```
#include <stdio.h>

struct MyStruct {
    int num1;
    int num2;
    int num3;
};

int main() {
    struct MyStruct myStruct;

    // Assign values to the struct members
    myStruct.num1 = 10;
    myStruct.num2 = 20;
    myStruct.num3 = 30;

    // Print the values and memory addresses of the members
    printf("Value of num1: %d, Address: %p\n", myStruct.num1, &myStruct.num1);
    printf("Value of num2: %d, Address: %p\n", myStruct.num2, &myStruct.num2);
    printf("Value of num3: %d, Address: %p\n", myStruct.num3, &myStruct.num3);

    return 0;
}

=================================================================================
========

#include <stdio.h>
#include <string.h>
struct student {
    int rollno;;
    char Name[20];
    char section;
    int marks_maths;

};
struct student studInfo(void);
int main() {
    struct student stud;
    printf("enter student details: \n");
    stud = studInfo();
    printf("Roll no=%d \n Name=%s \n section=%c \n marks_maths=%d
\n",stud.rollno,stud.Name,stud.section,stud.marks_maths);
    return 0;

}


struct student studInfo(void){
        struct student stud1;
        stud1.rollno=100;
        strcpy(stud1.Name,"Noel");
        stud1.section=’c’;
        stud1.marks_maths=50;
        return stud1;


}


#include <stdio.h>

int main()
{
   struct node{
       int a;
       int b;
       int c;
   };
   struct node sum;
   printf("%p\n",&sum.a);
   printf("%p\n",&sum.b);
   printf("%p\n",&sum.c);
}
======================================================================

#include <stdio.h>

struct num{
    int a,b;
};
int sum(struct num,struct num);
int main(){

  struct num num1,num2;
  num1.a=30;
  num2.a=40;

  int sumA= sum(num1,num2);
   printf("sumA = %d",sumA);
    return 0;
}
int sum(struct num num1,struct num num2){
    int sum = num1.a+num2.a;
    return sum;
}
===========================================================================

#include <stdio.h>
#include <string.h>
struct student {
    int rollno;;
    char Name[20];
    char section;
    int marks_maths;

};
struct student studInfo(void);
int main() {
    struct student stud;
    printf("enter student details: \n");
    stud = studInfo();
    printf("Roll no=%d \n Name=%s \n section=%c \n marks_maths=%d
\n",stud.rollno,stud.Name,stud.section,stud.marks_maths);
    return 0;

}


struct student studInfo(void){
        struct student stud1;
        stud1.rollno=100;
        strcpy(stud1.Name,"Noel");
        stud1.section='c';
        stud1.marks_maths=50;

        return stud1;


}
=====================================================================

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

struct student {
    int rollNo;
    char Name[20];
    char section;
    int marks_maths;
};

int main() {


    struct student* students = (struct student*) malloc(5 * sizeof(struct student));

    if (students == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        printf("Enter details for student %d:\n", i + 1);

        printf("Roll Number: ");
        scanf("%d", &students[i].rollNo);

        printf("Name: ");
        scanf("%s", students[i].Name);

        printf("Section: ");
        scanf(" %c", &students[i].section);
        printf("Marks in Maths: ");
        scanf("%d", &students[i].marks_maths);
        printf("\n");
    }

    printf("\nStudent details:\n");
    for (int i = 0; i < 5; i++) {
        printf("Student %d - Roll No: %d, Name: %s, Section: %c, Marks in Maths: %d\n",
               i + 1, students[i].rollNo, students[i].Name, students[i].section, students[i].marks_maths);
    }

    free(students);

    return 0;
}
```

==
=========

SET OF PROBLEMS
==
====

1.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.

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

struct Vehicle {
    char registrationNumber[15];
    char model[30];
    int yearOfManufacture;
    float mileage;
    float fuelEfficiency;
};

void addVehicle(struct Vehicle **fleet, int *fleetSize) {
    (*fleetSize)++;
    *fleet = realloc(*fleet, (*fleetSize) * sizeof(struct Vehicle));

    if (*fleet == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    printf("Enter vehicle details:\n");
    printf("Registration Number: ");
    scanf("%s", (*fleet)[*fleetSize - 1].registrationNumber);
    printf("Model: ");
    scanf("%s", (*fleet)[*fleetSize - 1].model);
    printf("Year of Manufacture: ");
    scanf("%d", &(*fleet)[*fleetSize - 1].yearOfManufacture);
    printf("Mileage: ");
    scanf("%f", &(*fleet)[*fleetSize - 1].mileage);
    printf("Fuel Efficiency: ");
    scanf("%f", &(*fleet)[*fleetSize - 1].fuelEfficiency);
}

void updateVehicle(struct Vehicle *fleet, int fleetSize) {
    char registrationNumber[15];
    printf("Enter the registration number of the vehicle to update: ");
    scanf("%s", registrationNumber);

    for (int i = 0; i < fleetSize; i++) {
        if (strcmp(fleet[i].registrationNumber, registrationNumber) == 0) {
            printf("Enter updated mileage: ");
            scanf("%f", &fleet[i].mileage);
            printf("Enter updated fuel efficiency: ");
            scanf("%f", &fleet[i].fuelEfficiency);
            printf("Vehicle updated successfully!\n");
            return;
        }
    }
    printf("Vehicle not found!\n");
}

void displayVehiclesAfterYear(struct Vehicle *fleet, int fleetSize, int year) {
    printf("Vehicles manufactured after %d:\n", year);
    for (int i = 0; i < fleetSize; i++) {
        if (fleet[i].yearOfManufacture > year) {
            printf("Registration: %s, Model: %s, Year: %d, Mileage: %.2f, Fuel Efficiency: %.2f\n",
                    fleet[i].registrationNumber, fleet[i].model, fleet[i].yearOfManufacture,
                    fleet[i].mileage, fleet[i].fuelEfficiency);
        }
    }
}

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

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

    printf("Vehicle with highest fuel efficiency:\n");
    printf("Registration: %s, Model: %s, Year: %d, Mileage: %.2f, Fuel Efficiency: %.2f\n",
            fleet[maxIndex].registrationNumber, fleet[maxIndex].model, fleet[maxIndex].yearOfManufacture,
            fleet[maxIndex].mileage, fleet[maxIndex].fuelEfficiency);
}

int main() {
    struct Vehicle *fleet = NULL;
    int fleetSize = 0;
    int choice;

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

        switch (choice) {
            case 1:
                addVehicle(&fleet, &fleetSize);
                break;
            case 2:
                updateVehicle(fleet, fleetSize);
                break;
            case 3:
                {
                    int year;
                    printf("Enter the year: ");
                    scanf("%d", &year);
                    displayVehiclesAfterYear(fleet, fleetSize, year);
                }
                break;
            case 4:
                findHighestFuelEfficiency(fleet, fleetSize);
                break;
            case 5:
                free(fleet);
                printf("Exiting program...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

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.

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

// Define a structure for Car Rental
struct CarRental {
    char carID[10];
    char customerName[50];
    char rentalDate[11]; // Format: YYYY-MM-DD
    char returnDate[11]; // Format: YYYY-MM-DD
    float rentalPricePerDay;
};

// Function prototypes
int calculateRentalDays(const char *rentalDate, const char *returnDate);
void bookCar(struct CarRental rentals[], int *rentalCount);
float calculateTotalPrice(int rentalDays, float rentalPricePerDay);
void displayAllRentals(struct CarRental rentals[], int rentalCount);
void searchByCustomerName(struct CarRental rentals[], int rentalCount, const char *customerName);
int isValidDate(const char *date);
int isLeapYear(int year);
int daysInMonth(int month, int year);
int convertToDate(const char *date);

int main() {
    struct CarRental rentals[100];
    int rentalCount = 0;
    int choice;

    // Main Menu
    while (1) {
        printf("\nCar Rental Reservation System\n");
        printf("1. Book a car\n");
        printf("2. Display all current rentals\n");
        printf("3. Search rentals by customer name\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar(); // Consume newline character left by scanf

        switch (choice) {
            case 1:
                bookCar(rentals, &rentalCount);
                break;
            case 2:
                displayAllRentals(rentals, rentalCount);
                break;
            case 3:
                {
                    char customerName[50];
                    printf("Enter customer name to search: ");
                    scanf("%s", customerName);
                    searchByCustomerName(rentals, rentalCount, customerName);
                }
                break;
            case 4:
                printf("Exiting the system.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}

