Structure containing pointers

```
#include <stdio.h>


struct num {
   int a, b;
};


int sum(struct num *num1, struct num *num2);


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;
}
```

Structure to store and print student details-------------

```
#include<stdio.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(" Rollno = %d\n Name = %s\n section = %c \n maths markk = %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,"Milan");

     stud1.section ='B';

     stud1.marks_maths = 90;

     return stud1;

}
```

```c
/******************************************************************************
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.


*******************************************************************************/

#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 *count, int capacity);
void updateVehicle(struct Vehicle *fleet, int count);
void displayVehiclesAfterYear(struct Vehicle *fleet, int count, int year);
void findHighestFuelEfficiency(struct Vehicle *fleet, int count);



int main() {
    int capacity = 10; // Set the initial capacity
    struct Vehicle *fleet = malloc(capacity * sizeof(struct Vehicle));
    int count = 0;     // Number of vehicles currently in the fleet
    int choice;

    while (1) {
        printf("\nVehicle Fleet Management Menu:\n");
        printf("1. Add Vehicle\n");
        printf("2. Update Vehicle\n");
        printf("3. Display Vehicles After Year\n");
        printf("4. Find Vehicle with Highest Fuel Efficiency\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

    return 0;
}



// Function to add a new vehicle to the fleet
void addVehicle(struct Vehicle *fleet, int *count, int capacity) {
    if (*count >= capacity) {
        printf("Fleet capacity reached. Cannot add more vehicles.\n");
        return;
    }

    struct Vehicle newVehicle;
    printf("Enter registration number: ");
    scanf(" %[^\n]", newVehicle.registrationNumber);
    printf("Enter model: ");
    scanf(" %[^\n]", newVehicle.model);
    printf("Enter year of manufacture: ");
    scanf("%d", &newVehicle.yearOfManufacture);
    printf("Enter mileage: ");
    scanf("%f", &newVehicle.mileage);
    printf("Enter fuel efficiency: ");
    scanf("%f", &newVehicle.fuelEfficiency);

    fleet[*count] = newVehicle;
    (*count)++;
}

// Function to update the mileage and fuel efficiency for a vehicle
void updateVehicle(struct Vehicle *fleet, int count) {
    char registrationNumber[15];
    printf("Enter registration number of the vehicle to update: ");
    scanf(" %[^\n]", registrationNumber);
    for (int i = 0; i < count; i++) {
        if (strcmp(fleet[i].registrationNumber, registrationNumber) == 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");
}

// Function to display all vehicles manufactured after a certain year
void displayVehiclesAfterYear(struct Vehicle *fleet, int count, int year) {
    printf("Vehicles manufactured after %d:\n", year);
    for (int i = 0; i < count; i++) {
        if (fleet[i].yearOfManufacture > year) {
            printf("Registration Number: %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);
        }
    }
}

// Function to find the vehicle with the highest fuel efficiency
void findHighestFuelEfficiency(struct Vehicle *fleet, int count) {
    if (count == 0) {
        printf("No vehicles in the fleet.\n");
        return;
    }

    struct Vehicle *bestVehicle = &fleet[0];
    for (int i = 1; i < count; i++) {
        if (fleet[i].fuelEfficiency > bestVehicle->fuelEfficiency) {
            bestVehicle = &fleet[i];
        }
    }
    printf("Vehicle with highest fuel efficiency:\n");
    printf("Registration Number: %s, Model: %s, Year: %d, Mileage: %.2f, Fuel Efficiency: %.2f\n",
           bestVehicle->registrationNumber, bestVehicle->model, bestVehicle->yearOfManufacture,
bestVehicle->mileage, bestVehicle->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 <stdlib.h>

#include <string.h>

struct CarRental {

   char carID[10];

   char customerName[50];

   char rentalDate[11]; // Format: YYYY-MM-DD

   char returnDate[11];

   float rentalPricePerDay;
```

```
};

int calculateRentalDays(const char* startDate, const char* endDate);
void bookCar(struct CarRental **rentals, int *count, int *capacity);
float calculateRentalPrice(struct CarRental rental);
void displayRentals(struct CarRental *rentals, int count);
void searchRentalsByCustomer(struct CarRental *rentals, int count, const char* customerName);
int validateDateFormat(const char* date);



int main() {
    int initialCapacity = 10;
    struct CarRental *rentals = malloc(initialCapacity * sizeof(struct CarRental));
    int count = 0;
    int capacity = initialCapacity;
    int choice;

    while (1) {
        printf("\nCar Rental Reservation Menu:\n");
        printf("1. Book a Car\n");
        printf("2. Display Total Rental Price\n");
        printf("3. Display All Rentals\n");
        printf("4. Search Rentals by Customer Name\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            bookCar(&rentals, &count, &capacity);
        } else if (choice == 2) {
            if (count > 0) {
                float totalPrice = calculateRentalPrice(rentals[count - 1]);
                printf("Total Rental Price: %.2f\n", totalPrice);
            } else {
                printf("No rentals available.\n");
            }
        } else if (choice == 3) {
            displayRentals(rentals, count);
        } else if (choice == 4) {
            char customerName[50];
            printf("Enter customer name: ");
            scanf(" %[^\n]", customerName);
            searchRentalsByCustomer(rentals, count, customerName);
        } else if (choice == 5) {
            free(rentals);
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

// Function to calculate the number of rental days
int calculateRentalDays(const char* startDate, const char* endDate) {
    int startYear, startMonth, startDay;
    int endYear, endMonth, endDay;
    sscanf(startDate, "%d-%d-%d", &startYear, &startMonth, &startDay);
    sscanf(endDate, "%d-%d-%d", &endYear, &endMonth, &endDay);
```

```c
    // Calculate the number of days (a simple approach, not accounting for all calendar nuances)
    return (endYear - startYear) * 365 + (endMonth - startMonth) * 30 + (endDay - startDay);
}



