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.

Sol: #include <stdio.h>

#include <stdlib.h>

#include <string.h>

```
struct Vehicle {
    char regNumber[15];
    char model[30];
    int year;
    float mileage;
    float fuelEfficiency;
};

int main() {
```

```
int fleetSize = 5, count = 0;
struct Vehicle *fleet = malloc(fleetSize * sizeof(struct Vehicle));


void addVehicle() {
    printf("Enter Registration: ");
    scanf("%s", fleet[count].regNumber);
    printf("Enter Model: ");
    scanf("%s", fleet[count].model);
    printf("Enter Year: ");
    scanf("%d", &fleet[count].year);
    printf("Enter Mileage: ");
    scanf("%f", &fleet[count].mileage);
    printf("Enter Fuel Efficiency: ");
    scanf("%f", &fleet[count].fuelEfficiency);
    count++;
}
void displayVehiclesAfterYear(int year) {
    for (int i = 0; i < count; i++) {
        if (fleet[i].year > year) {
            printf("regNumbe:%s model:%s year:%d float mileage:%.2f float
fuelEfficiency: %.2f\n", fleet[i].regNumber, fleet[i].model, fleet[i].year,
fleet[i].mileage, fleet[i].fuelEfficiency);
```

```
        }
    }
}
void findHighestFuelEfficiency() {
    if (count == 0)
    return;
    int maxIndex = 0;
    for (int i = 1; i < count; i++) {
        if (fleet[i].fuelEfficiency > fleet[maxIndex].fuelEfficiency) {
            maxIndex = i;
        }
    }
    printf("Highest Fuel Efficiency: %s %s %.2f\n", fleet[maxIndex].regNumber,
fleet[maxIndex].model, fleet[maxIndex].fuelEfficiency);
}


int choice;
do {
    printf("\n1. Add Vehicle\n2. Display Vehicles After Year\n3. Find Highest
Fuel Efficiency\n4. Exit\n");
    scanf("%d", &choice);
```

```
        if (choice == 1) addVehicle();
        else if (choice == 2) {
            int year;
            printf("Enter year: ");
            scanf("%d", &year);
            displayVehiclesAfterYear(year);
        }
        else if (choice == 3) findHighestFuelEfficiency();
    } while (choice != 4);

    free(fleet);
    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.

Sol:

```
#include <stdio.h>

#include <string.h>

struct CarRental {

    char carID[10];

    char customerName[50];

    char rentalDate[11];

    char returnDate[11];

    float rentalPricePerDay;

    float totalPrice;

};


int calculateDays(char *rentalDate, char *returnDate) {

    int rentalYear, rentalMonth, rentalDay;

    int returnYear, returnMonth, returnDay;

    sscanf(rentalDate, "%d-%d-%d", &rentalYear, &rentalMonth, &rentalDay);

    sscanf(returnDate, "%d-%d-%d", &returnYear, &returnMonth, &returnDay);

    return (returnYear - rentalYear) * 365 + (returnMonth - rentalMonth) * 30 +
(returnDay - rentalDay);

}
```

```
int main() {
    struct CarRental rental;
    int choice;
    do {
        printf("\n1. Book Car\n2. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            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);
            printf("Enter Return Date (YYYY-MM-DD): ");
            scanf("%s", rental.returnDate);
            printf("Enter Price per Day: ");
            scanf("%f", &rental.rentalPricePerDay);

            int days = calculateDays(rental.rentalDate, rental.returnDate);
```

```
            rental.totalPrice = days * rental.rentalPricePerDay;
            printf("Total Price: %.2f\n", rental.totalPrice);
        }
    } while (choice != 2);

    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.