// Function to check if a date is valid
int isValidDate(const char *date) {
    int year, month, day;
    int valid = sscanf(date, "%4d-%2d-%2d", &year, &month, &day);

    if (valid != 3) return 0; // Invalid format

    if (month < 1 || month > 12 || day < 1 || day > daysInMonth(month, year)) {
        return 0; // Invalid month or day
    }
    return 1; // Valid date
}

// Function to check if a year is a leap year
int isLeapYear(int year) {
    return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
}

// Function to get the number of days in a month
int daysInMonth(int month, int year) {
    switch (month) {
        case 1: case 3: case 5: case 7: case 8: case 10: case 12:
            return 31;
        case 4: case 6: case 9: case 11:
            return 30;
        case 2:
            return isLeapYear(year) ? 29 : 28;
        default:
            return 0;
    }
}
```

```
// Function to convert a date string (YYYY-MM-DD) into a date value (integer number of days since a fixed
reference date)
int convertToDate(const char *date) {
    int year, month, day;
    sscanf(date, "%4d-%2d-%2d", &year, &month, &day);

    int days = 0;
    // Add days for each previous year
    for (int i = 1970; i < year; i++) {
        days += isLeapYear(i) ? 366 : 365;
    }

    // Add days for each previous month in the current year
    for (int i = 1; i < month; i++) {
        days += daysInMonth(i, year);
    }

    // Add the days of the current month
    days += day;

    return days;
}

// Function to calculate the number of rental days
int calculateRentalDays(const char *rentalDate, const char *returnDate) {
    int rentalDays = convertToDate(rentalDate);
    int returnDays = convertToDate(returnDate);

    return returnDays - rentalDays; // Return difference in days
}

// Function to calculate the total rental price
float calculateTotalPrice(int rentalDays, float rentalPricePerDay) {
    return rentalDays * rentalPricePerDay;
}

// Function to book a car
void bookCar(struct CarRental rentals[], int *rentalCount) {
    struct CarRental rental;
    printf("Enter car ID: ");
    scanf("%s", rental.carID);

    printf("Enter customer name: ");
    scanf("%s", rental.customerName);

    printf("Enter rental date (YYYY-MM-DD): ");
    scanf("%s", rental.rentalDate);

    // Validate rental date
    if (!isValidDate(rental.rentalDate)) {
        printf("Invalid rental date format. Please use YYYY-MM-DD format.\n");
        return;
    }
    printf("Enter return date (YYYY-MM-DD): ");
    scanf("%s", rental.returnDate);

    // Validate return date
    if (!isValidDate(rental.returnDate)) {
        printf("Invalid return date format. Please use YYYY-MM-DD format.\n");
        return;
    }

    // Calculate rental days
    int rentalDays = calculateRentalDays(rental.rentalDate, rental.returnDate);
    if (rentalDays <= 0) {
        printf("Invalid rental period. Return date must be after rental date.\n");
        return;
    }

    printf("Enter rental price per day: ");
    scanf("%f", &rental.rentalPricePerDay);

    // Calculate total price
    float totalPrice = calculateTotalPrice(rentalDays, rental.rentalPricePerDay);
    printf("Total rental price: %.2f\n", totalPrice);

    rentals[*rentalCount] = rental;
    (*rentalCount)++;
    printf("Car booked successfully!\n");
}

// Function to display all current rentals
void displayAllRentals(struct CarRental rentals[], int rentalCount) {
    if (rentalCount == 0) {
        printf("No rentals available.\n");
        return;
    }

    for (int i = 0; i < rentalCount; i++) {
        printf("\nCar ID: %s\n", rentals[i].carID);
        printf("Customer Name: %s\n", rentals[i].customerName);
        printf("Rental Date: %s\n", rentals[i].rentalDate);
        printf("Return Date: %s\n", rentals[i].returnDate);
        printf("Rental Price Per Day: %.2f\n", rentals[i].rentalPricePerDay);
        int rentalDays = calculateRentalDays(rentals[i].rentalDate, rentals[i].returnDate);
        float totalPrice = calculateTotalPrice(rentalDays, rentals[i].rentalPricePerDay);
        printf("Total Rental Price: %.2f\n", totalPrice);
    }
}

// Function to search rentals by customer name
void searchByCustomerName(struct CarRental rentals[], int rentalCount, const char *customerName) {
    int found = 0;
    for (int i = 0; i < rentalCount; i++) {
        if (strstr(rentals[i].customerName, customerName) != NULL) {
            found = 1;
            printf("\nCar ID: %s\n", rentals[i].carID);
            printf("Customer Name: %s\n", rentals[i].customerName);
            printf("Rental Date: %s\n", rentals[i].rentalDate);
            printf("Return Date: %s\n", rentals[i].returnDate);
            printf("Rental Price Per Day: %.2f\n", rentals[i].rentalPricePerDay);
            int rentalDays = calculateRentalDays(rentals[i].rentalDate, rentals[i].returnDate);
            float totalPrice = calculateTotalPrice(rentalDays, rentals[i].rentalPricePerDay);
            printf("Total Rental Price: %.2f\n", totalPrice);
        }
    }
    if (!found) {
        printf("No rentals found for customer: %s\n", customerName);
    }
}
```

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];
    float speed;
    float latitude;
    float longitude;
};

// Function to log new sensor data
void logSensorData(struct SensorData* data, int* count) {
    printf("Enter Sensor Data:\n");

    printf("Sensor ID: ");
    scanf("%d", &data[*count].sensorID);
    printf("Timestamp (YYYY-MM-DD HH:MM:SS): ");
    scanf("%s", data[*count].timestamp);
    printf("Speed: ");
    scanf("%f", &data[*count].speed);
    printf("Latitude: ");
    scanf("%f", &data[*count].latitude);
    printf("Longitude: ");
    scanf("%f", &data[*count].longitude);

    (*count)++;
}

// Function to display sensor data for a specific time range
void displayDataInRange(struct SensorData* data, int count, const char* start, const char* end) {
    printf("Displaying data between %s and %s:\n", start, end);
    for (int i = 0; i < count; i++) {
        if (strcmp(data[i].timestamp, start) >= 0 && strcmp(data[i].timestamp, end) <= 0) {
            printf("Sensor ID: %d, Timestamp: %s, Speed: %.2f, Latitude: %.2f, Longitude: %.2f\n",
                   data[i].sensorID, data[i].timestamp, data[i].speed, data[i].latitude, data[i].longitude);
        }
    }
}

// Function to find the maximum speed recorded
float findMaxSpeed(struct SensorData* data, int count) {
    float maxSpeed = data[0].speed;
    for (int i = 1; i < count; i++) {
        if (data[i].speed > maxSpeed) {
            maxSpeed = data[i].speed;
        }
    }
    return maxSpeed;
}

// Function to calculate the average speed over a specific time range
float calculateAverageSpeed(struct SensorData* data, int count, const char* start, const char* end) {
    float totalSpeed = 0.0;
    int validCount = 0;

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

    return (validCount > 0) ? totalSpeed / validCount : 0.0;
}

int main() {
    int count = 0;
    int capacity = 2;

    struct SensorData* data = (struct SensorData*)malloc(capacity * sizeof(struct SensorData));

    if (data == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    // Log 3 sensor data entries
    logSensorData(data, &count);
    logSensorData(data, &count);
    logSensorData(data, &count);


    // Display data for a specific time range
    char startTime[20], endTime[20];
    printf("Enter start time (YYYY-MM-DD HH:MM:SS): ");
    scanf("%s", startTime);
    printf("Enter end time (YYYY-MM-DD HH:MM:SS): ");
    scanf("%s", endTime);

    displayDataInRange(data, count, startTime, endTime);

    // Find and display the maximum speed
    float maxSpeed = findMaxSpeed(data, count);
    printf("Maximum speed recorded: %.2f\n", maxSpeed);

    // Calculate and display the average speed in the given range
    float avgSpeed = calculateAverageSpeed(data, count, startTime, endTime);
    printf("Average speed between %s and %s: %.2f\n", startTime, endTime, avgSpeed);

    free(data);

    return 0;
}
```

4.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).