// Function to book a car for a customer
void bookCar(struct CarRental **rentals, int *count, int *capacity) {
    if (*count >= *capacity) {
        *capacity *= 2;
        *rentals = realloc(*rentals, (*capacity) * sizeof(struct CarRental));
    }

    struct CarRental newRental;
    printf("Enter car ID: ");
    scanf(" %[^\n]", newRental.carID);
    printf("Enter customer name: ");
    scanf(" %[^\n]", newRental.customerName);
    printf("Enter rental date (YYYY-MM-DD): ");
    scanf(" %[^\n]", newRental.rentalDate);
    printf("Enter return date (YYYY-MM-DD): ");
    scanf(" %[^\n]", newRental.returnDate);
    printf("Enter rental price per day: ");
    scanf("%f", &newRental.rentalPricePerDay);

    (*rentals)[*count] = newRental;
    (*count)++;
}

// Function to calculate the total rental price
float calculateRentalPrice(struct CarRental rental) {
    int rentalDays = calculateRentalDays(rental.rentalDate, rental.returnDate);
    return rentalDays * rental.rentalPricePerDay;
}

// Function to display all current rentals
void displayRentals(struct CarRental *rentals, int count) {
    printf("Current Rentals:\n");
    for (int i = 0; i < count; i++) {
        printf("Car ID: %s, Customer Name: %s, Rental Date: %s, Return Date: %s, Rental Price Per Day: %.2f\n",
               rentals[i].carID, rentals[i].customerName, rentals[i].rentalDate, rentals[i].returnDate, rentals[i].rentalPricePerDay);
    }
}

// Function to search for rentals by customer name
void searchRentalsByCustomer(struct CarRental *rentals, int count, const char* customerName) {
    printf("Rentals for customer %s:\n", customerName);
    for (int i = 0; i < count; i++) {
        if (strcmp(rentals[i].customerName, customerName) == 0) {
            printf("Car ID: %s, Rental Date: %s, Return Date: %s, Rental Price Per Day: %.2f\n",
                   rentals[i].carID, rentals[i].rentalDate, rentals[i].returnDate, rentals[i].rentalPricePerDay);
        }
    }
}

// Function to validate date format
int validateDateFormat(const char* date) {
    int year, month, day;
    if (sscanf(date, "%d-%d-%d", &year, &month, &day) == 3) {
        return (year >= 0 && month >= 1 && month <= 12 && day >= 1 && day <= 31);
    }
    return 0;
}
/*****************************************************************************
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: \n");
    scanf("%f", &data[*count].speed);
    printf("Latitude: \n");
    scanf("%f", &data[*count].latitude);
    printf("Longitude: \n");
    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
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
float avgSpeed = calculateAverageSpeed(data, count, startTime, endTime);
printf("Average speed between %s and %s: %.2f\n", startTime, endTime, avgSpeed);

free(data);
```

```
    return 0;

}



/*
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 linked lists to store and manage performance data entries.*/






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

// Define the structure
struct EnginePerformance {
    char engineID[10];

    float temperature;

    float rpm;

    float fuelConsumptionRate;

    float oilPressure;

    struct EnginePerformance* next;

};


// Function to create a new performance data node

struct EnginePerformance* createNode(const char* engineID, float temperature, float rpm, float
fuelConsumptionRate, float oilPressure) {

    struct EnginePerformance* newNode = (struct EnginePerformance*)malloc(sizeof(struct
EnginePerformance));

    strcpy(newNode->engineID, engineID);

    newNode->temperature = temperature;

    newNode->rpm = rpm;

    newNode->fuelConsumptionRate = fuelConsumptionRate;

    newNode->oilPressure = oilPressure;

    newNode->next = NULL;

    return newNode;

}


// Function to add performance data for a specific engine

void addPerformanceData(struct EnginePerformance** head, const char* engineID, float
temperature, float rpm, float fuelConsumptionRate, float oilPressure) {

    struct EnginePerformance* newNode = createNode(engineID, temperature, rpm,
fuelConsumptionRate, oilPressure);

    newNode->next = *head;

    *head = newNode;

}


// Function to display all performance data for a specific engine ID
```

```
void displayPerformanceData(struct EnginePerformance* head, const char* engineID) {
    struct EnginePerformance* temp = head;
    printf("Performance data for engine ID %s:\n", engineID);
    while (temp != NULL) {
        if (strcmp(temp->engineID, engineID) == 0) {
            printf("Temperature: %.2f, RPM: %.2f, Fuel Consumption Rate: %.2f, Oil Pressure: %.2f\n",
temp->temperature, temp->rpm, temp->fuelConsumptionRate, temp->oilPressure);
        }
        temp = temp->next;
    }
}

// Function to calculate the average temperature and RPM for a specific engine
void calculateAverageTempAndRPM(struct EnginePerformance* head, const char* engineID) {
    struct EnginePerformance* temp = head;
    float totalTemp = 0.0, totalRPM = 0.0;
    int count = 0;
    while (temp != NULL) {
        if (strcmp(temp->engineID, engineID) == 0) {
            totalTemp += temp->temperature;
            totalRPM += temp->rpm;
            count++;
        }
        temp = temp->next;
    }
    if (count > 0) {
        printf("Average Temperature for engine ID %s: %.2f\n", engineID, totalTemp / count);
        printf("Average RPM for engine ID %s: %.2f\n", engineID, totalRPM / count);
    } else {
        printf("No performance data found for engine ID %s.\n", engineID);
    }
```

```
}


// Function to identify any engine with abnormal oil pressure
void identifyAbnormalOilPressure(struct EnginePerformance* head, float minPressure, float
maxPressure) {
    struct EnginePerformance* temp = head;
    printf("Engines with abnormal oil pressure (less than %.2f or greater than %.2f):\n", minPressure,
maxPressure);
    while (temp != NULL) {
        if (temp->oilPressure < minPressure || temp->oilPressure > maxPressure) {
            printf("Engine ID: %s, Oil Pressure: %.2f\n", temp->engineID, temp->oilPressure);
        }
        temp = temp->next;
    }
}


int main() {
    struct EnginePerformance* head = NULL;
    int choice;
    char engineID[10];
    float temperature, rpm, fuelConsumptionRate, oilPressure;
    float minPressure, maxPressure;


    while (1) {
        printf("\nEngine Performance Monitoring Menu:\n");