Sol:

```
#include <stdio.h>

struct SensorData {
    int sensorID;
    char timestamp[20];
    float speed;
```

```
    float latitude;
    float longitude;
};

void logSensorData(struct SensorData *data, int *count) {
    printf("Enter Sensor ID: ");
    scanf("%d", &data[*count].sensorID);

    getchar();

    printf("Enter Timestamp (YYYY-MM-DD HH:MM:SS): ");
    scanf("%19[^\n]", data[*count].timestamp);  // Read until newline, avoid overflow

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

    (*count)++;
```

```
}

void displayData(struct SensorData *data, int count) {
    if (count == 0) {
        printf("No data available.\n");
        return;
    }
    for (int i = 0; i < count; i++) {
        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);
    }
}

void findMaxSpeed(struct SensorData *data, int count) {
    if (count == 0) {
        printf("No data available.\n");
        return;
    }

    float maxSpeed = data[0].speed;
```

```
    for (int i = 1; i < count; i++) {
        if (data[i].speed > maxSpeed) {
            maxSpeed = data[i].speed;
        }
    }
    printf("Maximum Speed: %.2f\n", maxSpeed);
}

void calculateAverageSpeed(struct SensorData *data, int count) {
    if (count == 0) {
        printf("No data available.\n");
        return;
    }
    float totalSpeed = 0;
    for (int i = 0; i < count; i++) {
        totalSpeed += data[i].speed;
    }
    printf("Average Speed: %.2f\n", totalSpeed / count);
}

int main() {
```

```
struct SensorData data[100];
int count = 0;

int choice;
do {
    printf("\nMenu:\n");
    printf("1. Log Sensor Data\n");
    printf("2. Display All Data\n");
    printf("3. Find Maximum Speed\n");
    printf("4. Calculate Average Speed\n");
    printf("5. Exit\n");

    printf("Enter your choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
        logSensorData(data, &count);
    }
    else if (choice == 2) {
        displayData(data, count);
    }
```

```
        else if (choice == 3) {
            findMaxSpeed(data, count);
        }
        else if (choice == 4) {
            calculateAverageSpeed(data, count);
        }
        else if (choice == 5) {
            printf("Exiting\n");
        }
        else {
            printf("Invalid choice. Please try again.\n");
        }

    } while (choice != 5);

    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.

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

#include <stdlib.h>

#include <string.h>



struct EnginePerformance {

    char engineID[10];

    float temperature;

    float rpm;

    float fuelConsumptionRate;

    float oilPressure;

};



void addPerformanceData(struct EnginePerformance *engines, int *count) {

    printf("Enter Engine ID, Temperature, RPM, Fuel Consumption Rate, Oil Pressure: ");

    scanf("%s %f %f %f %f", engines[*count].engineID, &engines[*count].temperature, &engines[*count].rpm,
```

```
        &engines[*count].fuelConsumptionRate, &engines[*count].oilPressure);
    (*count)++;
}

void displayPerformanceData(struct EnginePerformance *engines, int count, char
*engineID) {
    for (int i = 0; i < count; i++) {
        if (strcmp(engines[i].engineID, engineID) == 0) {
            printf("Engine %s: Temperature %.2f, RPM %.2f, Fuel Consumption %.2f,
Oil Pressure %.2f\n",
                    engines[i].engineID, engines[i].temperature, engines[i].rpm,
engines[i].fuelConsumptionRate, engines[i].oilPressure);
        }
    }
}

void calculateAvgTemperatureRPM(struct EnginePerformance *engines, int count,
char *engineID) {
    float totalTemp = 0, totalRPM = 0;
    int matchCount = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(engines[i].engineID, engineID) == 0) {
            totalTemp += engines[i].temperature;
```

```
            totalRPM += engines[i].rpm;
            matchCount++;
        }
    }
    if (matchCount > 0) {
        printf("Average Temperature: %.2f, Average RPM: %.2f\n", totalTemp / matchCount, totalRPM / matchCount);
    } else {
        printf("No data found for engine %s.\n", engineID);
    }
}

void checkAbnormalOilPressure(struct EnginePerformance *engines, int count, float lowThreshold, float highThreshold) {
    for (int i = 0; i < count; i++) {
        if (engines[i].oilPressure < lowThreshold || engines[i].oilPressure > highThreshold) {
            printf("Engine %s has abnormal oil pressure: %.2f\n", engines[i].engineID, engines[i].oilPressure);
        }
    }
}
```

```
int main() {
    struct EnginePerformance engines[100];
    int count = 0, choice;
    char engineID[10];
    float lowThreshold = 10.0, highThreshold = 90.0;  // Example thresholds

    while (1) {
        printf("\n1. Add Performance Data\n2. Display Performance Data\n3. Calculate Avg Temperature & RPM\n4. Check Abnormal Oil Pressure\n5. Exit\nChoice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            addPerformanceData(engines, &count);
        } else if (choice == 2) {
            printf("Enter Engine ID: ");
            scanf("%s", engineID);
            displayPerformanceData(engines, count, engineID);
        } else if (choice == 3) {
            printf("Enter Engine ID: ");
            scanf("%s", engineID);
            calculateAvgTemperatureRPM(engines, count, engineID);
```

```
        } else if (choice == 4) {
            checkAbnormalOilPressure(engines, count, lowThreshold, highThreshold);
        } else {
            break;
        }
    }

    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.

Sol: #include <stdio.h>

#include <stdlib.h>

#include <string.h>

