match, count, date);
            break;
        }
        case 4: {
            char team1[30], team2[30], date[11];
            printf("Enter Team 1: ");
            scanf("%s", team1);
            printf("Enter Team 2: ");
            scanf("%s", team2);
            printf("Enter Date (YYYY-MM-DD): ");
            scanf("%s", date);
            cancel(&match, &count, team1, team2, date);
            break;
        }
        case 5:
            free(match);
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  }

  return 0;
}

// Function to schedule a new match
void schedule(struct Match **match, int *count, int *limit) {
  if (*count == *limit) {
    *limit *= 2;
    *match = (struct Match *)realloc(*match, *limit * sizeof(struct Match));
  }
  printf("Enter the details for the new match:\n");
```

```
    printf("Team 1: ");
    scanf("%s", (*match)[*count].team1);
    printf("Team 2: ");
    scanf("%s", (*match)[*count].team2);
    printf("Date (YYYY-MM-DD): ");
    scanf("%s", (*match)[*count].date);
    printf("Venue: ");
    scanf("%s", (*match)[*count].venue);

    (*count)++;
    printf("Match scheduled successfully!\n");
}

// Function to display all scheduled matches
void display(struct Match *match, int count) {
    if (count == 0) {
        printf("No matches scheduled.\n");
        return;
    }

    printf("\nScheduled Matches:\n");
    for (int i = 0; i < count; i++) {
        printf("Match %d:\n", i + 1);
        printf("  Team 1: %s\n", match[i].team1);
        printf("  Team 2: %s\n", match[i].team2);
        printf("  Date: %s\n", match[i].date);
        printf("  Venue: %s\n", match[i].venue);
    }
```

```
}

// Function to search for matches by date
void search(struct Match *match, int count, char *date) {
  int found = 0;
  for (int i = 0; i < count; i++) {
    if (strcmp(match[i].date, date) == 0) {
      printf("Match %d:\n", i + 1);
      printf("  Team 1: %s\n", match[i].team1);
      printf("  Team 2: %s\n", match[i].team2);
      printf("  Venue: %s\n", match[i].venue);
      found = 1;
    }
  }
  if (!found) {
    printf("No matches scheduled on %s.\n", date);
  }
}

// Function to cancel a match
void cancel(struct Match **match, int *count, char *team1, char *team2, char *date) {
  for (int i = 0; i < *count; i++) {
    if (strcmp((*match)[i].team1, team1) == 0 &&
      strcmp((*match)[i].team2, team2) == 0 &&
      strcmp((*match)[i].date, date) == 0) {

      // Shift remaining matches to overwrite the canceled match
      for (int j = i; j < *count - 1; j++) {
        (*match)[j] = (*match)[j + 1];
      }

      (*count)--;
      printf("Match between %s and %s on %s canceled successfully.\n", team1,
team2, date);
      return;
    }
  }
  printf("No such match found to cancel.\n");
}
```

Problem 3: Sports Event Medal Tally

Requirements:

- Define a structure CountryMedalTally with members:

o char country[30]

o int gold

o int silver

o int bronze

- Functions to:
- Add a new country's medal tally.
- Update the medal count for a country.
- Display the medal tally for all countries.
- Find and display the country with the highest number of gold medals.
- Use an array to store the medal tally, and resize the array dynamically as new countries are added.

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct CountryMedalTally {

   char country[30];

   int gold;

   int silver;

   int bronze;

};


void addCountry(struct CountryMedalTally **tally, int *count, int *limit);

void updateMedals(struct CountryMedalTally *tally, int count);

void displayTally(struct CountryMedalTally *tally, int count);

void findHighestGold(struct CountryMedalTally *tally, int count);


int main() {

   struct CountryMedalTally *tally = NULL;

   int count = 0, limit = 2;

   tally = (struct CountryMedalTally *)malloc(limit * sizeof(struct CountryMedalTally));

   int choice;


   while (1) {

      printf("\n--- Sports Event Medal Tally ---\n");

      printf("1. Add Country\n");

      printf("2. Update Medal Count\n");

      printf("3. Display Medal Tally\n");
```

```
        printf("4. Find Country with Highest Gold\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addCountry(&tally, &count, &limit);
                break;
            case 2:
                updateMedals(tally, count);
                break;
            case 3:
                displayTally(tally, count);
                break;
            case 4:
                findHighestGold(tally, count);
                break;
            case 5:
                free(tally);
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