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

#define MAX_ENGINES 100

// Define the EnginePerformance structure
struct EnginePerformance {
    char engineID[10];
    float temperature;
    float rpm;
    float fuelConsumptionRate;
    float oilPressure;
};
```

```
// Function to add performance data for a specific engine
void addEngineData(struct EnginePerformance engines[], int *numEngines, const char *id, float temp,
float rpm, float fuelRate, float oilPressure) {
    struct EnginePerformance engine;
    strcpy(engine.engineID, id);
    engine.temperature = temp;
    engine.rpm = rpm;
    engine.fuelConsumptionRate = fuelRate;
    engine.oilPressure = oilPressure;

    // Add the new engine data to the array
    engines[*numEngines] = engine;
    (*numEngines)++;
}

// Function to display all performance data for a specific engine ID
void displayEngineData(struct EnginePerformance engines[], int numEngines, const char *engineID) {
    int found = 0;
    for (int i = 0; i < numEngines; i++) {
        if (strcmp(engines[i].engineID, engineID) == 0) {
            found = 1;
            printf("Engine ID: %s\n", engines[i].engineID);
            printf("Temperature: %.2f °C\n", engines[i].temperature);
            printf("RPM: %.2f\n", engines[i].rpm);
            printf("Fuel Consumption Rate: %.2f L/h\n", engines[i].fuelConsumptionRate);
            printf("Oil Pressure: %.2f psi\n", engines[i].oilPressure);
            printf("--------------------------------\n");
        }
    }
    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 engines[], int numEngines, const char *engineID) {
    float totalTemp = 0;
    float totalRPM = 0;
    int count = 0;

    for (int i = 0; i < numEngines; i++) {
        if (strcmp(engines[i].engineID, engineID) == 0) {
            totalTemp += engines[i].temperature;
            totalRPM += engines[i].rpm;
            count++;
        }
    }

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

```
// Function to identify engines with abnormal oil pressure (above or below specified thresholds)
void identifyAbnormalOilPressure(struct EnginePerformance engines[], int numEngines, float
lowThreshold, float highThreshold) {
    for (int i = 0; i < numEngines; i++) {
        if (engines[i].oilPressure < lowThreshold || engines[i].oilPressure > highThreshold) {
            printf("Engine ID: %s has abnormal oil pressure: %.2f psi\n", engines[i].engineID,
engines[i].oilPressure);
        }
    }
}

int main() {
    struct EnginePerformance engines[MAX_ENGINES];
    int numEngines = 0;

    // Add some performance data for engines
    addEngineData(engines, &numEngines, "ENG001", 90.5, 3000, 10.2, 45.0);
    addEngineData(engines, &numEngines, "ENG002", 95.0, 3500, 12.5, 35.0);
    addEngineData(engines, &numEngines, "ENG001", 92.0, 3200, 11.5, 50.0);
    addEngineData(engines, &numEngines, "ENG003", 100.0, 2800, 14.0, 60.0);

    // Display all performance data for a specific engine
    printf("Displaying data for engine ID ENG001:\n");
    displayEngineData(engines, numEngines, "ENG001");

    // Calculate the average temperature and RPM for a specific engine
    printf("Calculating average data for engine ID ENG001:\n");
    calculateAverage(engines, numEngines, "ENG001");

    // Identify any engine with abnormal oil pressure (e.g., thresholds 40-50 psi)
    printf("Identifying engines with abnormal oil pressure:\n");
    identifyAbnormalOilPressure(engines, numEngines, 40.0, 50.0);

    return 0;
}
```