```
struct ServiceRecord {
    char serviceID[10];
    char vehicleID[15];
    char serviceDate[11];
    char description[100];
    float serviceCost;
};

void addServiceRecord(struct ServiceRecord records[], int *count) {
    printf("Enter service ID: ");
    scanf("%s", records[*count].serviceID);
    printf("Enter vehicle ID: ");
    scanf("%s", records[*count].vehicleID);
    printf("Enter service date (DD/MM/YYYY): ");
    scanf("%s", records[*count].serviceDate);
    printf("Enter service description: ");
    getchar();  // to clear the newline left by previous input
    fgets(records[*count].description, 100, stdin);
    printf("Enter service cost: ");
    scanf("%f", &records[*count].serviceCost);
```

```
    (*count)++;
}

void displayServiceRecords(struct ServiceRecord records[], int count, char
vehicleID[]) {
    printf("Service records for vehicle %s:\n", vehicleID);
    for (int i = 0; i < count; i++) {
        if (strcmp(records[i].vehicleID, vehicleID) == 0) {
            printf("%s %s %s %.2f\n", records[i].serviceID, records[i].serviceDate,
records[i].description, records[i].serviceCost);
        }
    }
}

float totalServiceCost(struct ServiceRecord records[], int count, char vehicleID[]) {
    float total = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(records[i].vehicleID, vehicleID) == 0) {
            total += records[i].serviceCost;
        }
    }
    return total;
```

```
}

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

void sortAndDisplayRecords(struct ServiceRecord records[], int count) {
    qsort(records, count, sizeof(struct ServiceRecord), compareDates);
    for (int i = 0; i < count; i++) {
        printf("%s %s %s %.2f\n", records[i].serviceID, records[i].vehicleID,
records[i].serviceDate, records[i].serviceCost);
    }
}

int main() {
    struct ServiceRecord records[100];
    int count = 0;
    int choice;
    char vehicleID[15];

    while (1) {
```

```
        printf("\n1. Add Service Record\n2. Display Service Records\n3. Total Service Cost\n4. Sort and Display Records\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                addServiceRecord(records, &count);
                break;
            case 2:
                printf("Enter vehicle ID to display: ");
                scanf("%s", vehicleID);
                displayServiceRecords(records, count, vehicleID);
                break;
            case 3:
                printf("Enter vehicle ID to calculate total cost: ");
                scanf("%s", vehicleID);
                printf("Total service cost: %.2f\n", totalServiceCost(records, count, vehicleID));
                break;
            case 4:
                sortAndDisplayRecords(records, count);
                break;
            case 5:
```

```
            return 0;
        }
    }
}
```

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.

Sol:

```
#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;
};

int playerCount = 0, capacity = 2;
struct Player* players;

void addPlayer() {
    if (playerCount == capacity) {
        capacity *= 2;
        players = realloc(players, capacity * sizeof(struct Player));
    }
    printf("Enter Name, Age, Team, Matches Played, Total Runs, Total Wickets: ");
    scanf("%s %d %s %d %d %d", players[playerCount].name,
&players[playerCount].age, players[playerCount].team,
&players[playerCount].matchesPlayed, &players[playerCount].totalRuns,
&players[playerCount].totalWickets);
    playerCount++;
}

void updatePlayer() {
    char name[50];
```

```
    printf("Enter player name to update: ");
    scanf("%s", name);
    for (int i = 0; i < playerCount; i++) {
        if (strcmp(players[i].name, name) == 0) {
            printf("Enter Matches Played, Total Runs, Total Wickets: ");
            scanf("%d %d %d", &players[i].matchesPlayed, &players[i].totalRuns,
&players[i].totalWickets);
            return;
        }
    }
    printf("Player not found!\n");
}

void displayByTeam() {
    char team[30];
    printf("Enter team name: ");
    scanf("%s", team);
    for (int i = 0; i < playerCount; i++) {
        if (strcmp(players[i].team, team) == 0) {
            printf("%s %d %s %d %d %d\n", players[i].name, players[i].age,
players[i].team, players[i].matchesPlayed, players[i].totalRuns,
players[i].totalWickets);
        }
```

```
    }
}

void findTopPlayers() {
    int maxRunsIdx = 0, maxWicketsIdx = 0;
    for (int i = 1; i < playerCount; i++) {
        if (players[i].totalRuns > players[maxRunsIdx].totalRuns) maxRunsIdx = i;
        if (players[i].totalWickets > players[maxWicketsIdx].totalWickets)
maxWicketsIdx = i;
    }
    printf("Top Runs: %s %d\nTop Wickets: %s %d\n",
players[maxRunsIdx].name, players[maxRunsIdx].totalRuns,
players[maxWicketsIdx].name, players[maxWicketsIdx].totalWickets);
}

int main() {
    players = malloc(capacity * sizeof(struct Player));
    int choice;
    do {
        printf("1. Add Player\n2. Update Player\n3. Display by Team\n4. Find Top
Players\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        switch (choice) {
```

```
            case 1: addPlayer(); break;
            case 2: updatePlayer(); break;
            case 3: displayByTeam(); break;
            case 4: findTopPlayers(); break;
            case 5: break;
            default: printf("Invalid choice!\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.