        printf("1. Add Performance Data\n");
        printf("2. Display Performance Data\n");
        printf("3. Calculate Average Temperature and RPM\n");
        printf("4. Identify Abnormal Oil Pressure\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
if (choice == 1) {
    printf("Enter engine ID: ");
    scanf(" %s", engineID);
    printf("Enter temperature: ");
    scanf("%f", &temperature);
    printf("Enter RPM: ");
    scanf("%f", &rpm);
    printf("Enter fuel consumption rate: ");
    scanf("%f", &fuelConsumptionRate);
    printf("Enter oil pressure: ");
    scanf("%f", &oilPressure);
    addPerformanceData(&head, engineID, temperature, rpm, fuelConsumptionRate, oilPressure);
} else if (choice == 2) {
    printf("Enter engine ID: ");
    scanf(" %[^\n]", engineID);
    displayPerformanceData(head, engineID);
} else if (choice == 3) {
    printf("Enter engine ID: ");
    scanf(" %[^\n]", engineID);
    calculateAverageTempAndRPM(head, engineID);
} else if (choice == 4) {
    printf("Enter minimum oil pressure: ");
    scanf("%f", &minPressure);
    printf("Enter maximum oil pressure: ");
    scanf("%f", &maxPressure);
    identifyAbnormalOilPressure(head, minPressure, maxPressure);
} else if (choice == 5) {
    while (head != NULL) {
        struct EnginePerformance* temp = head;
        head = head->next;
```

```
                free(temp);
            }
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! \n");
        }
    }

    return 0;
}


/****************************************************************************

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>



struct ServiceRecord {

    char serviceID[10];

    char vehicleID[15];

    char serviceDate[11];

    char description[100];

    float serviceCost;

};



void addServiceRecord(struct ServiceRecord **records, int *count, int *capacity);

void displayServiceRecords(struct ServiceRecord *records, int count, const char *vehicleID);

float calculateTotalServiceCost(struct ServiceRecord *records, int count, const char *vehicleID);

int compareByDate(const void *a, const void *b);

void sortAndDisplayServiceRecords(struct ServiceRecord *records, int count);




int main() {

    int initialCapacity = 10;

    struct ServiceRecord *records = malloc(initialCapacity * sizeof(struct ServiceRecord));

    int count = 0;

    int capacity = initialCapacity;

    int choice;


    while (1) {

        printf("\nVehicle Service History Tracker Menu:\n");

        printf("1. Add Service Record\n");
        printf("2. Display Service Records by Vehicle ID\n");
        printf("3. Calculate Total Service Cost for a Vehicle\n");
        printf("4. Sort and Display Service Records by Date\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            addServiceRecord(&records, &count, &capacity);
        } else if (choice == 2) {
            char vehicleID[15];
            printf("Enter vehicle ID: ");
            scanf(" %[^\n]", vehicleID);
            displayServiceRecords(records, count, vehicleID);
        } else if (choice == 3) {
            char vehicleID[15];
            printf("Enter vehicle ID: ");
            scanf(" %[^\n]", vehicleID);
            float totalCost = calculateTotalServiceCost(records, count, vehicleID);
            printf("Total Service Cost for vehicle ID %s: %.2f\n", vehicleID, totalCost);
        } else if (choice == 4) {
            sortAndDisplayServiceRecords(records, count);
        } else if (choice == 5) {
            free(records);
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! Please try again.\n");
        }
    }
```

```
    return 0;
}



// Function to add a new service record for a vehicle
void addServiceRecord(struct ServiceRecord **records, int *count, int *capacity) {
    if (*count >= *capacity) {
        *capacity *= 2;
        *records = realloc(*records, (*capacity) * sizeof(struct ServiceRecord));
    }

    struct ServiceRecord newRecord;
    printf("Enter service ID: ");
    scanf(" %[^\n]", newRecord.serviceID);
    printf("Enter vehicle ID: ");
    scanf(" %[^\n]", newRecord.vehicleID);
    printf("Enter service date (YYYY-MM-DD): ");
    scanf(" %[^\n]", newRecord.serviceDate);
    printf("Enter description: ");
    scanf(" %[^\n]", newRecord.description);
    printf("Enter service cost: ");
    scanf("%f", &newRecord.serviceCost);

    (*records)[*count] = newRecord;
    (*count)++;
}

// Function to display all service records for a given vehicle ID
void displayServiceRecords(struct ServiceRecord *records, int count, const char *vehicleID) {
    printf("Service records for vehicle ID %s:\n", vehicleID);
    for (int i = 0; i < count; i++) {
        if (strcmp(records[i].vehicleID, vehicleID) == 0) {
            printf("Service ID: %s, Service Date: %s, Description: %s, Service Cost: %.2f\n",
                   records[i].serviceID, records[i].serviceDate, records[i].description, records[i].serviceCost);
        }
    }
}

// Function to calculate the total cost of services for a vehicle
float calculateTotalServiceCost(struct ServiceRecord *records, int count, const char *vehicleID) {
    float totalCost = 0.0;
    for (int i = 0; i < count; i++) {
        if (strcmp(records[i].vehicleID, vehicleID) == 0) {
            totalCost += records[i].serviceCost;
        }
    }
    return totalCost;
}

// Function to compare service records by service date
int compareByDate(const void *a, const void *b) {
    struct ServiceRecord *recordA = (struct ServiceRecord *)a;
    struct ServiceRecord *recordB = (struct ServiceRecord *)b;
    return strcmp(recordA->serviceDate, recordB->serviceDate);
}

// Function to sort and display service records by service date
void sortAndDisplayServiceRecords(struct ServiceRecord *records, int count) {
    qsort(records, count, sizeof(struct ServiceRecord), compareByDate);
    printf("Sorted service records by date:\n");
    for (int i = 0; i < count; i++) {
        printf("Service ID: %s, Vehicle ID: %s, Service Date: %s, Description: %s, Service Cost: %.2f\n",
            records[i].serviceID, records[i].vehicleID, records[i].serviceDate, records[i].description,
records[i].serviceCost);
   }
}
```

```
/******************************************************************************
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>
```