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

```
// Define the structure ServiceRecord
struct ServiceRecord {
    char serviceID[10];
    char vehicleID[15];
    char serviceDate[11];  // Format: YYYY-MM-DD
    char description[100];
    float serviceCost;
};

// Function to add a new service record
void addServiceRecord(struct ServiceRecord records[], int *numRecords) {
    struct ServiceRecord newRecord;

    printf("Enter Service ID: ");
    scanf("%s", newRecord.serviceID);

    printf("Enter Vehicle ID: ");
    scanf("%s", newRecord.vehicleID);

    printf("Enter Service Date (YYYY-MM-DD): ");
    scanf("%s", newRecord.serviceDate);

    printf("Enter Service Description (no spaces): ");
    scanf("%s", newRecord.description);  // Read description without spaces

    printf("Enter Service Cost: ");
    scanf("%f", &newRecord.serviceCost);

    // Store the new record in the records array
    records[*numRecords] = newRecord;
    (*numRecords)++;
    printf("Service record added successfully!\n");
}

// Function to display all service records for a given vehicle ID
void displayServiceRecords(struct ServiceRecord records[], int numRecords, const char vehicleID[]) {
    printf("\nService records for vehicle ID %s:\n", vehicleID);
    int found = 0;
    for (int i = 0; i < numRecords; i++) {
        if (strcmp(records[i].vehicleID, vehicleID) == 0) {
            printf("Service ID: %s\n", records[i].serviceID);
            printf("Service Date: %s\n", records[i].serviceDate);
            printf("Description: %s\n", records[i].description);
            printf("Service Cost: %.2f\n", records[i].serviceCost);
            printf("------------------------------\n");
            found = 1;
        }
    }
    if (!found) {
        printf("No service records found for vehicle ID %s.\n", vehicleID);
    }
}

// Function to calculate the total cost of services for a vehicle
```

```c
float calculateTotalCost(struct ServiceRecord records[], int numRecords, const char vehicleID[]) {
    float totalCost = 0;
    for (int i = 0; i < numRecords; i++) {
        if (strcmp(records[i].vehicleID, vehicleID) == 0) {
            totalCost += records[i].serviceCost;
        }
    }
    return totalCost;
}

// Function to compare service records for sorting by date
int compareServiceDates(const void *a, const void *b) {
    return strcmp(((struct ServiceRecord *)a)->serviceDate, ((struct ServiceRecord *)b)->serviceDate);
}

// Function to sort and display service records by service date
void sortAndDisplayRecordsByDate(struct ServiceRecord records[], int numRecords) {
    qsort(records, numRecords, sizeof(struct ServiceRecord), compareServiceDates);

    printf("\nSorted Service Records by Date:\n");
    for (int i = 0; i < numRecords; i++) {
        printf("Service ID: %s\n", records[i].serviceID);
        printf("Vehicle ID: %s\n", records[i].vehicleID);
        printf("Service Date: %s\n", records[i].serviceDate);
        printf("Description: %s\n", records[i].description);
        printf("Service Cost: %.2f\n", records[i].serviceCost);
        printf("------------------------------\n");
    }
}

int main() {
    struct ServiceRecord records[100];  // Array to store service records
    int numRecords = 0;  // Counter for the number of records

    int choice;
    char vehicleID[15];

    do {
        printf("\nVehicle Service History Tracker Menu:\n");
        printf("1. Add a new service record\n");
        printf("2. Display all service records for a given vehicle ID\n");
        printf("3. Calculate the total cost of services 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(records, &numRecords);
                break;
            case 2:
                printf("Enter Vehicle ID to display records: ");
                scanf("%s", vehicleID);
                displayServiceRecords(records, numRecords, vehicleID);
                break;
            case 3:
                printf("Enter Vehicle ID to calculate total service cost: ");
                scanf("%s", vehicleID);
                printf("Total cost of services for vehicle ID %s: %.2f\n", vehicleID, calculateTotalCost(records,
numRecords, vehicleID));
                break;
            case 4:
                sortAndDisplayRecordsByDate(records, numRecords);
                break;
            case 5:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

SECOND SET OF PROBLEMS
====================================

Problem 1: Player Statistics Management
Requirements:
Define a structure Player with the following members:
char name[50]
int age
char team[30]
int matchesPlayed
int totalRuns
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>

// Define the Player structure
struct Player {
  char name[50];
  int age;
  char team[30];
  int matchesPlayed;
  int totalRuns;
  int totalWickets;
};
```

```
// Function to add a new player
void addPlayer(struct Player **players, int *playerCount) {
    (*playerCount)++;
    *players = realloc(*players, (*playerCount) * sizeof(struct Player));

    if (*players == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    struct Player newPlayer;
    printf("Enter player's name: ");
    scanf(" %[^\n]", newPlayer.name);
    printf("Enter player's age: ");
    scanf("%d", &newPlayer.age);
    printf("Enter player's team: ");
    scanf(" %[^\n]", newPlayer.team);
    newPlayer.matchesPlayed = 0;
    newPlayer.totalRuns = 0;
    newPlayer.totalWickets = 0;

    (*players)[*playerCount - 1] = newPlayer;
    printf("Player added successfully!\n");
}

// Function to update a player's statistics after a match
void updatePlayerStats(struct Player *players, int playerCount) {
    char playerName[50];
    printf("Enter the player's name to update stats: ");
    scanf(" %[^\n]", playerName);

    for (int i = 0; i < playerCount; i++) {
        if (strcmp(players[i].name, playerName) == 0) {
            int runs, wickets;
            printf("Enter runs scored by %s: ", playerName);
            scanf("%d", &runs);
            printf("Enter wickets taken by %s: ", playerName);
            scanf("%d", &wickets);

            players[i].matchesPlayed++;
            players[i].totalRuns += runs;
            players[i].totalWickets += wickets;

            printf("Statistics updated successfully!\n");
            return;
        }
    }
    printf("Player not found!\n");
}

// Function to display players of a specific team
void displayPlayersByTeam(struct Player *players, int playerCount, char *team) {
    int found = 0;
    printf("Players from team %s:\n", team);
    for (int i = 0; i < playerCount; i++) {
```

```c
        if (strcmp(players[i].team, team) == 0) {
            printf("Name: %s, Age: %d, Matches Played: %d, Runs: %d, Wickets: %d\n",
                   players[i].name, players[i].age, players[i].matchesPlayed,
                   players[i].totalRuns, players[i].totalWickets);
            found = 1;
        }
    }
    if (!found) {
        printf("No players found for team %s.\n", team);
    }
}

// Function to find the player with the highest runs and the player with the most wickets
void findTopPlayers(struct Player *players, int playerCount) {
    if (playerCount == 0) {
        printf("No players in the system!\n");
        return;
    }

    struct Player *highestRunsPlayer = &players[0];
    struct Player *mostWicketsPlayer = &players[0];

    for (int i = 1; i < playerCount; i++) {
        if (players[i].totalRuns > highestRunsPlayer->totalRuns) {
            highestRunsPlayer = &players[i];
        }
        if (players[i].totalWickets > mostWicketsPlayer->totalWickets) {
            mostWicketsPlayer = &players[i];
        }
    }

    printf("Player with highest runs: %s with %d runs\n", highestRunsPlayer->name,
highestRunsPlayer->totalRuns);
    printf("Player with most wickets: %s with %d wickets\n", mostWicketsPlayer->name,
mostWicketsPlayer->totalWickets);
}

int main() {
    struct Player *players = NULL;
    int playerCount = 0;
    int choice;

    do {
        printf("\nPlayer Statistics Management System\n");
        printf("1. Add a new player\n");
        printf("2. Update player's statistics\n");
        printf("3. Display players from a specific team\n");
        printf("4. Find player with highest runs and most wickets\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addPlayer(&players, &playerCount);
                break;
            case 2:
                updatePlayerStats(players, playerCount);
                break;
            case 3: {
                char team[30];
                printf("Enter the team name: ");
                scanf(" %[^\n]", team);
                displayPlayersByTeam(players, playerCount, team);
                break;
            }
            case 4:
                findTopPlayers(players, playerCount);
                break;
            case 5:
                free(players);
                printf("Exiting the program...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

Problem 2: Tournament Fixture Scheduler
Requirements:
Create a structure Match with members:
char team1[30]
char team2[30]
char date[11] (format: YYYY-MM-DD)
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>

#define INITIAL_CAPACITY 2

// Structure to store match details
struct Match {
    char team1[30];
    char team2[30];
    char date[11];  // Format: YYYY-MM-DD
    char venue[50];
};

// Function prototypes
```

```
void scheduleMatch(struct Match **matches, int *count, int *capacity);
void displayMatches(struct Match *matches, int count);
void searchMatchesByDate(struct Match *matches, int count, const char *date);
void cancelMatch(struct Match **matches, int *count, int *capacity, const char *team1, const char *team2,
const char *date);
void resizeArray(struct Match **matches, int *capacity);
int findMatchIndex(struct Match *matches, int count, const char *team1, const char *team2, const char
*date);

int main() {
    struct Match *matches = NULL;  // Pointer to store matches dynamically
    int matchCount = 0;            // Number of scheduled matches
    int matchCapacity = INITIAL_CAPACITY;  // Initial capacity of the array

    matches = (struct Match *)malloc(matchCapacity * sizeof(struct Match)); // Allocate initial memory

    if (matches == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    int choice;
    do {
        printf("\nTournament 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);
        getchar();  // Consume newline character after integer input

        switch (choice) {
            case 1:
                scheduleMatch(&matches, &matchCount, &matchCapacity);
                break;
            case 2:
                displayMatches(matches, matchCount);
                break;
            case 3: {
                char searchDate[11];
                printf("Enter date to search (YYYY-MM-DD): ");
                scanf("%s", searchDate);
                searchMatchesByDate(matches, matchCount, searchDate);
                break;
            }
            case 4: {
                char team1[30], team2[30], matchDate[11];
                printf("Enter team1: ");
                scanf("%s", team1);
                printf("Enter team2: ");
                scanf("%s", team2);
                printf("Enter match date (YYYY-MM-DD): ");
                scanf("%s", matchDate);
                cancelMatch(&matches, &matchCount, &matchCapacity, team1, team2, matchDate);
                break;
            }
            case 5:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    free(matches);  // Free allocated memory
    return 0;
}

// Function to schedule a new match
void scheduleMatch(struct Match **matches, int *count, int *capacity) {
    if (*count == *capacity) {
        resizeArray(matches, capacity);  // Resize array if capacity is reached
    }

    printf("Enter team1: ");
    scanf("%s", (*matches)[*count].team1);
    printf("Enter team2: ");
    scanf("%s", (*matches)[*count].team2);
    printf("Enter match date (YYYY-MM-DD): ");
    scanf("%s", (*matches)[*count].date);
    printf("Enter venue: ");
    scanf(" %[^\n]s", (*matches)[*count].venue);

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

// Function to display all scheduled matches
void displayMatches(struct Match *matches, 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", matches[i].team1);
        printf("  Team 2: %s\n", matches[i].team2);
        printf("  Date: %s\n", matches[i].date);
        printf("  Venue: %s\n", matches[i].venue);
        printf("\n");
    }
}

// Function to search matches by date
void searchMatchesByDate(struct Match *matches, int count, const char *date) {
    int found = 0;
    printf("\nMatches on %s:\n", date);
    for (int i = 0; i < count; i++) {
        if (strcmp(matches[i].date, date) == 0) {
            printf("  Team 1: %s vs Team 2: %s at %s\n", matches[i].team1, matches[i].team2,
matches[i].venue);
            found = 1;
        }
    }

    if (!found) {
        printf("No matches found on this date.\n");
    }
}

// Function to cancel a match
void cancelMatch(struct Match **matches, int *count, int *capacity, const char *team1, const char *team2,
const char *date) {
    int index = findMatchIndex(*matches, *count, team1, team2, date);

    if (index == -1) {
        printf("No match found with the specified details.\n");
        return;
    }

    // Shift all elements to remove the match
    for (int i = index; i < *count - 1; i++) {
        (*matches)[i] = (*matches)[i + 1];
    }

    (*count)--;  // Decrement match count
    printf("Match between %s and %s on %s has been canceled.\n", team1, team2, date);
}

// Function to resize the match array
void resizeArray(struct Match **matches, int *capacity) {
    *capacity *= 2;  // Double the capacity
    *matches = (struct Match *)realloc(*matches, *capacity * sizeof(struct Match));

    if (*matches == NULL) {
        printf("Memory allocation failed during resizing!\n");
        exit(1);
    }

    printf("Match schedule array resized to %d.\n", *capacity);
}

// Function to find the index of a match for cancellation
int findMatchIndex(struct Match *matches, int count, const char *team1, const char *team2, const char
*date) {
    for (int i = 0; i < count; i++) {
        if (strcmp(matches[i].team1, team1) == 0 && strcmp(matches[i].team2, team2) == 0 &&
strcmp(matches[i].date, date) == 0) {
            return i;
        }
    }
    return -1;  // Match not found
}
```

Problem 3: Sports Event Medal Tally
Requirements:
Define a structure CountryMedalTally with members:
char country[30]
int gold
int silver
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.