```
void addCountry(struct CountryMedalTally **tally, int *count, int *limit) {
    if (*count == *limit) {
        *limit *= 2;
        *tally = (struct CountryMedalTally *)realloc(*tally, *limit * sizeof(struct
CountryMedalTally));
    }
    printf("Enter country name: ");
    scanf("%s", (*tally)[*count].country);
    printf("Enter gold, silver, bronze medals: ");
    scanf("%d %d %d", &(*tally)[*count].gold, &(*tally)[*count].silver,
&(*tally)[*count].bronze);
    (*count)++;
    printf("Country added successfully\n");
}

void updateMedals(struct CountryMedalTally *tally, int count) {
    char country[30];
    printf("Enter country name to update: ");
    scanf("%s", country);
    for (int i = 0; i < count; i++) {
        if (strcmp(tally[i].country, country) == 0) {
            printf("Enter updated gold, silver, bronze medals: ");
            scanf("%d %d %d", &tally[i].gold, &tally[i].silver, &tally[i].bronze);
            printf("Medal tally updated successfully\n");
            return;
        }
    }
    printf("Country not found\n");
}


void displayTally(struct CountryMedalTally *tally, int count) {
    if (count == 0) {
        printf("No countries in the tally\n");
        return;
    }
    printf("\nMedal Tally:\n");
    for (int i = 0; i < count; i++) {
        printf("Country: %s, Gold: %d, Silver: %d, Bronze: %d\n",
               tally[i].country, tally[i].gold, tally[i].silver, tally[i].bronze);
    }
}


void findHighestGold(struct CountryMedalTally *tally, int count) {
    if (count == 0) {
        printf("No countries in the tally\n");
        return;
    }
    int maxIndex = 0;
    for (int i = 1; i < count; i++) {
        if (tally[i].gold > tally[maxIndex].gold) {
            maxIndex = i;
        }
    }
    printf("Country with highest gold medals: %s (%d gold)\n",
           tally[maxIndex].country, tally[maxIndex].gold);
}
```

Problem 4: Athlete Performance Tracker

Requirements:

- Create a structure Athlete with fields:
  - char athleteID[10]
  - char name[50]
  - char sport[30]
  - float personalBest
  - float lastPerformance
- Functions to:
- Add a new athlete to the system.
- Update an athlete's last performance.
- Display all athletes in a specific sport.
- Identify and display athletes who have set a new personal best in their last performance.
- Utilize dynamic memory allocation to manage athlete data in an expandable array.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Athlete {

   char athleteID[10];

   char name[50];

   char sport[30];

   float personalBest;

   float lastPerformance;

};
```

```
void addAthlete(struct Athlete **athletes, int *count, int *limit);
void updatePerformance(struct Athlete *athletes, int count);
void displayAthletesBySport(struct Athlete *athletes, int count, char *sport);
void displayNewPersonalBest(struct Athlete *athletes, int count);

int main() {
  struct Athlete *athletes = NULL;
  int count = 0, limit = 2;
  athletes = (struct Athlete *)malloc(limit * sizeof(struct Athlete));
  int choice;

  while (1) {
    printf("\n--- Athlete Performance Tracker ---\n");
    printf("1. Add Athlete\n");
    printf("2. Update Athlete Performance\n");
    printf("3. Display Athletes by Sport\n");
    printf("4. Display Athletes with New Personal Best\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        addAthlete(&athletes, &count, &limit);
        break;
      case 2:
        updatePerformance(athletes, count);
```

```
            break;
        case 3: {
            char sport[30];
            printf("Enter sport: ");
            scanf("%s", sport);
            displayAthletesBySport(athletes, count, sport);
            break;
        }
        case 4:
            displayNewPersonalBest(athletes, count);
            break;
        case 5:
            free(athletes);
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}

void addAthlete(struct Athlete **athletes, int *count, int *limit) {
    if (*count == *limit) {
        *limit *= 2;
        *athletes = (struct Athlete *)realloc(*athletes, *limit * sizeof(struct Athlete));
    }
    printf("Enter athlete ID: ");
    scanf("%s", (*athletes)[*count].athleteID);
    printf("Enter athlete name: ");
    scanf("%s", (*athletes)[*count].name);
    printf("Enter sport: ");
    scanf("%s", (*athletes)[*count].sport);
    printf("Enter personal best: ");
    scanf("%f", &(*athletes)[*count].personalBest);
    (*athletes)[*count].lastPerformance = (*athletes)[*count].personalBest;  // Initial last
performance same as personal best
    (*count)++;
    printf("Athlete added successfully\n");
}

void updatePerformance(struct Athlete *athletes, int count) {
    char athleteID[10];
    printf("Enter athlete ID to update: ");
    scanf("%s", athleteID);
    for (int i = 0; i < count; i++) {
        if (strcmp(athletes[i].athleteID, athleteID) == 0) {
            printf("Enter last performance: ");
            scanf("%f", &athletes[i].lastPerformance);
            if (athletes[i].lastPerformance > athletes[i].personalBest) {
                athletes[i].personalBest = athletes[i].lastPerformance;
                printf("New personal best set!\n");
            }
            printf("Performance updated successfully\n");
            return;
        }
    }
    printf("Athlete not found\n");
}

void displayAthletesBySport(struct Athlete *athletes, int count, char *sport) {
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(athletes[i].sport, sport) == 0) {
            printf("Athlete ID: %s, Name: %s, Sport: %s, Personal Best: %.2f, Last
Performance: %.2f\n",
                   athletes[i].athleteID, athletes[i].name, athletes[i].sport,
                   athletes[i].personalBest, athletes[i].lastPerformance);
            found = 1;
        }
    }
    if (!found) {
        printf("No athletes found in %s sport\n", sport);
    }
}

void displayNewPersonalBest(struct Athlete *athletes, int count) {
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (athletes[i].lastPerformance > athletes[i].personalBest) {
            printf("Athlete ID: %s, Name: %s, Sport: %s, New Personal Best: %.2f\n",
                   athletes[i].athleteID, athletes[i].name, athletes[i].sport,
                   athletes[i].lastPerformance);
            found = 1;
        }
    }
    if (!found) {
        printf("No new personal bests found\n");
    }
}
```