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

void addPlayer(struct Player **players, int *count, int *capacity);
void updatePlayerStats(struct Player *players, int count);
void displayPlayersByTeam(struct Player *players, int count, const char *team);
void findTopPlayers(struct Player *players, int count);




int main() {
    int initialCapacity = 10;
    struct Player *players = malloc(initialCapacity * sizeof(struct Player));
    int count = 0;
    int capacity = initialCapacity;
    int choice;

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

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

    return 0;
}



// Function to add a new player to the system
```

```
void addPlayer(struct Player **players, int *count, int *capacity) {
    if (*count >= *capacity) {
        *capacity *= 2;
        *players = realloc(*players, (*capacity) * sizeof(struct Player));
    }

    struct Player newPlayer;
    printf("Enter player name: ");
    scanf(" %[^\n]", newPlayer.name);
    printf("Enter player age: ");
    scanf("%d", &newPlayer.age);
    printf("Enter team name: ");
    scanf(" %[^\n]", newPlayer.team);
    printf("Enter number of matches played: ");
    scanf("%d", &newPlayer.matchesPlayed);
    printf("Enter total runs: ");
    scanf("%d", &newPlayer.totalRuns);
    printf("Enter total wickets: ");
    scanf("%d", &newPlayer.totalWickets);

    (*players)[*count] = newPlayer;
    (*count)++;
}

// Function to update a player's statistics after a match
void updatePlayerStats(struct Player *players, int count) {
    char name[50];
    printf("Enter player name to update: ");
    scanf(" %[^\n]", name);
    for (int i = 0; i < count; i++) {
        if (strcmp(players[i].name, name) == 0) {
            printf("Enter updated number of matches played: ");
            scanf("%d", &players[i].matchesPlayed);
            printf("Enter updated total runs: ");
            scanf("%d", &players[i].totalRuns);
            printf("Enter updated total wickets: ");
            scanf("%d", &players[i].totalWickets);
            return;
        }
    }
    printf("Player not found.\n");
}

// Function to display the details of players from a specific team
void displayPlayersByTeam(struct Player *players, int count, const char *team) {
    printf("Players from team %s:\n", team);
    for (int i = 0; i < count; i++) {
        if (strcmp(players[i].team, team) == 0) {
            printf("Name: %s, Age: %d, Matches Played: %d, Total Runs: %d, Total Wickets: %d\n",
                   players[i].name, players[i].age, players[i].matchesPlayed, players[i].totalRuns,
players[i].totalWickets);
        }
    }
}

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

    for (int i = 1; i < count; i++) {
        if (players[i].totalRuns > topRunsPlayer->totalRuns) {
            topRunsPlayer = &players[i];
        }
        if (players[i].totalWickets > topWicketsPlayer->totalWickets) {
            topWicketsPlayer = &players[i];
        }
    }

    printf("Player with highest runs:\n");
    printf("Name: %s, Team: %s, Total Runs: %d\n", topRunsPlayer->name, topRunsPlayer->team,
topRunsPlayer->totalRuns);

    printf("Player with most wickets:\n");
    printf("Name: %s, Team: %s, Total Wickets: %d\n", topWicketsPlayer->name, topWicketsPlayer-
>team, topWicketsPlayer->totalWickets);
}
```

```
/****************************************************************************
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>



struct Match {
   char team1[30];
   char team2[30];
   char date[11]; // Format: YYYY-MM-DD
   char venue[50];
};
```

```c
// Function to schedule a new match between two teams
void scheduleMatch(struct Match *matches, int *count, int capacity) {
    if (*count >= capacity) {
        printf("Cannot schedule new match. Capacity reached.\n");
        return;
    }

    struct Match newMatch;
    printf("Enter team 1: ");
    scanf(" %[^\n]", newMatch.team1);
    printf("Enter team 2: ");
    scanf(" %[^\n]", newMatch.team2);
    printf("Enter date (YYYY-MM-DD): ");
    scanf(" %[^\n]", newMatch.date);
    printf("Enter venue: ");
    scanf(" %[^\n]", newMatch.venue);

    matches[*count] = newMatch;
    (*count)++;
}

// Function to display all scheduled matches
void displayMatches(struct Match *matches, int count) {
    printf("Scheduled Matches:\n");
    for (int i = 0; i < count; i++) {
        printf("Match: %s vs %s, Date: %s, Venue: %s\n",
               matches[i].team1, matches[i].team2, matches[i].date, matches[i].venue);
    }
}

// Function to search for matches scheduled on a specific date
```

```
void searchMatchesByDate(struct Match *matches, int count, const char *date) {
    printf("Matches scheduled on %s:\n", date);
    for (int i = 0; i < count; i++) {
        if (strcmp(matches[i].date, date) == 0) {
            printf("Match: %s vs %s, Venue: %s\n",
                   matches[i].team1, matches[i].team2, matches[i].venue);
        }
    }
}

// Function to cancel a match by specifying both team names and the date
void cancelMatch(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) {
            for (int j = i; j < *count - 1; j++) {
                matches[j] = matches[j + 1];
            }
            (*count)--;
            printf("Match between %s and %s on %s has been cancelled.\n", team1, team2, date);
            return;
        }
    }
    printf("Match not found.\n");
}

int main() {
    int capacity = 10; // Set initial capacity
    struct Match *matches = malloc(capacity * sizeof(struct Match));
    int count = 0; // Number of scheduled matches
    int choice;
```

```
while (1) {
    printf("\nTournament Fixture Scheduler Menu:\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);

    if (choice == 1) {
        scheduleMatch(matches, &count, capacity);
    } else if (choice == 2) {
        displayMatches(matches, count);
    } else if (choice == 3) {
        char date[11];
        printf("Enter the date (YYYY-MM-DD): ");
        scanf(" %[^\n]", date);
        searchMatchesByDate(matches, count, date);
    } else if (choice == 4) {
        char team1[30], team2[30], date[11];
        printf("Enter team 1: ");
        scanf(" %[^\n]", team1);
        printf("Enter team 2: ");
        scanf(" %[^\n]", team2);
        printf("Enter date (YYYY-MM-DD): ");
        scanf(" %[^\n]", date);
        cancelMatch(matches, &count, team1, team2, date);
    } else if (choice == 5) {
        free(matches);
```

```c
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}



/****************************************************************************
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>
```

```
struct CountryMedalTally {
    char country[30];
    int gold;
    int silver;
    int bronze;
};