Sol:

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

```
#include <string.h>

struct Match {
    char team1[30];
    char team2[30];
    char date[11];
    char venue[50];
};

int matchCount = 0, capacity = 2;
struct Match* matches;

void scheduleMatch() {
    if (matchCount == capacity) {
        capacity *= 2;
        matches = realloc(matches, capacity * sizeof(struct Match));
    }
    printf("Enter Team 1, Team 2, Date (YYYY-MM-DD), Venue: ");
    scanf("%s %s %s %s", matches[matchCount].team1,
matches[matchCount].team2, matches[matchCount].date,
matches[matchCount].venue);
    matchCount++;
```

```
}

void displayMatches() {
    printf("Scheduled Matches:\n");
    for (int i = 0; i < matchCount; i++) {
        printf("%s vs %s on %s at %s\n", matches[i].team1, matches[i].team2,
matches[i].date, matches[i].venue);
    }
}

void searchMatchesByDate() {
    char date[11];
    printf("Enter date (YYYY-MM-DD): ");
    scanf("%s", date);
    for (int i = 0; i < matchCount; i++) {
        if (strcmp(matches[i].date, date) == 0) {
            printf("%s vs %s on %s at %s\n", matches[i].team1, matches[i].team2,
matches[i].date, matches[i].venue);
        }
    }
}
```

```
void cancelMatch() {
    char team1[30], team2[30], date[11];
    printf("Enter Team 1, Team 2, and Date (YYYY-MM-DD): ");
    scanf("%s %s %s", team1, team2, date);
    for (int i = 0; i < matchCount; 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 < matchCount - 1; j++) {
                matches[j] = matches[j + 1];
            }
            matchCount--;
            printf("Match cancelled successfully.\n");
            return;
        }
    }
    printf("Match not found!\n");
}

int main() {
    matches = malloc(capacity * sizeof(struct Match));
    int choice;
    do {
```

```
        printf("1. Schedule Match\n2. Display Matches\n3. Search Matches by Date\n4. Cancel Match\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: scheduleMatch(); break;
            case 2: displayMatches(); break;
            case 3: searchMatchesByDate(); break;
            case 4: cancelMatch(); break;
            case 5: break;
            default: printf("Invalid choice!\n");
        }
    } while (choice != 5);
    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.

Sol:

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

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

int countryCount = 0, capacity = 2;
struct CountryMedalTally* tallies;

void addCountry() {
    if (countryCount == capacity) {
        capacity *= 2;
        tallies = realloc(tallies, capacity * sizeof(struct CountryMedalTally));
    }
    printf("Enter country name, gold, silver, bronze medals: ");
```

```
    scanf("%s %d %d %d", tallies[countryCount].country,
&tallies[countryCount].gold, &tallies[countryCount].silver,
&tallies[countryCount].bronze);

    countryCount++;

}


void updateMedals() {

    char country[30];

    printf("Enter country name to update: ");

    scanf("%s", country);

    for (int i = 0; i < countryCount; i++) {

        if (strcmp(tallies[i].country, country) == 0) {

            printf("Enter new gold, silver, bronze medal counts: ");

            scanf("%d %d %d", &tallies[i].gold, &tallies[i].silver, &tallies[i].bronze);

            return;

        }

    }

    printf("Country not found!\n");

}


void displayAllTallies() {

    printf("Medal Tally:\n");
```

```
    for (int i = 0; i < countryCount; i++) {
        printf("%s - Gold: %d, Silver: %d, Bronze: %d\n", tallies[i].country, tallies[i].gold, tallies[i].silver, tallies[i].bronze);
    }
}

void findTopGoldCountry() {
    if (countryCount == 0) {
        printf("No countries added yet!\n");
        return;
    }
    int maxGoldIdx = 0;
    for (int i = 1; i < countryCount; i++) {
        if (tallies[i].gold > tallies[maxGoldIdx].gold) {
            maxGoldIdx = i;
        }
    }
    printf("Country with most gold medals: %s (%d gold medals)\n", tallies[maxGoldIdx].country, tallies[maxGoldIdx].gold);
}

int main() {
```

```
    tallies = malloc(capacity * sizeof(struct CountryMedalTally));

    int choice;

    do {

        printf("1. Add Country\n2. Update Medals\n3. Display Medal Tally\n4. Find
Top Gold Medal Country\n5. Exit\nChoice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1: addCountry(); break;

            case 2: updateMedals(); break;

            case 3: displayAllTallies(); break;

            case 4: findTopGoldCountry(); break;

            case 5: break;

            default: printf("Invalid choice!\n");

        }

    } while (choice != 5);



    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.

Sol: #include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Athlete {

    char athleteID[10];

    char name[50];

    char sport[30];

    float personalBest;

    float lastPerformance;

};

int athleteCount = 0, capacity = 2;

struct Athlete* athletes;