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

#define INITIAL_SIZE 5

// Define structure to hold country medal tally
struct CountryMedalTally {
    char country[30];
    int gold;
    int silver;
    int bronze;
};

// Global array to store the medal tally and its size
struct CountryMedalTally *medalTally = NULL;
int currentSize = 0;
int allocatedSize = 0;

// Function to resize the array dynamically
void resizeArray(int newSize) {
    allocatedSize = newSize;
    medalTally = realloc(medalTally, allocatedSize * sizeof(struct CountryMedalTally));
    if (medalTally == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}

// Function to add a new country's medal tally
void addCountryMedalTally(char country[], int gold, int silver, int bronze) {
    // If the array is full, resize it
    if (currentSize == allocatedSize) {
        resizeArray(allocatedSize == 0 ? INITIAL_SIZE : allocatedSize * 2);
    }

    // Add the country's medal tally
    strcpy(medalTally[currentSize].country, country);
    medalTally[currentSize].gold = gold;
    medalTally[currentSize].silver = silver;
    medalTally[currentSize].bronze = bronze;
    currentSize++;
}

// Function to update the medal count for a country
void updateMedalTally(char country[], int gold, int silver, int bronze) {
    for (int i = 0; i < currentSize; i++) {
        if (strcmp(medalTally[i].country, country) == 0) {
            medalTally[i].gold += gold;
            medalTally[i].silver += silver;
            medalTally[i].bronze += bronze;
            return;
        }
    }
    printf("Country not found.\n");
}

// Function to display the medal tally for all countries
void displayMedalTally() {
    printf("Medal Tally:\n");
    for (int i = 0; i < currentSize; i++) {
        printf("%s: Gold: %d, Silver: %d, Bronze: %d\n",
               medalTally[i].country,
               medalTally[i].gold,
               medalTally[i].silver,
               medalTally[i].bronze);
    }
}

// Function to find the country with the highest number of gold medals
void findCountryWithHighestGold() {
    if (currentSize == 0) {
        printf("No countries in the tally.\n");
        return;
    }

    int maxGoldIndex = 0;
    for (int i = 1; i < currentSize; i++) {
        if (medalTally[i].gold > medalTally[maxGoldIndex].gold) {
            maxGoldIndex = i;
        }
    }

    printf("Country with the highest gold medals: %s with %d gold medals.\n",
           medalTally[maxGoldIndex].country, medalTally[maxGoldIndex].gold);
}

// Main function to drive the program
int main() {
    // Allocate initial memory for the medal tally array
    resizeArray(INITIAL_SIZE);
    // Add some countries and their medal tallies
    addCountryMedalTally("USA", 50, 20, 10);
    addCountryMedalTally("China", 48, 22, 12);
    addCountryMedalTally("Germany", 20, 15, 18);

    // Display current medal tally
    displayMedalTally();

    // Update medal tally for a country
    updateMedalTally("USA", 5, 3, 2);
    updateMedalTally("Germany", 1, 4, 3);

    // Display updated medal tally
    printf("\nUpdated Medal Tally:\n");
    displayMedalTally();

    // Find and display the country with the highest number of gold medals
    findCountryWithHighestGold();

    // Free the allocated memory
    free(medalTally);

    return 0;
}
```

4.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>

// Structure definition
struct Athlete {
    char athleteID[10];
    char name[50];
    char sport[30];
    float personalBest;
    float lastPerformance;
};

// Function prototypes
void addAthlete(struct Athlete** athletes, int* numAthletes);
void updateLastPerformance(struct Athlete* athletes, int numAthletes);
void displayAllAthletes(struct Athlete* athletes, int numAthletes, const char* sport);
void displayNewPersonalBest(struct Athlete* athletes, int numAthletes);

int main() {
    struct Athlete* athletes = NULL;
    int numAthletes = 0;
    int choice;

    do {
        printf("\nPerformance Tracker Menu:\n");
        printf("1. Add a new athlete\n");
        printf("2. Update an athlete's last performance\n");
        printf("3. Display all athletes in a specific sport\n");
        printf("4. Identify and display athletes who set a new personal best\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                addAthlete(&athletes, &numAthletes);
                break;
            case 2:
                updateLastPerformance(athletes, numAthletes);
                break;
            case 3: {
                char sport[30];
                printf("Enter sport name: ");
                scanf("%s", sport);
                displayAllAthletes(athletes, numAthletes, sport);
                break;
            }
            case 4:
                displayNewPersonalBest(athletes, numAthletes);
                break;
            case 5:
                printf("Exiting the system.\n");
                break;
            default:
                printf("Invalid choice, please try again.\n");
        }
    } while(choice != 5);

    free(athletes);  // Free allocated memory before exiting
    return 0;
}

void addAthlete(struct Athlete** athletes, int* numAthletes) {
    // Allocate memory for one more athlete
    *athletes = realloc(*athletes, (*numAthletes + 1) * sizeof(struct Athlete));

    if (*athletes == NULL) {
        printf("Memory allocation failed. Exiting...\n");
        exit(1);  // Exit if memory allocation fails
    }

    // Input athlete details
    struct Athlete* newAthlete = &(*athletes)[*numAthletes];
    printf("Enter athlete ID: ");
    scanf("%s", newAthlete->athleteID);

    printf("Enter athlete name: ");
    scanf("%s", newAthlete->name);

    printf("Enter athlete sport: ");
    scanf("%s", newAthlete->sport);

    printf("Enter athlete's personal best: ");
    scanf("%f", &newAthlete->personalBest);

    printf("Enter athlete's last performance: ");
    scanf("%f", &newAthlete->lastPerformance);

    (*numAthletes)++;  // Increase athlete count
    printf("Athlete added successfully!\n");
}

void updateLastPerformance(struct Athlete* athletes, int numAthletes) {
    char athleteID[10];
    int found = 0;

    printf("Enter athlete ID to update performance: ");
    scanf("%s", athleteID);

    for (int i = 0; i < numAthletes; i++) {
        if (strcmp(athletes[i].athleteID, athleteID) == 0) {
            printf("Enter new performance for athlete %s: ", athletes[i].name);
            scanf("%f", &athletes[i].lastPerformance);
            printf("Last performance updated for athlete %s!\n", athletes[i].name);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Athlete with ID %s not found.\n", athleteID);
    }
}

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

void displayNewPersonalBest(struct Athlete* athletes, int numAthletes) {
    int found = 0;
    printf("\nAthletes who set a new personal best:\n");
    for (int i = 0; i < numAthletes; i++) {
        if (athletes[i].lastPerformance < athletes[i].personalBest) {
            printf("Athlete ID: %s, Name: %s, Previous Personal Best: %.2f, Last Performance: %.2f\n",
                   athletes[i].athleteID, athletes[i].name, athletes[i].personalBest, athletes[i].lastPerformance);
            found = 1;
        }
    }

    if (!found) {
        printf("No athletes have set a new personal best.\n");
    }
}
```

Problem 5: Sports Equipment Inventory System
Requirements:
Define a structure Equipment with members:
char equipmentID[10]
char name[30]
char category[20] (e.g., balls, rackets)
int quantity
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 *size, int *capacity) {
    if (*size == *capacity) {
        *capacity *= 2;
        *inventory = realloc(*inventory, *capacity * sizeof(struct Equipment));
        if (*inventory == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
    }

    struct Equipment newEquipment;

    printf("Enter Equipment ID: ");
    scanf("%s", newEquipment.equipmentID);
    printf("Enter Equipment Name: ");
    scanf("%s", newEquipment.name);
    printf("Enter Category (e.g., balls, rackets): ");
    scanf("%s", newEquipment.category);
    printf("Enter Quantity: ");
    scanf("%d", &newEquipment.quantity);
    printf("Enter Price per Unit: ");
    scanf("%f", &newEquipment.pricePerUnit);

    (*inventory)[*size] = newEquipment;
    (*size)++;
}

void updateQuantity(struct Equipment *inventory, int size, char *equipmentID, int newQuantity) {
    for (int i = 0; i < size; i++) {
        if (strcmp(inventory[i].equipmentID, equipmentID) == 0) {
            inventory[i].quantity = newQuantity;
            printf("Quantity updated successfully.\n");
            return;
        }
    }
    printf("Equipment with ID %s not found.\n", equipmentID);
}