// Function to add a new country's medal tally
void addCountry(struct CountryMedalTally **tally, int *count, int *capacity) {
    if (*count >= *capacity) {
        *capacity *= 2;
        *tally = realloc(*tally, (*capacity) * sizeof(struct CountryMedalTally));
    }

    struct CountryMedalTally newCountry;
    printf("Enter country name: ");
    scanf(" %[^\n]", newCountry.country);
    printf("Enter gold medals: ");
    scanf("%d", &newCountry.gold);
    printf("Enter silver medals: ");
    scanf("%d", &newCountry.silver);
    printf("Enter bronze medals: ");
    scanf("%d", &newCountry.bronze);

    (*tally)[*count] = newCountry;
    (*count)++;
}

// Function to update the medal count for a country
```

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

// Function to display the medal tally for all countries
void displayMedalTally(struct CountryMedalTally *tally, int count) {
    printf("Medal 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);
    }
}

// Function to find and display the country with the highest number of gold medals
void displayCountryWithHighestGold(struct CountryMedalTally *tally, int count) {
    if (count == 0) {
        printf("No countries in the tally.\n");
        return;
    }

    struct CountryMedalTally *topCountry = &tally[0];
    for (int i = 1; i < count; i++) {
        if (tally[i].gold > topCountry->gold) {
            topCountry = &tally[i];
        }
    }

    printf("Country with the highest number of gold medals:\n");
    printf("Country: %s, Gold: %d, Silver: %d, Bronze: %d\n",
           topCountry->country, topCountry->gold, topCountry->silver, topCountry->bronze);
}

int main() {
    int initialCapacity = 10;
    struct CountryMedalTally *tally = malloc(initialCapacity * sizeof(struct CountryMedalTally));
    int count = 0;
    int capacity = initialCapacity;
    int choice;

    while (1) {
        printf("\nSports Event Medal Tally Menu:\n");
        printf("1. Add Country\n");
        printf("2. Update Medal Count\n");
        printf("3. Display Medal Tally\n");
        printf("4. Display Country with Highest Gold Medals\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        if (choice == 1) {
            addCountry(&tally, &count, &capacity);
        } else if (choice == 2) {
            updateMedalCount(tally, count);
        } else if (choice == 3) {
            displayMedalTally(tally, count);
        } else if (choice == 4) {
            displayCountryWithHighestGold(tally, count);
        } else if (choice == 5) {
            free(tally);
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

```
/*****************************************************************************
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;
};

// Function to add a new athlete to the system
void addAthlete(struct Athlete **athletes, int *count, int *capacity) {
    if (*count >= *capacity) {
        *capacity *= 2;
        *athletes = realloc(*athletes, (*capacity) * sizeof(struct Athlete));
    }

    struct Athlete newAthlete;
    printf("Enter athlete ID: ");
    scanf(" %[^\n]", newAthlete.athleteID);
    printf("Enter name: ");
    scanf(" %[^\n]", newAthlete.name);
    printf("Enter sport: ");
    scanf(" %[^\n]", newAthlete.sport);
    printf("Enter personal best: ");
    scanf("%f", &newAthlete.personalBest);
    printf("Enter last performance: ");
    scanf("%f", &newAthlete.lastPerformance);

    (*athletes)[*count] = newAthlete;
    (*count)++;
}

// Function to update an athlete's last performance
void updateAthletePerformance(struct Athlete *athletes, int count) {
    char athleteID[10];
    printf("Enter athlete ID to update: ");
    scanf(" %[^\n]", athleteID);
    for (int i = 0; i < count; i++) {
        if (strcmp(athletes[i].athleteID, athleteID) == 0) {
            printf("Enter updated last performance: ");
            scanf("%f", &athletes[i].lastPerformance);
            if (athletes[i].lastPerformance > athletes[i].personalBest) {
                athletes[i].personalBest = athletes[i].lastPerformance;
            }
            return;
        }
    }
    printf("Athlete not found.\n");
}

// Function to display all athletes in a specific sport
void displayAthletesBySport(struct Athlete *athletes, int count, const char *sport) {
    printf("Athletes in sport %s:\n", sport);
    for (int i = 0; i < count; 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);
        }
    }
}

// Function to identify and display athletes who have set a new personal best in their last
performance
void displayAthletesWithNewPB(struct Athlete *athletes, int count) {
    printf("Athletes who have set a new personal best in their last performance:\n");
    for (int i = 0; i < count; i++) {
        if (athletes[i].lastPerformance == athletes[i].personalBest) {
            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);
        }
    }
}

int main() {
    int initialCapacity = 10;
    struct Athlete *athletes = malloc(initialCapacity * sizeof(struct Athlete));
    int count = 0;
    int capacity = initialCapacity;
    int choice;

    while (1) {
        printf("\nAthlete Performance Tracker Menu:\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);