```
void addAthlete() {
    if (athleteCount == capacity) {
        capacity *= 2;
        athletes = realloc(athletes, capacity * sizeof(struct Athlete));
    }
    printf("Enter Athlete ID, Name, Sport, Personal Best, Last Performance: ");
    scanf("%s %s %s %f %f", athletes[athleteCount].athleteID,
athletes[athleteCount].name, athletes[athleteCount].sport,
&athletes[athleteCount].personalBest, &athletes[athleteCount].lastPerformance);
    athleteCount++;
}


void updateLastPerformance() {
    char athleteID[10];
    printf("Enter Athlete ID to update: ");
    scanf("%s", athleteID);
    for (int i = 0; i < athleteCount; i++) {
        if (strcmp(athletes[i].athleteID, athleteID) == 0) {
            printf("Enter new 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");
}

void displayAthletesBySport() {
    char sport[30];
    printf("Enter sport: ");
    scanf("%s", sport);
    for (int i = 0; i < athleteCount; i++) {
        if (strcmp(athletes[i].sport, sport) == 0) {
            printf("ID: %s, Name: %s, Personal Best: %.2f, Last Performance: %.2f\n",
athletes[i].athleteID, athletes[i].name, athletes[i].personalBest,
athletes[i].lastPerformance);
        }
    }
}

void displayNewPersonalBests() {
    printf("Athletes with new personal bests:\n");
```

```
    for (int i = 0; i < athleteCount; i++) {
        if (athletes[i].lastPerformance == athletes[i].personalBest) {
            printf("ID: %s, Name: %s, Sport: %s, Personal Best: %.2f\n",
athletes[i].athleteID, athletes[i].name, athletes[i].sport, athletes[i].personalBest);
        }
    }
}

int main() {
    athletes = malloc(capacity * sizeof(struct Athlete));
    int choice;
    do {
        printf("1. Add Athlete\n2. Update Last Performance\n3. Display Athletes by
Sport\n4. Display New Personal Bests\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: addAthlete(); break;
            case 2: updateLastPerformance(); break;
            case 3: displayAthletesBySport(); break;
            case 4: displayNewPersonalBests(); break;
            case 5: break;
            default: printf("Invalid choice!\n");
```

```
        }
    } while (choice != 5);
    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.

Sol:

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

struct Equipment {
    char equipmentID[10];
    char name[30];
    char category[20];
```

```
    int quantity;
    float pricePerUnit;
};

int equipmentCount = 0, capacity = 2;
struct Equipment* inventory;

void addEquipment() {
    if (equipmentCount == capacity) {
        capacity *= 2;
        inventory = realloc(inventory, capacity * sizeof(struct Equipment));
    }
    printf("Enter Equipment ID, Name, Category, Quantity, Price per Unit: ");
    scanf("%s %s %s %d %f", inventory[equipmentCount].equipmentID,
inventory[equipmentCount].name, inventory[equipmentCount].category,
&inventory[equipmentCount].quantity,
&inventory[equipmentCount].pricePerUnit);
    equipmentCount++;
}

void updateQuantity() {
    char equipmentID[10];
    printf("Enter Equipment ID to update quantity: ");
```

```
    scanf("%s", equipmentID);
    for (int i = 0; i < equipmentCount; i++) {
        if (strcmp(inventory[i].equipmentID, equipmentID) == 0) {
            printf("Enter new quantity: ");
            scanf("%d", &inventory[i].quantity);
            printf("Quantity updated successfully.\n");
            return;
        }
    }
    printf("Equipment not found!\n");
}

void displayEquipmentByCategory() {
    char category[20];
    printf("Enter category: ");
    scanf("%s", category);
    printf("Equipment in category '%s':\n", category);
    for (int i = 0; i < equipmentCount; i++) {
        if (strcmp(inventory[i].category, category) == 0) {
            printf("ID: %s, Name: %s, Quantity: %d, Price per Unit: %.2f\n",
inventory[i].equipmentID, inventory[i].name, inventory[i].quantity,
inventory[i].pricePerUnit);
```

```
        }
    }
}

void calculateTotalValue() {
    float totalValue = 0;
    for (int i = 0; i < equipmentCount; i++) {
        totalValue += inventory[i].quantity * inventory[i].pricePerUnit;
    }
    printf("Total value of inventory: %.2f\n", totalValue);
}

int main() {
    inventory = malloc(capacity * sizeof(struct Equipment));
    int choice;
    do {
        printf("\n1. Add Equipment\n2. Update Quantity\n3. Display Equipment by Category\n4. Calculate Total Value\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: addEquipment(); break;
            case 2: updateQuantity(); break;
```

```
            case 3: displayEquipmentByCategory(); break;
            case 4: calculateTotalValue(); break;
            case 5: break;
            default: printf("Invalid choice!\n");
        }
    } while (choice != 5);
    free(inventory);
    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.