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

float calculateTotalValue(struct Equipment *inventory, int size) {
    float totalValue = 0.0;
    for (int i = 0; i < size; i++) {
        totalValue += inventory[i].quantity * inventory[i].pricePerUnit;
    }
    return totalValue;
}

int main() {
    struct Equipment *inventory = NULL;
    int size = 0, capacity = 2;

    inventory = (struct Equipment *)malloc(capacity * sizeof(struct Equipment));
    if (inventory == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    int choice;
    do {
        printf("\n1. Add New Equipment\n");
        printf("2. Update Quantity\n");
        printf("3. Display Equipment by Category\n");
        printf("4. Calculate Total Value of Inventory\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addEquipment(&inventory, &size, &capacity);
                break;
            case 2: {
                char equipmentID[10];
                int newQuantity;
                printf("Enter Equipment ID to update: ");
                scanf("%s", equipmentID);
                printf("Enter new quantity: ");
                scanf("%d", &newQuantity);
                updateQuantity(inventory, size, equipmentID, newQuantity);
                break;
            }
            case 3: {
                char category[20];
                printf("Enter category to display (e.g., balls, rackets): ");
                scanf("%s", category);
                displayByCategory(inventory, size, category);
                break;
            }
            case 4:
                printf("Total value of inventory: %.2f\n", calculateTotalValue(inventory, size));
                break;
            case 5:
                printf("Exiting the system.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);
    free(inventory);

    return 0;
}
```

THIRD SET OF PROBLEMS
================================

Problem 1: Research Paper Database Management
Requirements:
Define a structure ResearchPaper with the following members:
char title[100]
char author[50]
char journal[50]
int year
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>

#define MAX_TITLE 100
#define MAX_AUTHOR 50
#define MAX_JOURNAL 50
#define MAX_DOI 30

// Define the ResearchPaper structure without typedef
struct ResearchPaper {
    char title[MAX_TITLE];
    char author[MAX_AUTHOR];
    char journal[MAX_JOURNAL];
    int year;
    char DOI[MAX_DOI];
};

// Function to add a new research paper to the database
void addPaper(struct ResearchPaper **papers, int *count, struct ResearchPaper newPaper) {
    *count += 1;
    *papers = realloc(*papers, (*count) * sizeof(struct ResearchPaper)); // Resize the array
    if (*papers == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    (*papers)[*count - 1] = newPaper;
}

// Function to update the details of an existing paper using its DOI
void updatePaper(struct ResearchPaper *papers, int count, const char *DOI, struct ResearchPaper
updatedPaper) {
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].DOI, DOI) == 0) {
            papers[i] = updatedPaper;
            printf("Paper with DOI %s has been updated.\n", DOI);
            return;
        }
    }
    printf("Paper with DOI %s not found.\n", DOI);
}

// Function to display all papers published in a specific journal
void displayPapersByJournal(struct ResearchPaper *papers, int count, const char *journal) {
    printf("Papers published in journal: %s\n", journal);
    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);
        }
    }
}

// Function to find and display the most recent papers by a specific author
void displayRecentPapersByAuthor(struct ResearchPaper *papers, int count, const char *author) {
    int mostRecentYear = -1;
    printf("Most recent papers by author: %s\n", author);
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0) {
            if (papers[i].year > mostRecentYear) {
                mostRecentYear = papers[i].year;
            }
        }
    }

    // Display the papers from the most recent year
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0 && papers[i].year == mostRecentYear) {
            printf("Title: %s, Author: %s, Year: %d, DOI: %s\n", papers[i].title, papers[i].author, papers[i].year,
papers[i].DOI);
        }
    }
}

int main() {
    struct ResearchPaper *papers = NULL; // Array to store research papers
    int count = 0; // Number of papers in the array

    // Example of adding a new paper
    struct ResearchPaper paper1 = {"Quantum Computing Advances", "Alice Smith", "Quantum Journal",
2023, "10.1234/qc2023"};
    addPaper(&papers, &count, paper1);

    struct ResearchPaper paper2 = {"Artificial Intelligence in Healthcare", "Bob Johnson", "AI Journal",
2024, "10.2345/ai2024"};
    addPaper(&papers, &count, paper2);
    // Example of updating a paper's details
    struct ResearchPaper updatedPaper = {"Quantum Computing Breakthrough", "Alice Smith", "Quantum
Journal", 2024, "10.1234/qc2023"};
    updatePaper(papers, count, "10.1234/qc2023", updatedPaper);

    // Display papers published in a specific journal
    displayPapersByJournal(papers, count, "Quantum Journal");

    // Display the most recent papers by a specific author
    displayRecentPapersByAuthor(papers, count, "Alice Smith");

    // Free the allocated memory
    free(papers);
    return 0;
}

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

#define MAX_TITLE 100
#define MAX_AUTHOR 50
#define MAX_JOURNAL 50
#define MAX_DOI 30

// Define the ResearchPaper structure without typedef
struct ResearchPaper {
    char title[MAX_TITLE];
    char author[MAX_AUTHOR];
    char journal[MAX_JOURNAL];
    int year;
    char DOI[MAX_DOI];
};

// Function to add a new research paper to the database
void addPaper(struct ResearchPaper **papers, int *count, struct ResearchPaper newPaper) {
    *count += 1;
    *papers = realloc(*papers, (*count) * sizeof(struct ResearchPaper)); // Resize the array
    if (*papers == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    (*papers)[*count - 1] = newPaper;
}

// Function to update the details of an existing paper using its DOI
void updatePaper(struct ResearchPaper *papers, int count, const char *DOI, struct ResearchPaper
updatedPaper) {
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].DOI, DOI) == 0) {
            papers[i] = updatedPaper;
            printf("Paper with DOI %s has been updated.\n", DOI);
            return;
        }
    }
    printf("Paper with DOI %s not found.\n", DOI);
}

// Function to display all papers published in a specific journal
void displayPapersByJournal(struct ResearchPaper *papers, int count, const char *journal) {
    printf("Papers published in journal: %s\n", journal);
    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);
        }
    }
}

// Function to find and display the most recent papers by a specific author
void displayRecentPapersByAuthor(struct ResearchPaper *papers, int count, const char *author) {
    int mostRecentYear = -1;
    printf("Most recent papers by author: %s\n", author);
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0) {
            if (papers[i].year > mostRecentYear) {
                mostRecentYear = papers[i].year;
            }
        }
    }

    // Display the papers from the most recent year
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0 && papers[i].year == mostRecentYear) {
            printf("Title: %s, Author: %s, Year: %d, DOI: %s\n", papers[i].title, papers[i].author, papers[i].year,
papers[i].DOI);
        }
    }
}

int main() {
    struct ResearchPaper *papers = NULL; // Array to store research papers
    int count = 0; // Number of papers in the array

    // Example of adding a new paper
    struct ResearchPaper paper1 = {"Quantum Computing Advances", "Alice Smith", "Quantum Journal",
2023, "10.1234/qc2023"};
    addPaper(&papers, &count, paper1);

    struct ResearchPaper paper2 = {"Artificial Intelligence in Healthcare", "Bob Johnson", "AI Journal",
2024, "10.2345/ai2024"};
    addPaper(&papers, &count, paper2);

    // Example of updating a paper's details
    struct ResearchPaper updatedPaper = {"Quantum Computing Breakthrough", "Alice Smith", "Quantum
Journal", 2024, "10.1234/qc2023"};
    updatePaper(papers, count, "10.1234/qc2023", updatedPaper);

    // Display papers published in a specific journal
    displayPapersByJournal(papers, count, "Quantum Journal");

    // Display the most recent papers by a specific author
    displayRecentPapersByAuthor(papers, count, "Alice Smith");

    // Free the allocated memory
    free(papers);
    return 0;
}

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

#define MAX_TITLE 100
#define MAX_AUTHOR 50
#define MAX_JOURNAL 50
#define MAX_DOI 30

// Define the ResearchPaper structure without typedef
struct ResearchPaper {
    char title[MAX_TITLE];
    char author[MAX_AUTHOR];
    char journal[MAX_JOURNAL];
    int year;
    char DOI[MAX_DOI];
};

// Function to add a new research paper to the database
void addPaper(struct ResearchPaper **papers, int *count, struct ResearchPaper newPaper) {
    *count += 1;
    *papers = realloc(*papers, (*count) * sizeof(struct ResearchPaper)); // Resize the array
    if (*papers == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    (*papers)[*count - 1] = newPaper;
}

// Function to update the details of an existing paper using its DOI
void updatePaper(struct ResearchPaper *papers, int count, const char *DOI, struct ResearchPaper
updatedPaper) {
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].DOI, DOI) == 0) {
            papers[i] = updatedPaper;
            printf("Paper with DOI %s has been updated.\n", DOI);
            return;
        }
    }
    printf("Paper with DOI %s not found.\n", DOI);
}

// Function to display all papers published in a specific journal
void displayPapersByJournal(struct ResearchPaper *papers, int count, const char *journal) {
    printf("Papers published in journal: %s\n", journal);
    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);
        }
    }
}

// Function to find and display the most recent papers by a specific author
void displayRecentPapersByAuthor(struct ResearchPaper *papers, int count, const char *author) {
    int mostRecentYear = -1;
    printf("Most recent papers by author: %s\n", author);
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0) {
            if (papers[i].year > mostRecentYear) {
                mostRecentYear = papers[i].year;
            }
        }
    }

    // Display the papers from the most recent year
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0 && papers[i].year == mostRecentYear) {
            printf("Title: %s, Author: %s, Year: %d, DOI: %s\n", papers[i].title, papers[i].author, papers[i].year,
papers[i].DOI);
        }
    }
}

int main() {
    struct ResearchPaper *papers = NULL; // Array to store research papers
    int count = 0; // Number of papers in the array

    // Example of adding a new paper
    struct ResearchPaper paper1 = {"Quantum Computing Advances", "Alice Smith", "Quantum Journal",
2023, "10.1234/qc2023"};
    addPaper(&papers, &count, paper1);

    struct ResearchPaper paper2 = {"Artificial Intelligence in Healthcare", "Bob Johnson", "AI Journal",
2024, "10.2345/ai2024"};
    addPaper(&papers, &count, paper2);

    // Example of updating a paper's details
    struct ResearchPaper updatedPaper = {"Quantum Computing Breakthrough", "Alice Smith", "Quantum
Journal", 2024, "10.1234/qc2023"};
    updatePaper(papers, count, "10.1234/qc2023", updatedPaper);

    // Display papers published in a specific journal
    displayPapersByJournal(papers, count, "Quantum Journal");

    // Display the most recent papers by a specific author
    displayRecentPapersByAuthor(papers, count, "Alice Smith");

    // Free the allocated memory
    free(papers);
    return 0;
}
```