        if (choice == 1) {
            addAthlete(&athletes, &count, &capacity);
        } else if (choice == 2) {
            updateAthletePerformance(athletes, count);
        } else if (choice == 3) {
            char sport[30];
            printf("Enter sport: ");
            scanf(" %[^\n]", sport);
            displayAthletesBySport(athletes, count, sport);
        } else if (choice == 4) {
            displayAthletesWithNewPB(athletes, count);
        } else if (choice == 5) {
            free(athletes);
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

```
/******************************************************************************
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;
};

// Function to add new equipment to the inventory
void addEquipment(struct Equipment **inventory, int *count, int *capacity) {
    if (*count >= *capacity) {
        *capacity *= 2;
        *inventory = realloc(*inventory, (*capacity) * sizeof(struct Equipment));
    }

    struct Equipment newEquipment;
    printf("Enter equipment ID: ");
    scanf(" %[^\n]", newEquipment.equipmentID);
    printf("Enter name: ");
    scanf(" %[^\n]", newEquipment.name);
    printf("Enter category: ");
    scanf(" %[^\n]", newEquipment.category);
    printf("Enter quantity: ");
    scanf("%d", &newEquipment.quantity);
    printf("Enter price per unit: ");
    scanf("%f", &newEquipment.pricePerUnit);

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

// Function to update the quantity of existing equipment
void updateEquipmentQuantity(struct Equipment *inventory, int count) {
    char equipmentID[10];
    printf("Enter equipment ID to update: ");
    scanf(" %[^\n]", equipmentID);
    for (int i = 0; i < count; i++) {
        if (strcmp(inventory[i].equipmentID, equipmentID) == 0) {
            printf("Enter new quantity: ");
            scanf("%d", &inventory[i].quantity);
            return;
        }
    }
    printf("Equipment not found.\n");
}

// Function to display all equipment in a specific category
void displayEquipmentByCategory(struct Equipment *inventory, int count, const char *category) {
    printf("Equipment in category %s:\n", category);
    for (int i = 0; i < count; i++) {
        if (strcmp(inventory[i].category, category) == 0) {
            printf("Equipment ID: %s, Name: %s, Quantity: %d, Price per Unit: %.2f\n",
                   inventory[i].equipmentID, inventory[i].name, inventory[i].quantity,
inventory[i].pricePerUnit);
        }
    }
}

// Function to calculate the total value of equipment in the inventory
void calculateTotalValue(struct Equipment *inventory, int count) {
    float totalValue = 0.0;
    for (int i = 0; i < count; i++) {
        totalValue += inventory[i].quantity * inventory[i].pricePerUnit;
    }
    printf("Total value of equipment in inventory: %.2f\n", totalValue);
}
```

```
int main() {
    int initialCapacity = 10;
    struct Equipment *inventory = malloc(initialCapacity * sizeof(struct Equipment));
    int count = 0;
    int capacity = initialCapacity;
    int choice;

    while (1) {
        printf("\nSports Equipment Inventory System Menu:\n");
        printf("1. Add Equipment\n");
        printf("2. Update Equipment 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);

        if (choice == 1) {
            addEquipment(&inventory, &count, &capacity);
        } else if (choice == 2) {
            updateEquipmentQuantity(inventory, count);
        } else if (choice == 3) {
            char category[20];
            printf("Enter category: ");
            scanf(" %[^\n]", category);
            displayEquipmentByCategory(inventory, count, category);
        } else if (choice == 4) {
            calculateTotalValue(inventory, count);
        } else if (choice == 5) {
            free(inventory);
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

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.

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Define the ResearchPaper structure

typedef struct {

    char title[100];

    char author[50];

    char journal[50];

    int year;

    char DOI[30];

} ResearchPaper;


// Function to add a new research paper to the database

void addResearchPaper(ResearchPaper **papers, int *count, int *capacity) {

    if (*count == *capacity) {
        // Custom array resizing
        *capacity *= 2;
        ResearchPaper *newArray = malloc(*capacity * sizeof(ResearchPaper));
        for (int i = 0; i < *count; i++) {
            newArray[i] = (*papers)[i];
        }
        free(*papers);
        *papers = newArray;
    }

    printf("Enter title: ");
    scanf("%99s", (*papers)[*count].title);
    printf("Enter author: ");
    scanf("%49s", (*papers)[*count].author);
    printf("Enter journal: ");
    scanf("%49s", (*papers)[*count].journal);
    printf("Enter year: ");
    scanf("%d", &(*papers)[*count].year);
    printf("Enter DOI: ");
    scanf("%29s", (*papers)[*count].DOI);

    (*count)++;
}

// Function to update the details of an existing paper using its DOI
void updateResearchPaper(ResearchPaper *papers, int count) {
    char doi[30];
    printf("Enter the DOI of the paper to update: ");
    scanf("%29s", doi);

    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].DOI, doi) == 0) {
            printf("Enter new title: ");
            scanf("%99s", papers[i].title);
            printf("Enter new author: ");
            scanf("%49s", papers[i].author);
            printf("Enter new journal: ");
            scanf("%49s", papers[i].journal);
            printf("Enter new year: ");
            scanf("%d", &papers[i].year);
            return;
        }
    }
    printf("Paper with DOI %s not found.\n", doi);
}

// Function to display all papers published in a specific journal
void displayPapersByJournal(ResearchPaper *papers, int count) {
    char journal[50];
    printf("Enter the journal name: ");
    scanf("%49s", journal);

    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].journal, journal) == 0) {
            printf("Title: %s\nAuthor: %s\nYear: %d\nDOI: %s\n\n", papers[i].title, papers[i].author,
papers[i].year, papers[i].DOI);
        }
    }
}

// Function to find and display the most recent papers published by a specific author
void displayRecentPapersByAuthor(ResearchPaper *papers, int count) {
    char author[50];
    printf("Enter the author name: ");
    scanf("%49s", author);

    int mostRecentYear = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0) {
            if (papers[i].year > mostRecentYear) {
                mostRecentYear = papers[i].year;
            }
        }
    }

    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0 && papers[i].year == mostRecentYear) {
            printf("Title: %s\nJournal: %s\nYear: %d\nDOI: %s\n\n", papers[i].title, papers[i].journal,
papers[i].year, papers[i].DOI);
        }
    }
}

int main() {
    int capacity = 5;
    int count = 0;
    ResearchPaper *papers = malloc(capacity * sizeof(ResearchPaper));

    int choice;
    while (1) {
        printf("1. Add Research Paper\n2. Update Research Paper\n3. Display Papers by Journal\n4.
Display Recent Papers by Author\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
    switch (choice) {
      case 1:
        addResearchPaper(&papers, &count, &capacity);
        break;
      case 2:
        updateResearchPaper(papers, count);
        break;
      case 3:
        displayPapersByJournal(papers, count);
        break;
      case 4:
        displayRecentPapersByAuthor(papers, count);
        break;
      case 5:
        free(papers);
        return 0;
      default:
        printf("Invalid choice. Please try again.\n");
    }
  }
```

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.