Sol:

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

```
struct ResearchPaper {
    char title[100];
    char author[50];
    char journal[50];
    int year;
    char DOI[30];
};

void addResearchPaper(struct ResearchPaper **papers, int *count) {
    *papers = realloc(*papers, (*count + 1) * sizeof(struct ResearchPaper));
    printf("Enter title, author, journal, year, DOI: ");
    scanf("%s %s %s %d %s", (*papers)[*count].title, (*papers)[*count].author,
(*papers)[*count].journal, &(*papers)[*count].year, (*papers)[*count].DOI);
    (*count)++;
}

void updateResearchPaper(struct ResearchPaper *papers, int count, char *DOI) {
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].DOI, DOI) == 0) {
            printf("Enter new title, author, journal, year, DOI: ");
```

```
            scanf("%s %s %s %d %s", papers[i].title, papers[i].author,
papers[i].journal, &papers[i].year, papers[i].DOI);

            printf("Paper details updated.\n");

            return;

        }

    }

    printf("Paper with DOI %s not found.\n", DOI);

}



void displayPapersByJournal(struct ResearchPaper *papers, int count, char
*journal) {

    for (int i = 0; i < count; i++) {

        if (strcmp(papers[i].journal, journal) == 0) {

            printf("%s by %s (%d) DOI: %s\n", papers[i].title, papers[i].author,
papers[i].year, papers[i].DOI);

        }

    }

}



void displayRecentPapersByAuthor(struct ResearchPaper *papers, int count, char
*author) {

    int maxYear = -1;

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

```
        if (strcmp(papers[i].author, author) == 0 && papers[i].year > maxYear) {
            maxYear = papers[i].year;
        }
    }
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0 && papers[i].year == maxYear) {
            printf("%s by %s (%d) DOI: %s\n", papers[i].title, papers[i].author,
papers[i].year, papers[i].DOI);
        }
    }
}

int main() {
    struct ResearchPaper *papers = NULL;
    int count = 0;
    int choice;
    char DOI[30], journal[50], author[50];

    while (1) {
        printf("\n1. Add Paper\n2. Update Paper\n3. Display by Journal\n4. Display
Recent by Author\n5. Exit\nChoice: ");
        scanf("%d", &choice);
```

```
    if (choice == 1) {
        addResearchPaper(&papers, &count);
    } else if (choice == 2) {
        printf("Enter DOI of paper to update: ");
        scanf("%s", DOI);
        updateResearchPaper(papers, count, DOI);
    } else if (choice == 3) {
        printf("Enter journal name: ");
        scanf("%s", journal);
        displayPapersByJournal(papers, count, journal);
    } else if (choice == 4) {
        printf("Enter author name: ");
        scanf("%s", author);
        displayRecentPapersByAuthor(papers, count, author);
    } else if (choice == 5) {
        free(papers);
        break;
    }
}
```

```
    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.

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

#include <stdlib.h>

#include <string.h>



struct Experiment {

    char experimentID[10];

    char researcher[50];

    char startDate[11];

    char endDate[11];

    float results[10];
```

```
};


void logExperiment(struct Experiment **experiments, int *count) {
    *experiments = realloc(*experiments, (*count + 1) * sizeof(struct Experiment));
    printf("Enter experiment ID, researcher name, start date (YYYY-MM-DD), end
date (YYYY-MM-DD): ");
    scanf("%s %s %s %s", (*experiments)[*count].experimentID,
(*experiments)[*count].researcher,
                  (*experiments)[*count].startDate,
(*experiments)[*count].endDate);
    printf("Enter up to 10 results (enter -1 to stop): ");
    for (int i = 0; i < 10; i++) {
        scanf("%f", &(*experiments)[*count].results[i]);
        if ((*experiments)[*count].results[i] == -1) break;
    }
    (*count)++;
}


void updateResults(struct Experiment *experiments, int count, char
*experimentID) {
    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].experimentID, experimentID) == 0) {
            printf("Enter new results for experiment ID %s (enter -1 to stop):\n",
experimentID);
```

```
            for (int j = 0; j < 10; j++) {
                scanf("%f", &experiments[i].results[j]);
                if (experiments[i].results[j] == -1) break;
            }
            printf("Results updated for experiment ID %s.\n", experimentID);
            return;
        }
    }
    printf("Experiment ID not found.\n");
}

void displayByResearcher(struct Experiment *experiments, int count, char
*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);
        }
    }
}

void calculateAverage(struct Experiment *experiments, int count, char
*experimentID) {
```