Problem 2: Experimental Data Logger
Requirements:
Create a structure Experiment with members:
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)
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>

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

// Function to log a new experiment
void logExperiment(struct Experiment **experiments, int *count, char *experimentID, char *researcher,
char *startDate, char *endDate) {
    // Resize the experiments array dynamically if needed
    *experiments = realloc(*experiments, (*count + 1) * sizeof(struct Experiment));

    // Input experiment details
    strncpy((*experiments)[*count].experimentID, experimentID,
sizeof((*experiments)[*count].experimentID) - 1);
    strncpy((*experiments)[*count].researcher, researcher, sizeof((*experiments)[*count].researcher) - 1);
    strncpy((*experiments)[*count].startDate, startDate, sizeof((*experiments)[*count].startDate) - 1);
    strncpy((*experiments)[*count].endDate, endDate, sizeof((*experiments)[*count].endDate) - 1);

    // Initialize results to zero
    for (int i = 0; i < 10; i++) {
        (*experiments)[*count].results[i] = 0.0;
    }

    // Increment experiment count
    (*count)++;
}

// Function to update the result readings of an experiment
void updateResults(struct Experiment *experiment, float results[10]) {
    for (int i = 0; i < 10; i++) {
        experiment->results[i] = results[i];
    }
}

// Function to display all experiments conducted by a specific researcher
void displayExperimentsByResearcher(struct Experiment *experiments, int count, char *researcher) {
    printf("Experiments conducted by %s:\n", researcher);
    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].researcher, researcher) == 0) {
            printf("Experiment ID: %s, Start Date: %s, End Date: %s\n",
                   experiments[i].experimentID,
                   experiments[i].startDate,
                   experiments[i].endDate);
        }
    }
}

// Function to calculate and display the average result for a specific experiment
void displayAverageResult(struct Experiment *experiment) {
    float sum = 0.0;
    int count = 0;

    // Calculate sum of results
    for (int i = 0; i < 10; i++) {
        if (experiment->results[i] != 0.0) {
            sum += experiment->results[i];
            count++;
        }
    }

    // Calculate and display average
    if (count > 0) {
        printf("Average result for experiment %s: %.2f\n", experiment->experimentID, sum / count);
    } else {
        printf("No results recorded for experiment %s.\n", experiment->experimentID);
    }
}

int main() {
    struct Experiment *experiments = NULL; // Dynamically allocated array for experiments
    int experimentCount = 0; // Keep track of the number of experiments

    // Example of logging experiments
    logExperiment(&experiments, &experimentCount, "EXP001", "Dr. Smith", "2025-01-01", "2025-01-10");
    logExperiment(&experiments, &experimentCount, "EXP002", "Dr. Jones", "2025-01-05",
"2025-01-15");

    // Example of updating experiment results
    float results1[10] = {1.5, 2.3, 3.7, 4.2, 5.0, 0, 0, 0, 0, 0};
    updateResults(&experiments[0], results1);

    float results2[10] = {2.1, 3.4, 4.8, 0, 0, 0, 0, 0, 0, 0};
    updateResults(&experiments[1], results2);
    // Display all experiments by Dr. Smith
    displayExperimentsByResearcher(experiments, experimentCount, "Dr. Smith");

    // Display the average result for EXP001
    displayAverageResult(&experiments[0]);

    // Free dynamically allocated memory
    free(experiments);

    return 0;
}
```

Problem 3: Grant Application Tracker
Requirements:
Define a structure GrantApplication with the following members:
char applicationID[10]
char applicantName[50]
char projectTitle[100]
float requestedAmount
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>

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

// Function to add a new grant application
void addGrantApplication(struct GrantApplication **applications, int *count, int *capacity) {
    // If capacity is reached, resize the array
    if (*count == *capacity) {
        *capacity *= 2;
        *applications = realloc(*applications, *capacity * sizeof(struct GrantApplication));
    }

    // Take input for the new application
    struct GrantApplication newApp;

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

    // Add the new application to the array
    (*applications)[*count] = newApp;
    (*count)++;
}

// Function to update the status of a grant application
void updateStatus(struct GrantApplication *applications, int count, char *applicationID, char *newStatus) {
    for (int i = 0; i < count; i++) {
        if (strcmp(applications[i].applicationID, applicationID) == 0) {
            strcpy(applications[i].status, newStatus);
            printf("Status of application %s updated to %s\n", applicationID, newStatus);
            return;
        }
    }
    printf("Application ID %s not found.\n", applicationID);
}

// Function to display all applications requesting an amount greater than a specified value
void displayApplicationsGreaterThan(struct GrantApplication *applications, int count, float amount) {
    printf("\nApplications requesting more than %.2f:\n", amount);
    for (int i = 0; i < count; i++) {
        if (applications[i].requestedAmount > amount) {
            printf("ID: %s, Applicant: %s, Project: %s, Requested Amount: %.2f, Status: %s\n",
                    applications[i].applicationID, applications[i].applicantName,
                    applications[i].projectTitle, applications[i].requestedAmount,
                    applications[i].status);
        }
    }
}

// Function to find and display applications that are currently "Approved"
void displayApprovedApplications(struct GrantApplication *applications, int count) {
    printf("\nApproved Applications:\n");
    for (int i = 0; i < count; i++) {
        if (strcmp(applications[i].status, "Approved") == 0) {
            printf("ID: %s, Applicant: %s, Project: %s, Requested Amount: %.2f\n",
                    applications[i].applicationID, applications[i].applicantName,
                    applications[i].projectTitle, applications[i].requestedAmount);
        }
    }
}

int main() {
    int capacity = 2;  // Initial capacity of the array
    int count = 0;  // The number of applications added
    struct GrantApplication *applications = malloc(capacity * sizeof(struct GrantApplication));  //
Dynamically allocated array
```

```
    int choice;
    while (1) {
        printf("\nGrant Application Tracker Menu:\n");
        printf("1. Add a new grant application\n");
        printf("2. Update application status\n");
        printf("3. Display applications requesting more than a certain amount\n");
        printf("4. Display approved applications\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            addGrantApplication(&applications, &count, &capacity);
        } else if (choice == 2) {
            char applicationID[10], newStatus[20];
            printf("\nEnter Application ID to update: ");
            scanf("%s", applicationID);
            printf("Enter new status: ");
            scanf("%s", newStatus);
            updateStatus(applications, count, applicationID, newStatus);
        } else if (choice == 3) {
            float amount;
            printf("\nEnter the amount to filter applications: ");
            scanf("%f", &amount);
            displayApplicationsGreaterThan(applications, count, amount);
        } else if (choice == 4) {
            displayApprovedApplications(applications, count);
        } else if (choice == 5) {
            break;
        } else {
            printf("Invalid choice! Try again.\n");
        }
    }

    // Free dynamically allocated memory
    free(applications);
    return 0;
}
```

Problem 4: Research Collaborator Management
Requirements:
Create a structure Collaborator with members:
char collaboratorID[10]
char name[50]
char institution[50]
char expertiseArea[30]
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.