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Define the Experiment structure

typedef struct {

   char experimentID[10];

   char researcher[50];

   char startDate[11];

   char endDate[11];

   float results[10];

} Experiment;


// Function to log a new experiment

void logExperiment(Experiment **experiments, int *count, int *capacity) {
   if (*count == *capacity) {
       // Custom array resizing
       *capacity *= 2;
       Experiment *newArray = malloc(*capacity * sizeof(Experiment));
       for (int i = 0; i < *count; i++) {
           newArray[i] = (*experiments)[i];
       }
       free(*experiments);
       *experiments = newArray;
   }

   printf("Enter experiment ID: ");
   scanf("%9s", (*experiments)[*count].experimentID);
   printf("Enter researcher: ");
   scanf("%49s", (*experiments)[*count].researcher);
   printf("Enter start date (YYYY-MM-DD): ");
   scanf("%10s", (*experiments)[*count].startDate);
   printf("Enter end date (YYYY-MM-DD): ");
   scanf("%10s", (*experiments)[*count].endDate);
   printf("Enter up to 10 result readings: ");
   for (int i = 0; i < 10; i++) {
       scanf("%f", &(*experiments)[*count].results[i]);
   }

   (*count)++;
}

// Function to update the result readings of an experiment
void updateResults(Experiment *experiments, int count) {
   char id[10];
   printf("Enter the experiment ID to update: ");
   scanf("%9s", id);


   for (int i = 0; i < count; i++) {
       if (strcmp(experiments[i].experimentID, id) == 0) {
           printf("Enter new result readings: ");
           for (int j = 0; j < 10; j++) {
               scanf("%f", &experiments[i].results[j]);
           }
           return;
       }
   }
   printf("Experiment with ID %s not found.\n", id);
}


// Function to display all experiments conducted by a specific researcher
void displayExperimentsByResearcher(Experiment *experiments, int count) {
   char researcher[50];
   printf("Enter the researcher name: ");
   scanf("%49s", researcher);


   for (int i = 0; i < count; i++) {
       if (strcmp(experiments[i].researcher, researcher) == 0) {
           printf("Experiment ID: %s\nStart Date: %s\nEnd Date: %s\nResults: ",
experiments[i].experimentID, experiments[i].startDate, experiments[i].endDate);
           for (int j = 0; j < 10; j++) {
               printf("%f ", experiments[i].results[j]);
           }
           printf("\n\n");
       }
   }
}
```

```
// Function to calculate and display the average result for a specific experiment
void displayAverageResult(Experiment *experiments, int count) {
    char id[10];
    printf("Enter the experiment ID: ");
    scanf("%9s", id);

    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].experimentID, id) == 0) {
            float sum = 0;
            for (int j = 0; j < 10; j++) {
                sum += experiments[i].results[j];
            }
            float average = sum / 10;
            printf("Average result for experiment %s: %f\n", id, average);
            return;
        }
    }
    printf("Experiment with ID %s not found.\n", id);
}

int main() {
    int capacity = 5;
    int count = 0;
    Experiment *experiments = malloc(capacity * sizeof(Experiment));

    int choice;
    while (1) {
        printf("1. Log New Experiment\n2. Update Results\n3. Display Experiments by Researcher\n4.
Display Average Result for Experiment\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
          case 1:
            logExperiment(&experiments, &count, &capacity);
            break;
          case 2:
            updateResults(experiments, count);
            break;
          case 3:
            displayExperimentsByResearcher(experiments, count);
            break;
          case 4:
            displayAverageResult(experiments, count);
            break;
          case 5:
            free(experiments);
            return 0;
          default:
            printf("Invalid choice. Please try again.\n");
        }
      }
    }
```

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.

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Define the GrantApplication structure

typedef struct {

    char applicationID[10];

    char applicantName[50];

    char projectTitle[100];

    float requestedAmount;

    char status[20];

} GrantApplication;


// Function to add a new grant application

void addGrantApplication(GrantApplication **applications, int *count, int *capacity) {

    if (*count == *capacity) {
        // Custom array resizing
        *capacity *= 2;
        GrantApplication *newArray = malloc(*capacity * sizeof(GrantApplication));
        for (int i = 0; i < *count; i++) {
            newArray[i] = (*applications)[i];
        }
        free(*applications);
        *applications = newArray;
    }

    printf("Enter application ID: ");
    scanf("%9s", (*applications)[*count].applicationID);
    printf("Enter applicant name: ");
    scanf("%49s", (*applications)[*count].applicantName);
    printf("Enter project title: ");
    scanf("%99s", (*applications)[*count].projectTitle);
    printf("Enter requested amount: ");
    scanf("%f", &(*applications)[*count].requestedAmount);
    printf("Enter status (Submitted, Approved, Rejected): ");
    scanf("%19s", (*applications)[*count].status);

    (*count)++;
}

// Function to update the status of an application
void updateApplicationStatus(GrantApplication *applications, int count) {
    char id[10];
    printf("Enter the application ID to update: ");
    scanf("%9s", id);

    for (int i = 0; i < count; i++) {
        if (strcmp(applications[i].applicationID, id) == 0) {
            printf("Enter new status (Submitted, Approved, Rejected): ");
            scanf("%19s", applications[i].status);
            return;
        }
    }
    printf("Application with ID %s not found.\n", id);
}

// Function to display all applications requesting an amount greater than a specified value
void displayApplicationsByAmount(GrantApplication *applications, int count) {
    float amount;
    printf("Enter the amount: ");
    scanf("%f", &amount);

    for (int i = 0; i < count; i++) {
        if (applications[i].requestedAmount > amount) {
            printf("Application ID: %s\nApplicant Name: %s\nProject Title: %s\nRequested Amount:
%f\nStatus: %s\n\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(GrantApplication *applications, int count) {
    for (int i = 0; i < count; i++) {
        if (strcmp(applications[i].status, "Approved") == 0) {
            printf("Application ID: %s\nApplicant Name: %s\nProject Title: %s\nRequested Amount:
%f\nStatus: %s\n\n",
                   applications[i].applicationID, applications[i].applicantName, applications[i].projectTitle,
                applications[i].requestedAmount, applications[i].status);
        }
    }
}

int main() {
    int capacity = 5;
    int count = 0;
    GrantApplication *applications = malloc(capacity * sizeof(GrantApplication));

    int choice;
    while (1) {
        printf("1. Add New Grant Application\n2. Update Application Status\n3. Display Applications by
Amount\n4. Display Approved Applications\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addGrantApplication(&applications, &count, &capacity);
                break;
            case 2:
                updateApplicationStatus(applications, count);
                break;
            case 3:
                displayApplicationsByAmount(applications, count);
                break;
            case 4:
                displayApprovedApplications(applications, count);
                break;
            case 5:
                free(applications);

                return 0;

            default:

                printf("Invalid choice. Please try again.\n");

        }

    }

}
```