```
    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].experimentID, experimentID) == 0) {
            float sum = 0;
            int validResults = 0;
            for (int j = 0; j < 10; j++) {
                if (experiments[i].results[j] == -1) break;
                sum += experiments[i].results[j];
                validResults++;
            }
            printf("Average result: %.2f\n", validResults > 0 ? sum / validResults : 0);
            return;
        }
    }
    printf("Experiment not found.\n");
}

int main() {
    struct Experiment *experiments = NULL;
    int count = 0, choice;
    char researcher[50], experimentID[10];
```

```
    while (1) {
        printf("\n1. Log Experiment\n2. Update Experiment Results\n3. Display by Researcher\n4. Calculate Average\n5. Exit\nChoice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            logExperiment(&experiments, &count);
        } else if (choice == 2) {
            printf("Enter experiment ID to update results: ");
            scanf("%s", experimentID);
            updateResults(experiments, count, experimentID);
        } else if (choice == 3) {
            printf("Enter researcher name: ");
            scanf("%s", researcher);
            displayByResearcher(experiments, count, researcher);
        } else if (choice == 4) {
            printf("Enter experiment ID to calculate average: ");
            scanf("%s", experimentID);
            calculateAverage(experiments, count, experimentID);
        } else if (choice == 5) {
            free(experiments);
            break;
```

```
        }
    }


    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.

Sol:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct GrantApplication {

    char applicationID[10], applicantName[50], projectTitle[100], status[20];

    float requestedAmount;
```

```
};

void addApplication(struct GrantApplication **apps, int *count) {
    *apps = realloc(*apps, (*count + 1) * sizeof(struct GrantApplication));
    printf("Enter ID, Name, Project, Amount, Status: ");
    scanf("%s %s %[^\n]s %f %s", (*apps)[*count].applicationID,
(*apps)[*count].applicantName,
           (*apps)[*count].projectTitle, &(*apps)[*count].requestedAmount,
(*apps)[*count].status);
    (*count)++;
}

void updateStatus(struct GrantApplication *apps, int count, char *id, char *status)
{
    for (int i = 0; i < count; i++) {
        if (strcmp(apps[i].applicationID, id) == 0) {
            strcpy(apps[i].status, status);
            printf("Updated application %s to status %s.\n", id, status);
            return;
        }
    }
    printf("Application %s not found.\n", id);
}
```

```
void displayByAmount(struct GrantApplication *apps, int count, float amount) {
    for (int i = 0; i < count; i++) {
        if (apps[i].requestedAmount > amount) {
            printf("ID: %s, Name: %s, Project: %s, Amount: %.2f, Status: %s\n",
                   apps[i].applicationID, apps[i].applicantName, apps[i].projectTitle,
                   apps[i].requestedAmount, apps[i].status);
        }
    }
}

void displayApproved(struct GrantApplication *apps, int count) {
    for (int i = 0; i < count; i++) {
        if (strcmp(apps[i].status, "Approved") == 0) {
            printf("ID: %s, Name: %s, Project: %s, Amount: %.2f, Status: %s\n",
                   apps[i].applicationID, apps[i].applicantName, apps[i].projectTitle,
                   apps[i].requestedAmount, apps[i].status);
        }
    }
}
```

```
int main() {
    struct GrantApplication *applications = NULL;
    int count = 0, choice;
    char id[10], status[20];
    float amount;

    while (1) {
        printf("\n1. Add Application\n2. Update Status\n3. Display by Amount\n4. Display Approved\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        if (choice == 1) addApplication(&applications, &count);
        else if (choice == 2) {
            printf("Enter ID and new status: ");
            scanf("%s %s", id, status);
            updateStatus(applications, count, id, status);
        }
        else if (choice == 3) {
            printf("Enter minimum requested amount: ");
            scanf("%f", &amount);
            displayByAmount(applications, count, amount);
        }
        else if (choice == 4) displayApproved(applications, count);
```

```
        else break;
    }

    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.