Problem 5: Sports Equipment Inventory System

Requirements:

- Define a structure Equipment with members:
  - o char equipmentID[10]
  - o char name[30]
  - o char category[20] (e.g., balls, rackets)
  - o int quantity
  - o float pricePerUnit
- Functions to:
- Add new equipment to the inventory.
- Update the quantity of existing equipment.
- Display all equipment in a specific category.
- Calculate the total value of equipment in the inventory.
- Store the inventory data in a dynamically allocated array and ensure proper resizing when needed.

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

struct Equipment {
    char equipmentID[10];
    char name[30];
    char category[20];
    int quantity;
    float pricePerUnit;
};

void addEquipment(struct Equipment **inventory, int *count, int *limit);
void updateQuantity(struct Equipment *inventory, int count);
void displayCategory(struct Equipment *inventory, int count, char *category);
float totalValue(struct Equipment *inventory, int count);

int main() {
    struct Equipment *inventory = NULL;
    int count = 0, limit = 2;
    inventory = (struct Equipment *)malloc(limit * sizeof(struct Equipment));
    int choice;

    while (1) {
        printf("\n--- Sports Equipment Inventory System ---\n");
        printf("1. Add New Equipment\n");
        printf("2. Update Equipment Quantity\n");
        printf("3. Display Equipment by Category\n");
        printf("4. Calculate Total Value\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                addEquipment(&inventory, &count, &limit);
                break;
            case 2:
                updateQuantity(inventory, count);
                break;
            case 3: {
                char category[20];
                printf("Enter category: ");
                scanf("%s", category);
                displayCategory(inventory, count, category);
                break;
            }
            case 4:
                printf("Total Value of Inventory: %.2f\n", totalValue(inventory, count));
                break;
            case 5:
                free(inventory);
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

```
void addEquipment(struct Equipment **inventory, int *count, int *limit) {
    if (*count == *limit) {
        *limit *= 2;
        *inventory = (struct Equipment *)realloc(*inventory, *limit * sizeof(struct
Equipment));
    }
    printf("Enter equipment ID: ");
    scanf("%s", (*inventory)[*count].equipmentID);
    printf("Enter equipment name: ");
    scanf("%s", (*inventory)[*count].name);
    printf("Enter category: ");
    scanf("%s", (*inventory)[*count].category);
    printf("Enter quantity: ");
    scanf("%d", &(*inventory)[*count].quantity);
    printf("Enter price per unit: ");
    scanf("%f", &(*inventory)[*count].pricePerUnit);
    (*count)++;
    printf("Equipment added successfully\n");
}

void updateQuantity(struct Equipment *inventory, int count) {
    char equipmentID[10];
    printf("Enter equipment ID to update: ");
    scanf("%s", equipmentID);
    for (int i = 0; i < count; i++) {
        if (strcmp(inventory[i].equipmentID, equipmentID) == 0) {
            printf("Enter new quantity: ");
            scanf("%d", &inventory[i].quantity);
            printf("Quantity updated successfully\n");
            return;
        }
    }
    printf("Equipment not found\n");
}

void displayCategory(struct Equipment *inventory, int count, char *category) {
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(inventory[i].category, category) == 0) {
            printf("ID: %s, Name: %s, Category: %s, Quantity: %d, Price: %.2f\n",
                   inventory[i].equipmentID, inventory[i].name, inventory[i].category,
                   inventory[i].quantity, inventory[i].pricePerUnit);
            found = 1;
        }
    }
    if (!found) {
        printf("No equipment found in category %s\n", category);
    }
}

float totalValue(struct Equipment *inventory, int count) {
    float total = 0;
    for (int i = 0; i < count; i++) {
        total += inventory[i].quantity * inventory[i].pricePerUnit;
    }
    return total;
```

```
}
```

Set 3:

Problem 1: Research Paper Database Management

Requirements:

- Define a structure ResearchPaper with the following members:

o char title[100]

o char author[50]

o char journal[50]

o int year

o char DOI[30]

- Functions to:

- Add a new research paper to the database.

- Update the details of an existing paper using its DOI.

- Display all papers published in a specific journal.

- Find and display the most recent papers published by a specific author.