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>


// Define the Collaborator structure

typedef struct {

    char collaboratorID[10];

    char name[50];

    char institution[50];

    char expertiseArea[30];

    int numberOfProjects;

} Collaborator;
```

```
// Function to add a new collaborator to the database
void addCollaborator(Collaborator **collaborators, int *count, int *capacity) {
    if (*count == *capacity) {
        // Custom array resizing
        *capacity *= 2;
        Collaborator *newArray = malloc(*capacity * sizeof(Collaborator));
        for (int i = 0; i < *count; i++) {
            newArray[i] = (*collaborators)[i];
        }
        free(*collaborators);
        *collaborators = newArray;
    }

    printf("Enter collaborator ID: ");
    scanf("%9s", (*collaborators)[*count].collaboratorID);
    printf("Enter name: ");
    scanf("%49s", (*collaborators)[*count].name);
    printf("Enter institution: ");
    scanf("%49s", (*collaborators)[*count].institution);
    printf("Enter expertise area: ");
    scanf("%29s", (*collaborators)[*count].expertiseArea);
    printf("Enter number of projects: ");
    scanf("%d", &(*collaborators)[*count].numberOfProjects);

    (*count)++;
}

// Function to update the number of projects a collaborator is involved in
void updateNumberOfProjects(Collaborator *collaborators, int count) {
    char id[10];
    printf("Enter the collaborator ID to update: ");
    scanf("%9s", id);

    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].collaboratorID, id) == 0) {
            printf("Enter new number of projects: ");
            scanf("%d", &collaborators[i].numberOfProjects);
            return;
        }
    }
    printf("Collaborator with ID %s not found.\n", id);
}

// Function to display all collaborators from a specific institution
void displayCollaboratorsByInstitution(Collaborator *collaborators, int count) {
    char institution[50];
    printf("Enter the institution name: ");
    scanf("%49s", institution);

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

// Function to find collaborators with expertise in a given area
void findCollaboratorsByExpertise(Collaborator *collaborators, int count) {
    char expertise[30];
    printf("Enter the expertise area: ");
    scanf("%29s", expertise);

    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].expertiseArea, expertise) == 0) {
            printf("Collaborator ID: %s\nName: %s\nInstitution: %s\nExpertise Area: %s\nNumber of
Projects: %d\n\n",
                   collaborators[i].collaboratorID, collaborators[i].name, collaborators[i].institution,
                   collaborators[i].expertiseArea, collaborators[i].numberOfProjects);
        }
    }
}

int main() {
    int capacity = 5;
    int count = 0;
    Collaborator *collaborators = malloc(capacity * sizeof(Collaborator));

    int choice;
    while (1) {
        printf("1. Add New Collaborator\n2. Update Number of Projects\n3. Display Collaborators by
Institution\n4. Find Collaborators by Expertise\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addCollaborator(&collaborators, &count, &capacity);
                break;
            case 2:
                updateNumberOfProjects(collaborators, count);
                break;
            case 3:
                displayCollaboratorsByInstitution(collaborators, count);
                break;
            case 4:
                findCollaboratorsByExpertise(collaborators, count);
                break;
            case 5:
                free(collaborators);
                return 0;
            default:
                printf("Invalid choice. Please try again.\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>


// Define the ConferenceSubmission structure

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 addConferenceSubmission(ConferenceSubmission **submissions, int *count, int *capacity) {
  if (*count == *capacity) {
    // Custom array resizing
    *capacity *= 2;
    ConferenceSubmission *newArray = malloc(*capacity * sizeof(ConferenceSubmission));
    for (int i = 0; i < *count; i++) {
      newArray[i] = (*submissions)[i];
    }
    free(*submissions);
    *submissions = newArray;
  }

  printf("Enter submission ID: ");
  scanf("%9s", (*submissions)[*count].submissionID);
  printf("Enter author name: ");
  scanf("%49s", (*submissions)[*count].authorName);
  printf("Enter paper title: ");
  scanf("%99s", (*submissions)[*count].paperTitle);
  printf("Enter conference name: ");
  scanf("%49s", (*submissions)[*count].conferenceName);
  printf("Enter submission date (YYYY-MM-DD): ");
  scanf("%10s", (*submissions)[*count].submissionDate);
  printf("Enter status (Pending, Accepted, Rejected): ");
  scanf("%19s", (*submissions)[*count].status);

  (*count)++;
}

// Function to update the status of a submission
```

```
void updateSubmissionStatus(ConferenceSubmission *submissions, int count) {
    char id[10];
    printf("Enter the submission ID to update: ");
    scanf("%9s", id);

    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].submissionID, id) == 0) {
            printf("Enter new status (Pending, Accepted, Rejected): ");
            scanf("%19s", submissions[i].status);
            return;
        }
    }
    printf("Submission with ID %s not found.\n", id);
}

// Function to display all submissions to a specific conference
void displaySubmissionsByConference(ConferenceSubmission *submissions, int count) {
    char conferenceName[50];
    printf("Enter the conference name: ");
    scanf("%49s", conferenceName);

    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].conferenceName, conferenceName) == 0) {
            printf("Submission ID: %s\nAuthor Name: %s\nPaper Title: %s\nSubmission Date: %s\nStatus: 
%s\n\n",
                   submissions[i].submissionID, submissions[i].authorName, submissions[i].paperTitle,
                   submissions[i].submissionDate, submissions[i].status);
        }
    }
}
```

```
// Function to find and display submissions by a specific author
void displaySubmissionsByAuthor(ConferenceSubmission *submissions, int count) {
    char authorName[50];
    printf("Enter the author name: ");
    scanf("%49s", authorName);

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

int main() {
    int capacity = 5;
    int count = 0;
    ConferenceSubmission *submissions = malloc(capacity * sizeof(ConferenceSubmission));

    int choice;
    while (1) {
        printf("1. Add New Conference Submission\n2. Update Submission Status\n3. Display 
Submissions by Conference\n4. Display Submissions by Author\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addConferenceSubmission(&submissions, &count, &capacity);
                break;
            case 2:
                updateSubmissionStatus(submissions, count);
                break;
            case 3:
                displaySubmissionsByConference(submissions, count);
                break;
            case 4:
                displaySubmissionsByAuthor(submissions, count);
                break;
            case 5:
                free(submissions);
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```