Sol:

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

struct Collaborator {
    char collaboratorID[10], name[50], institution[50], expertiseArea[30];
```

```
    int numberOfProjects;
};

void addCollaborator(struct Collaborator **collaborators, int *count) {
    *collaborators = realloc(*collaborators, (*count + 1) * sizeof(struct
Collaborator));
    printf("Enter ID, Name, Institution, Expertise, Projects: ");
    scanf("%s %s %s %s %d", (*collaborators)[*count].collaboratorID,
(*collaborators)[*count].name,
          (*collaborators)[*count].institution, (*collaborators)[*count].expertiseArea,
&(*collaborators)[*count].numberOfProjects);
    (*count)++;
}

void updateProjects(struct Collaborator *collaborators, int count, char *id, int
newProjects) {
    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].collaboratorID, id) == 0) {
            collaborators[i].numberOfProjects = newProjects;
            printf("Updated %s with %d projects.\n", id, newProjects);
            return;
        }
    }
```

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

void displayByInstitution(struct Collaborator *collaborators, int count, char
*institution) {
    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].institution, institution) == 0) {
            printf("%s %s %s %s %d\n", collaborators[i].collaboratorID,
collaborators[i].name, collaborators[i].institution,
                collaborators[i].expertiseArea, collaborators[i].numberOfProjects);
        }
    }
}

void findByExpertise(struct Collaborator *collaborators, int count, char *expertise)
{
    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].expertiseArea, expertise) == 0) {
            printf("%s %s %s %d\n", collaborators[i].collaboratorID,
collaborators[i].name, collaborators[i].institution,
                collaborators[i].numberOfProjects);
        }
    }
```

```
}

int main() {
    struct Collaborator *collaborators = NULL;
    int count = 0, choice;
    char id[10], institution[50], expertise[30];
    int newProjects;

    while (1) {
        printf("\n1. Add Collaborator\n2. Update Projects\n3. Display by Institution\n4. Find by Expertise\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        if (choice == 1) addCollaborator(&collaborators, &count);
        else if (choice == 2) {
            printf("Enter ID and new number of projects: ");
            scanf("%s %d", id, &newProjects);
            updateProjects(collaborators, count, id, newProjects);
        }
        else if (choice == 3) {
            printf("Enter institution: ");
            scanf("%s", institution);
            displayByInstitution(collaborators, count, institution);
```

```
        }
        else if (choice == 4) {
            printf("Enter expertise: ");
            scanf("%s", expertise);
            findByExpertise(collaborators, count, expertise);
        }
        else break;
    }

    free(collaborators);
    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

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

#include <stdlib.h>

#include <string.h>


struct ConferenceSubmission {

   char submissionID[10], authorName[50], paperTitle[100], conferenceName[50], submissionDate[11], status[20];

};


void addSubmission(struct ConferenceSubmission **submissions, int *count) {

   *submissions = realloc(*submissions, (*count + 1) * sizeof(struct ConferenceSubmission));

   printf("Enter ID, Author, Paper Title, Conference, Date, Status: ");

   scanf("%s %s %s %s %s %s", (*submissions)[*count].submissionID, (*submissions)[*count].authorName,

         (*submissions)[*count].paperTitle, (*submissions)[*count].conferenceName, (*submissions)[*count].submissionDate,

         (*submissions)[*count].status);

   (*count)++;

}


void updateStatus(struct ConferenceSubmission *submissions, int count, char *id, char *newStatus) {

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

```
        if (strcmp(submissions[i].submissionID, id) == 0) {
            strcpy(submissions[i].status, newStatus);
            printf("Updated submission %s with status %s.\n", id, newStatus);
            return;
        }
    }
    printf("Submission %s not found.\n", id);
}

void displayByConference(struct ConferenceSubmission *submissions, int count,
char *conference) {
    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].conferenceName, conference) == 0) {
            printf("%s %s %s %s %s\n", submissions[i].submissionID,
submissions[i].authorName, submissions[i].paperTitle,
                    submissions[i].submissionDate, submissions[i].status);
        }
    }
}

void displayByAuthor(struct ConferenceSubmission *submissions, int count, char
*author) {
    for (int i = 0; i < count; i++) {
```

```
        if (strcmp(submissions[i].authorName, author) == 0) {
            printf("%s %s %s %s %s\n", submissions[i].submissionID,
submissions[i].authorName, submissions[i].paperTitle,
                   submissions[i].conferenceName, submissions[i].status);
        }
    }
}

int main() {
    struct ConferenceSubmission *submissions = NULL;
    int count = 0, choice;
    char id[10], conference[50], author[50], newStatus[20];

    while (1) {
        printf("\n1. Add Submission\n2. Update Status\n3. Display by Conference\n4.
Display by Author\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        if (choice == 1) addSubmission(&submissions, &count);
        else if (choice == 2) {
            printf("Enter ID and new status: ");
            scanf("%s %s", id, newStatus);
            updateStatus(submissions, count, id, newStatus);
```

```
        }
        else if (choice == 3) {
            printf("Enter conference name: ");
            scanf("%s", conference);
            displayByConference(submissions, count, conference);
        }
        else if (choice == 4) {
            printf("Enter author name: ");
            scanf("%s", author);
            displayByAuthor(submissions, count, author);
        }
        else break;
    }

    free(submissions);
    return 0;
}
```