- Use dynamic memory allocation to store and manage the research papers in an array, resizing it as needed.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct ResearchPaper {

  char title[100];

  char author[50];

  char journal[50];

  int year;
```

```
    char DOI[30];

};

void addPaper(struct ResearchPaper **papers, int *count, int *limit);
void updatePaper(struct ResearchPaper *papers, int count, char *DOI);
void displayByJournal(struct ResearchPaper *papers, int count, char *journal);
void displayByAuthor(struct ResearchPaper *papers, int count, char *author);
void resizePapers(struct ResearchPaper **papers, int *limit);

int main() {
    struct ResearchPaper *papers = NULL;
    int count = 0, limit = 2;
    papers = (struct ResearchPaper *)malloc(limit * sizeof(struct ResearchPaper));
    int choice;

    while (1) {
        printf("\n--- Research Paper Database Management ---\n");
        printf("1. Add New Research Paper\n");
        printf("2. Update Paper Details\n");
        printf("3. Display Papers by Journal\n");
        printf("4. Display Recent Papers by Author\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
  addPaper(&papers, &count, &limit);
  break;
case 2: {
  char DOI[30];
  printf("Enter DOI to update: ");
  scanf("%s", DOI);
  updatePaper(papers, count, DOI);
  break;
}
case 3: {
  char journal[50];
  printf("Enter journal name: ");
  scanf("%s", journal);
  displayByJournal(papers, count, journal);
  break;
}
case 4: {
  char author[50];
  printf("Enter author name: ");
  scanf("%s", author);
  displayByAuthor(papers, count, author);
  break;
}
case 5:
  free(papers);
  printf("Exiting...\n");
  return 0;
default:
```

```
            printf("Invalid choice\n");
        }
    }

    return 0;
}

void addPaper(struct ResearchPaper **papers, int *count, int *limit) {
    if (*count == *limit) {
        resizePapers(papers, limit);
    }

    printf("Enter paper details:\n");
    printf("Title: ");
    scanf(" %[^\n]", (*papers)[*count].title);
    printf("Author: ");
    scanf("%s", (*papers)[*count].author);
    printf("Journal: ");
    scanf("%s", (*papers)[*count].journal);
    printf("Year: ");
    scanf("%d", &(*papers)[*count].year);
    printf("DOI: ");
    scanf("%s", (*papers)[*count].DOI);

    (*count)++;
    printf("Research paper added successfully!\n");
}
```

```
void updatePaper(struct ResearchPaper *papers, int count, char *DOI) {
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].DOI, DOI) == 0) {
            printf("Enter new details:\n");
            printf("Title: ");
            scanf(" %[^\n]", papers[i].title);
            printf("Author: ");
            scanf("%s", papers[i].author);
            printf("Journal: ");
            scanf("%s", papers[i].journal);
            printf("Year: ");
            scanf("%d", &papers[i].year);
            printf("DOI: ");
            scanf("%s", papers[i].DOI);
            printf("Paper updated successfully!\n");
            return;
        }
    }
    printf("Paper with DOI %s not found.\n", DOI);
}

void displayByJournal(struct ResearchPaper *papers, int count, char *journal) {
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].journal, journal) == 0) {
            printf("Title: %s, Author: %s, Year: %d, DOI: %s\n",
                papers[i].title, papers[i].author, papers[i].year, papers[i].DOI);
            found = 1;
        }
    }
    if (!found) {
        printf("No papers found in journal %s.\n", journal);
    }

}

void displayByAuthor(struct ResearchPaper *papers, int count, char *author) {
    int found = 0;
    int latestYear = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0) {
            if (papers[i].year > latestYear) {
                latestYear = papers[i].year;
            }
        }
    }

    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0 && papers[i].year == latestYear) {
            printf("Title: %s, Journal: %s, Year: %d, DOI: %s\n",
                   papers[i].title, papers[i].journal, papers[i].year, papers[i].DOI);
            found = 1;
        }
    }
    if (!found) {
        printf("No recent papers found for author %s.\n", author);
    }
}

void resizePapers(struct ResearchPaper **papers, int *limit) {
    *limit *= 2;
    *papers = (struct ResearchPaper *)realloc(*papers, *limit * sizeof(struct
ResearchPaper));
    printf("Database resized to accommodate more papers.\n");
}
```

Problem 2: Experimental Data Logger

Requirements:

- Create a structure Experiment with members:

o char experimentID[10]

o char researcher[50]

o char startDate[11] (format: YYYY-MM-DD)

o char endDate[11]

o float results[10] (store up to 10 result readings)

- Functions to:
- Log a new experiment.
- Update the result readings of an experiment.
- Display all experiments conducted by a specific researcher.
- Calculate and display the average result for a specific experiment.
- Use a dynamically allocated array for storing experiments and manage resizing as more data is logged.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

```
struct Experiment {
    char experimentID[10];
    char researcher[50];
    char startDate[11];  // Format: YYYY-MM-DD
    char endDate[11];
    float results[10];  // Store up to 10 result readings
};