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

// Structure to store collaborator information
struct Collaborator {
    char collaboratorID[10];
    char name[50];
    char institution[50];
    char expertiseArea[30];
    int numberOfProjects;
};

// Function to add a new collaborator to the database
void addCollaborator(struct Collaborator **collaborators, int *size, int *capacity) {
    if (*size == *capacity) {
        // Reallocate memory if the array is full
        *capacity *= 2;
        *collaborators = realloc(*collaborators, (*capacity) * sizeof(struct Collaborator));
        if (*collaborators == NULL) {
            printf("Memory allocation failed\n");
            exit(1);
        }
    }

    // Read the collaborator information
    printf("Enter collaborator ID: ");
    scanf("%s", (*collaborators)[*size].collaboratorID);
    printf("Enter name: ");
    scanf(" %[^\n]%*c", (*collaborators)[*size].name);
    printf("Enter institution: ");
    scanf(" %[^\n]%*c", (*collaborators)[*size].institution);
    printf("Enter expertise area: ");
    scanf(" %[^\n]%*c", (*collaborators)[*size].expertiseArea);
    printf("Enter number of projects: ");
    scanf("%d", &(*collaborators)[*size].numberOfProjects);

    // Increase the size of the collaborator array
    (*size)++;
}

// Function to update the number of projects a collaborator is involved in
void updateProjects(struct Collaborator *collaborators, int size, char *collaboratorID, int newProjectCount)
{
    for (int i = 0; i < size; i++) {
        if (strcmp(collaborators[i].collaboratorID, collaboratorID) == 0) {
            collaborators[i].numberOfProjects = newProjectCount;
            printf("Updated number of projects for collaborator %s to %d\n", collaboratorID,
newProjectCount);
            return;
        }
    }
    printf("Collaborator with ID %s not found.\n", collaboratorID);
}
```

```
// Function to display all collaborators from a specific institution
void displayByInstitution(struct Collaborator *collaborators, int size, char *institution) {
    int displayed = 0;
    for (int i = 0; i < size; i++) {
        if (strcmp(collaborators[i].institution, institution) == 0) {
            printf("ID: %s, Name: %s, Expertise: %s, Projects: %d\n",
                   collaborators[i].collaboratorID,
                   collaborators[i].name,
                   collaborators[i].expertiseArea,
                   collaborators[i].numberOfProjects);
            displayed = 1;
        }
    }
    if (displayed == 0) {
        printf("No collaborators found for institution: %s\n", institution);
    }
}

// Function to find collaborators with expertise in a given area
void findByExpertise(struct Collaborator *collaborators, int size, char *expertiseArea) {
    int displayed = 0;
    for (int i = 0; i < size; i++) {
        if (strcmp(collaborators[i].expertiseArea, expertiseArea) == 0) {
            printf("ID: %s, Name: %s, Institution: %s, Projects: %d\n",
                   collaborators[i].collaboratorID,
                   collaborators[i].name,
                   collaborators[i].institution,
                   collaborators[i].numberOfProjects);
            displayed = 1;
        }
    }
    if (displayed == 0) {
        printf("No collaborators found with expertise in: %s\n", expertiseArea);
    }
}

int main() {
    struct Collaborator *collaborators = NULL;
    int size = 0;
    int capacity = 2; // Initial capacity
    collaborators = malloc(capacity * sizeof(struct Collaborator));
    if (collaborators == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    int choice;
    char collaboratorID[10];
    char institution[50];
    char expertiseArea[30];
    int newProjectCount;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Add Collaborator\n");
        printf("2. Update Projects\n");
        printf("3. Display Collaborators by Institution\n");
        printf("4. Find Collaborators by Expertise\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addCollaborator(&collaborators, &size, &capacity);
                break;
            case 2:
                printf("Enter collaborator ID to update projects: ");
                scanf("%s", collaboratorID);
                printf("Enter new number of projects: ");
                scanf("%d", &newProjectCount);
                updateProjects(collaborators, size, collaboratorID, newProjectCount);
                break;
            case 3:
                printf("Enter institution name: ");
                scanf(" %[^\n]%*c", institution);
                displayByInstitution(collaborators, size, institution);
                break;
            case 4:
                printf("Enter expertise area: ");
                scanf(" %[^\n]%*c", expertiseArea);
                findByExpertise(collaborators, size, expertiseArea);
                break;
            case 5:
                free(collaborators);
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

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>

#define INITIAL_SIZE 10

// Structure to represent a conference submission
typedef struct {
    char submissionID[10];
    char authorName[50];
    char paperTitle[100];
    char conferenceName[50];
    char submissionDate[11];
    char status[20];
} ConferenceSubmission;

// Function to add a new conference submission
void addSubmission(ConferenceSubmission **submissions, int *count, int *capacity) {
    if (*count == *capacity) {
        *capacity *= 2;
        *submissions = realloc(*submissions, *capacity * sizeof(ConferenceSubmission));
        if (*submissions == NULL) {
            printf("Memory allocation failed.\n");
            exit(1);
        }
    }

    ConferenceSubmission newSubmission;
    printf("Enter Submission ID: ");
    scanf("%s", newSubmission.submissionID);
    printf("Enter Author Name: ");
    scanf(" %[^\n]", newSubmission.authorName);  // Read with spaces
    printf("Enter Paper Title: ");
    scanf(" %[^\n]", newSubmission.paperTitle);
    printf("Enter Conference Name: ");
    scanf(" %[^\n]", newSubmission.conferenceName);
    printf("Enter Submission Date (YYYY-MM-DD): ");
    scanf("%s", newSubmission.submissionDate);
    printf("Enter Status (Pending, Accepted, Rejected): ");
    scanf("%s", newSubmission.status);

    (*submissions)[*count] = newSubmission;
    (*count)++;
}

// Function to update the status of a submission
void updateStatus(ConferenceSubmission *submissions, int count, const char *submissionID, const char
*newStatus) {
    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].submissionID, submissionID) == 0) {
            strcpy(submissions[i].status, newStatus);
            printf("Status updated successfully for submission ID %s.\n", submissionID);
            return;
        }
```

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

// Function to display all submissions for a specific conference
void displaySubmissionsByConference(ConferenceSubmission *submissions, int count, const char
*conferenceName) {
    printf("Submissions for 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: %s\n", submissions[i].authorName);
            printf("Title: %s\n", submissions[i].paperTitle);
            printf("Date: %s\n", submissions[i].submissionDate);
            printf("Status: %s\n\n", submissions[i].status);
        }
    }
}

// Function to display all submissions by a specific author
void displaySubmissionsByAuthor(ConferenceSubmission *submissions, int count, const char
*authorName) {
    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("Title: %s\n", submissions[i].paperTitle);
            printf("Conference: %s\n", submissions[i].conferenceName);
            printf("Date: %s\n", submissions[i].submissionDate);
            printf("Status: %s\n\n", submissions[i].status);
        }
    }
}

int main() {
    int count = 0;    // Current number of submissions
    int capacity = INITIAL_SIZE;    // Initial capacity of array
    ConferenceSubmission *submissions = malloc(capacity * sizeof(ConferenceSubmission));

    if (submissions == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    int choice;
    while (1) {
        printf("\nScientific Conference Submission Tracker\n");
        printf("1. Add a new submission\n");
        printf("2. Update the status of a submission\n");
        printf("3. Display all submissions for a specific conference\n");
        printf("4. Display all submissions by a specific author\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                addSubmission(&submissions, &count, &capacity);
                break;
            case 2:
                {
                    char submissionID[10];
                    char newStatus[20];
                    printf("Enter Submission ID: ");
                    scanf("%s", submissionID);
                    printf("Enter new Status: ");
                    scanf("%s", newStatus);
                    updateStatus(submissions, count, submissionID, newStatus);
                }
                break;
            case 3:
                {
                    char conferenceName[50];
                    printf("Enter Conference Name: ");
                    scanf(" %[^\n]", conferenceName);
                    displaySubmissionsByConference(submissions, count, conferenceName);
                }
                break;
            case 4:
                {
                    char authorName[50];
                    printf("Enter Author Name: ");
                    scanf(" %[^\n]", authorName);
                    displaySubmissionsByAuthor(submissions, count, authorName);
                }
                break;
            case 5:
                free(submissions);
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```