// Function declarations
void logExperiment(struct Experiment **experiments, int *count, int *limit);
void updateResults(struct Experiment *experiments, int count);
void displayExperiments(struct Experiment *experiments, int count, char *researcher);
void calculateAverage(struct Experiment *experiments, int count, char *experimentID);
void resizeArray(struct Experiment **experiments, int *limit);

int main() {
    struct Experiment *experiments = NULL;
    int count = 0, limit = 2;  // Initial size limit for the dynamic array
    experiments = (struct Experiment *)malloc(limit * sizeof(struct Experiment));

    int choice;
    while (1) {
        printf("\n--- Experimental Data Logger ---\n");
        printf("1. Log a New Experiment\n");
        printf("2. Update Result Readings\n");
        printf("3. Display Experiments by Researcher\n");
        printf("4. Calculate Average Result for an Experiment\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
  case 1:
    logExperiment(&experiments, &count, &limit);
    break;
  case 2:
    updateResults(experiments, count);
    break;
  case 3:
    {
      char researcher[50];
      printf("Enter Researcher Name: ");
      scanf("%s", researcher);
      displayExperiments(experiments, count, researcher);
      break;
    }
  case 4:
    {
      char experimentID[10];
      printf("Enter Experiment ID: ");
      scanf("%s", experimentID);
      calculateAverage(experiments, count, experimentID);
      break;
    }
  case 5:
```

```
            free(experiments);
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice.\n");
        }
    }

    return 0;
}

void logExperiment(struct Experiment **experiments, int *count, int *limit) {
    if (*count == *limit) {
        resizeArray(experiments, limit);
    }

    printf("Enter Experiment ID: ");
    scanf("%s", (*experiments)[*count].experimentID);
    printf("Enter Researcher Name: ");
    scanf("%s", (*experiments)[*count].researcher);
    printf("Enter Start Date (YYYY-MM-DD): ");
    scanf("%s", (*experiments)[*count].startDate);
    printf("Enter End Date (YYYY-MM-DD): ");
    scanf("%s", (*experiments)[*count].endDate);

    printf("Enter up to 10 result readings:\n");
    for (int i = 0; i < 10; i++) {
        printf("Result %d: ", i + 1);
        scanf("%f", &(*experiments)[*count].results[i]);
    }

    (*count)++;
    printf("Experiment logged successfully.\n");
}

void updateResults(struct Experiment *experiments, int count) {
    char experimentID[10];
    printf("Enter Experiment ID to update: ");
    scanf("%s", experimentID);

    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].experimentID, experimentID) == 0) {
            printf("Enter new result readings:\n");
            for (int j = 0; j < 10; j++) {
                printf("Result %d: ", j + 1);
                scanf("%f", &experiments[i].results[j]);
            }
            printf("Results updated successfully.\n");
            return;
        }
    }

    printf("Experiment with ID %s not found.\n", experimentID);
}

void displayExperiments(struct Experiment *experiments, int count, char *researcher) {
    int found = 0;
    printf("Experiments by %s:\n", researcher);

    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].researcher, researcher) == 0) {
            printf("Experiment ID: %s\n", experiments[i].experimentID);
            printf("Start Date: %s\n", experiments[i].startDate);
            printf("End Date: %s\n", experiments[i].endDate);
            printf("Results: ");
            for (int j = 0; j < 10; j++) {
                printf("%0.2f ", experiments[i].results[j]);
            }
            printf("\n");
            found = 1;
        }
    }

    if (!found) {
        printf("No experiments found for researcher %s.\n", researcher);
    }
}

void calculateAverage(struct Experiment *experiments, int count, char *experimentID) {
    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].experimentID, experimentID) == 0) {
            float sum = 0;
            for (int j = 0; j < 10; j++) {
                sum += experiments[i].results[j];
        }
        printf("Average result for Experiment ID %s: %.2f\n", experimentID, sum / 10);
        return;
      }
    }

    printf("Experiment with ID %s not found.\n", experimentID);
}

void resizeArray(struct Experiment **experiments, int *limit) {
    *limit *= 2;
    *experiments = (struct Experiment *)realloc(*experiments, *limit * sizeof(struct
Experiment));
    printf("Array resized to accommodate more experiments.\n");
}
```

Problem 3: Grant Application Tracker

Requirements:

- Define a structure GrantApplication with the following members:

o char applicationID[10]

o char applicantName[50]

o char projectTitle[100]

o float requestedAmount

o char status[20] (e.g., Submitted, Approved, Rejected)

- Functions to:
- Add a new grant application.
- Update the status of an application.
- Display all applications requesting an amount greater than a specified value.

- Find and display applications that are currently "Approved."
- Store the grant applications in a dynamically allocated array, resizing it as necessary.

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

struct GrantApplication {
   char applicationID[10];
   char applicantName[50];
   char projectTitle[100];
   float requestedAmount;
   char status[20];  // e.g., Submitted, Approved, Rejected
};

// Function declarations
void addGrantApplication(struct GrantApplication **applications, int *count, int *limit);
void updateApplicationStatus(struct GrantApplication *applications, int count);
void displayApplicationsByAmount(struct GrantApplication *applications, int count,
float amount);
void displayApprovedApplications(struct GrantApplication *applications, int count);
void resizeArray(struct GrantApplication **applications, int *limit);

int main() {
   struct GrantApplication *applications = NULL;
   int count = 0, limit = 2;  // Initial size limit for the dynamic array
   applications = (struct GrantApplication *)malloc(limit * sizeof(struct
GrantApplication));
```

```
int choice;
while (1) {
  printf("\n--- Grant Application Tracker ---\n");
  printf("1. Add a New Grant Application\n");
  printf("2. Update Application Status\n");
  printf("3. Display Applications Requesting Amount Greater Than a Value\n");
  printf("4. Display Approved Applications\n");
  printf("5. Exit\n");
  printf("Enter your choice: ");
  scanf("%d", &choice);

  switch (choice) {
    case 1:
      addGrantApplication(&applications, &count, &limit);
      break;
    case 2:
      updateApplicationStatus(applications, count);
      break;
    case 3:
      {
        float amount;
        printf("Enter the minimum requested amount: ");
        scanf("%f", &amount);
        displayApplicationsByAmount(applications, count, amount);
        break;
      }
    case 4:
      displayApprovedApplications(applications, count);
```

```
            break;
          case 5:
            free(applications);
            printf("Exiting...\n");
            return 0;
          default:
            printf("Invalid choice.\n");
        }
    }

    return 0;
}

void addGrantApplication(struct GrantApplication **applications, int *count, int *limit) {
    if (*count == *limit) {
        resizeArray(applications, limit);
    }

    printf("Enter Application ID: ");
    scanf("%s", (*applications)[*count].applicationID);
    printf("Enter Applicant Name: ");
    scanf("%s", (*applications)[*count].applicantName);
    printf("Enter Project Title: ");
    scanf(" %[^\n]%*c", (*applications)[*count].projectTitle);  // To allow spaces in project
title
    printf("Enter Requested Amount: ");
    scanf("%f", &(*applications)[*count].requestedAmount);
    printf("Enter Status (Submitted/Approved/Rejected): ");
    scanf("%s", (*applications)[*count].status);


    (*count)++;
    printf("Grant Application added successfully.\n");
}


void updateApplicationStatus(struct GrantApplication *applications, int count) {
    char applicationID[10];
    printf("Enter Application ID to update status: ");
    scanf("%s", applicationID);


    for (int i = 0; i < count; i++) {
        if (strcmp(applications[i].applicationID, applicationID) == 0) {
            printf("Enter new status (Submitted/Approved/Rejected): ");
            scanf("%s", applications[i].status);
            printf("Status updated successfully.\n");
            return;
        }
    }


    printf("Application with ID %s not found.\n", applicationID);
}


void displayApplicationsByAmount(struct GrantApplication *applications, int count,
float amount) {
    int found = 0;
    printf("Applications requesting an amount greater than %.2f:\n", amount);
```

```
  for (int i = 0; i < count; i++) {
    if (applications[i].requestedAmount > amount) {
      printf("Application ID: %s\n", applications[i].applicationID);
      printf("Applicant Name: %s\n", applications[i].applicantName);
      printf("Project Title: %s\n", applications[i].projectTitle);
      printf("Requested Amount: %.2f\n", applications[i].requestedAmount);
      printf("Status: %s\n\n", applications[i].status);
      found = 1;
    }
  }

  if (!found) {
    printf("No applications found with a requested amount greater than %.2f.\n", 
amount);
  }
}

void displayApprovedApplications(struct GrantApplication *applications, int count) {
  int found = 0;
  printf("Approved Applications:\n");

  for (int i = 0; i < count; i++) {
    if (strcmp(applications[i].status, "Approved") == 0) {
      printf("Application ID: %s\n", applications[i].applicationID);
      printf("Applicant Name: %s\n", applications[i].applicantName);
      printf("Project Title: %s\n", applications[i].projectTitle);
      printf("Requested Amount: %.2f\n", applications[i].requestedAmount);
      printf("Status: %s\n\n", applications[i].status);
```

```
            found = 1;
        }
    }

    if (!found) {
        printf("No approved applications found.\n");
    }
}

void resizeArray(struct GrantApplication **applications, int *limit) {
    *limit *= 2;
    *applications = (struct GrantApplication *)realloc(*applications, *limit * sizeof(struct
GrantApplication));
    printf("Array resized to accommodate more applications.\n");
}
```

Problem 4: Research Collaborator Management

Requirements:

- Create a structure Collaborator with members:

o char collaboratorID[10]

o char name[50]

o char institution[50]

o char expertiseArea[30]

o int numberOfProjects

- Functions to:
- Add a new collaborator to the database.

- Update the number of projects a collaborator is involved in.
- Display all collaborators from a specific institution.
- Find collaborators with expertise in a given area.
- Use dynamic memory allocation to manage the list of collaborators, allowing for expansion as more are added.

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

struct Collaborator {
  char collaboratorID[10];
  char name[50];
  char institution[50];
  char expertiseArea[30];
  int numberOfProjects;
};

// Function declarations
void addCollaborator(struct Collaborator **collaborators, int *count, int *limit);
void updateNumberOfProjects(struct Collaborator *collaborators, int count);
void displayCollaboratorsByInstitution(struct Collaborator *collaborators, int count, const char *institution);
void findCollaboratorsByExpertise(struct Collaborator *collaborators, int count, const char *expertiseArea);
void resizeArray(struct Collaborator **collaborators, int *limit);

int main() {
  struct Collaborator *collaborators = NULL;
```

```
int count = 0, limit = 2;  // Initial size limit for the dynamic array
collaborators = (struct Collaborator *)malloc(limit * sizeof(struct Collaborator));

int choice;
while (1) {
  printf("\n--- Research Collaborator Management ---\n");
  printf("1. Add a New Collaborator\n");
  printf("2. Update Number of Projects for a Collaborator\n");
  printf("3. Display Collaborators from a Specific Institution\n");
  printf("4. Find Collaborators with Expertise in a Given Area\n");
  printf("5. Exit\n");
  printf("Enter your choice: ");
  scanf("%d", &choice);

  switch (choice) {
    case 1:
      addCollaborator(&collaborators, &count, &limit);
      break;
    case 2:
      updateNumberOfProjects(collaborators, count);
      break;
    case 3:
      {
        char institution[50];
        printf("Enter the institution name: ");
        scanf(" %[^\n]%*c", institution);  // To allow spaces in the institution name
        displayCollaboratorsByInstitution(collaborators, count, institution);
        break;
```

```
            }
          case 4:
            {
              char expertiseArea[30];
              printf("Enter the expertise area: ");
              scanf(" %[^\n]%*c", expertiseArea);
              findCollaboratorsByExpertise(collaborators, count, expertiseArea);
              break;
            }
          case 5:
            free(collaborators);
            printf("Exiting...\n");
            return 0;
          default:
            printf("Invalid choice.\n");
      }
    }

    return 0;
}

void addCollaborator(struct Collaborator **collaborators, int *count, int *limit) {
    if (*count == *limit) {
      resizeArray(collaborators, limit);
    }

    printf("Enter Collaborator ID: ");
    scanf("%s", (*collaborators)[*count].collaboratorID);
```

```
  printf("Enter Name: ");
  scanf(" %[^\n]%*c", (*collaborators)[*count].name);  // To allow spaces in name
  printf("Enter Institution: ");
  scanf(" %[^\n]%*c", (*collaborators)[*count].institution);  // To allow spaces in
institution
  printf("Enter Expertise Area: ");
  scanf(" %[^\n]%*c", (*collaborators)[*count].expertiseArea);  // To allow spaces in
expertise area
  printf("Enter Number of Projects Involved: ");
  scanf("%d", &(*collaborators)[*count].numberOfProjects);

  (*count)++;
  printf("Collaborator added successfully.\n");
}

void updateNumberOfProjects(struct Collaborator *collaborators, int count) {
  char collaboratorID[10];
  printf("Enter Collaborator ID to update number of projects: ");
  scanf("%s", collaboratorID);

  for (int i = 0; i < count; i++) {
    if (strcmp(collaborators[i].collaboratorID, collaboratorID) == 0) {
      printf("Enter new number of projects: ");
      scanf("%d", &collaborators[i].numberOfProjects);
      printf("Number of projects updated successfully.\n");
      return;
    }
  }
```

```
    printf("Collaborator with ID %s not found.\n", collaboratorID);
}

void displayCollaboratorsByInstitution(struct Collaborator *collaborators, int count,
const char *institution) {
    int found = 0;
    printf("Collaborators from institution %s:\n", institution);

    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].institution, institution) == 0) {
            printf("Collaborator ID: %s\n", collaborators[i].collaboratorID);
            printf("Name: %s\n", collaborators[i].name);
            printf("Expertise Area: %s\n", collaborators[i].expertiseArea);
            printf("Number of Projects: %d\n\n", collaborators[i].numberOfProjects);
            found = 1;
        }
    }

    if (!found) {
        printf("No collaborators found from institution %s.\n", institution);
    }
}

void findCollaboratorsByExpertise(struct Collaborator *collaborators, int count, const
char *expertiseArea) {
    int found = 0;
    printf("Collaborators with expertise in %s:\n", expertiseArea);

    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].expertiseArea, expertiseArea) == 0) {
            printf("Collaborator ID: %s\n", collaborators[i].collaboratorID);
            printf("Name: %s\n", collaborators[i].name);
            printf("Institution: %s\n", collaborators[i].institution);
            printf("Number of Projects: %d\n\n", collaborators[i].numberOfProjects);
            found = 1;
        }
    }

    if (!found) {
        printf("No collaborators found with expertise in %s.\n", expertiseArea);
    }
}

void resizeArray(struct Collaborator **collaborators, int *limit) {
    *limit *= 2;
    *collaborators = (struct Collaborator *)realloc(*collaborators, *limit * sizeof(struct
Collaborator));
    printf("Array resized to accommodate more collaborators.\n");
}
```

Problem 5: Scientific Conference Submission Tracker

Requirements:

- Define a structure ConferenceSubmission with the following:
  - char submissionID[10]
  - char authorName[50]
  - char paperTitle[100]
  - char conferenceName[50]
  - char submissionDate[11]
  - char status[20] (e.g., Pending, Accepted, Rejected)
- Functions to:
- Add a new conference submission.
- Update the status of a submission.
- Display all submissions to a specific conference.
- Find and display submissions by a specific author.
- Store the conference submissions in a dynamically allocated array, resizing the array as needed when more submissions are added.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct ConferenceSubmission {

  char submissionID[10];

  char authorName[50];

  char paperTitle[100];

  char conferenceName[50];

  char submissionDate[11];

  char status[20];

};
```

```
// Function declarations

void addSubmission(struct ConferenceSubmission **submissions, int *count, int
*limit);

void updateSubmissionStatus(struct ConferenceSubmission *submissions, int count);

void displaySubmissionsByConference(struct ConferenceSubmission *submissions,
int count, const char *conferenceName);

void findSubmissionsByAuthor(struct ConferenceSubmission *submissions, int count,
const char *authorName);

void resizeArray(struct ConferenceSubmission **submissions, int *limit);


int main() {

  struct ConferenceSubmission *submissions = NULL;

  int count = 0, limit = 2;  // Initial size limit for the dynamic array

  submissions = (struct ConferenceSubmission *)malloc(limit * sizeof(struct
ConferenceSubmission));


  int choice;

  while (1) {

    printf("\n--- Scientific Conference Submission Tracker ---\n");

    printf("1. Add a New Submission\n");

    printf("2. Update Submission Status\n");

    printf("3. Display All Submissions to a Specific Conference\n");

    printf("4. Find and Display Submissions by a Specific Author\n");

    printf("5. Exit\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);


    switch (choice) {
        case 1:
          addSubmission(&submissions, &count, &limit);
          break;
        case 2:
          updateSubmissionStatus(submissions, count);
          break;
        case 3:
          {
            char conferenceName[50];
            printf("Enter the conference name: ");
            scanf(" %[^\n]%*c", conferenceName);  // To allow spaces in the conference
  name
            displaySubmissionsByConference(submissions, count, conferenceName);
            break;
          }
        case 4:
          {
            char authorName[50];
            printf("Enter the author's name: ");
            scanf(" %[^\n]%*c", authorName);
            findSubmissionsByAuthor(submissions, count, authorName);
            break;
          }
        case 5:
          free(submissions);
          printf("Exiting...\n");
          return 0;
        default:
            printf("Invalid choice.\n");
        }
    }

    return 0;
}

void addSubmission(struct ConferenceSubmission **submissions, int *count, int
*limit) {
    if (*count == *limit) {
        resizeArray(submissions, limit);
    }

    printf("Enter Submission ID: ");
    scanf("%s", (*submissions)[*count].submissionID);
    printf("Enter Author Name: ");
    scanf(" %[^\n]%*c", (*submissions)[*count].authorName);  // To allow spaces in
author name
    printf("Enter Paper Title: ");
    scanf(" %[^\n]%*c", (*submissions)[*count].paperTitle);  // To allow spaces in paper
title
    printf("Enter Conference Name: ");
    scanf(" %[^\n]%*c", (*submissions)[*count].conferenceName);  // To allow spaces in
conference name
    printf("Enter Submission Date (YYYY-MM-DD): ");
    scanf("%s", (*submissions)[*count].submissionDate);
    printf("Enter Status (Pending, Accepted, Rejected): ");
    scanf("%s", (*submissions)[*count].status);
    (*count)++;
    printf("Submission added successfully.\n");
}

void updateSubmissionStatus(struct ConferenceSubmission *submissions, int count) {
    char submissionID[10];
    printf("Enter Submission ID to update status: ");
    scanf("%s", submissionID);

    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].submissionID, submissionID) == 0) {
            printf("Enter new status (Pending, Accepted, Rejected): ");
            scanf("%s", submissions[i].status);
            printf("Status updated successfully.\n");
            return;
        }
    }

    printf("Submission with ID %s not found.\n", submissionID);
}

void displaySubmissionsByConference(struct ConferenceSubmission *submissions,
int count, const char *conferenceName) {
    int found = 0;
    printf("Submissions to conference %s:\n", conferenceName);

    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].conferenceName, conferenceName) == 0) {
            printf("Submission ID: %s\n", submissions[i].submissionID);
            printf("Author Name: %s\n", submissions[i].authorName);
            printf("Paper Title: %s\n", submissions[i].paperTitle);
            printf("Submission Date: %s\n", submissions[i].submissionDate);
            printf("Status: %s\n\n", submissions[i].status);
            found = 1;
        }
    }

    if (!found) {
        printf("No submissions found for conference %s.\n", conferenceName);
    }
}

void findSubmissionsByAuthor(struct ConferenceSubmission *submissions, int count,
const char *authorName) {
    int found = 0;
    printf("Submissions by author %s:\n", authorName);

    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].authorName, authorName) == 0) {
            printf("Submission ID: %s\n", submissions[i].submissionID);
            printf("Paper Title: %s\n", submissions[i].paperTitle);
            printf("Conference Name: %s\n", submissions[i].conferenceName);
            printf("Submission Date: %s\n", submissions[i].submissionDate);
            printf("Status: %s\n\n", submissions[i].status);
            found = 1;
        }
    }

    if (!found) {
        printf("No submissions found by author %s.\n", authorName);
    }
}

void resizeArray(struct ConferenceSubmission **submissions, int *limit) {
    *limit *= 2;
    *submissions = (struct ConferenceSubmission *)realloc(*submissions, *limit *
sizeof(struct ConferenceSubmission));
    printf("Array resized to accommodate more submissions.\n");
}
```