Day 13 programs

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;
};
```

```
// Function to add a new vehicle to the fleet
void addVehicle(struct Vehicle* fleet, int* count) {
    printf("Enter Vehicle Registration Number: ");
    scanf("%s", fleet[*count].registrationNumber);
    printf("Enter Vehicle Model: ");
    scanf("%s", fleet[*count].model);
    printf("Enter Year of Manufacture: ");
    scanf("%d", &fleet[*count].yearOfManufacture);
    printf("Enter Mileage: ");
    scanf("%f", &fleet[*count].mileage);
    printf("Enter Fuel Efficiency: ");
    scanf("%f", &fleet[*count].fuelEfficiency);

    (*count)++;
}

// Function to update the mileage and fuel efficiency for a vehicle
void updateVehicle(struct Vehicle* fleet, int count, const char* 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);
            printf("Vehicle updated successfully!\n");
            return;
        }
    }
```

```
    printf("Vehicle with registration number %s not found.\n", registrationNumber);
}

// Function to display all vehicles manufactured after a certain year
void displayVehiclesManufacturedAfterYear(struct Vehicle* fleet, int count, int year) {
    printf("Vehicles manufactured after year %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 findVehicleWithHighestFuelEfficiency(struct Vehicle* fleet, int count) {
    if (count == 0) {
        printf("No vehicles in the fleet.\n");
        return;
    }

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

```
    }

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

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

    struct Vehicle* fleet = (struct Vehicle*)malloc(capacity * sizeof(struct Vehicle));

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

    int choice;

    while (1) {
        // Display menu
        printf("\nVehicle Fleet Management System\n");
        printf("1. Add New Vehicle\n");
        printf("2. Update Vehicle Mileage and Fuel Efficiency\n");
        printf("3. Display Vehicles Manufactured After a Specific 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);
}
else if (choice == 2) {
    char regNo[15];
    printf("Enter Registration Number of the vehicle to update: ");
    scanf("%s", regNo);
    updateVehicle(fleet, count, regNo);
}
else if (choice == 3) {
    int year;
    printf("Enter the year to display vehicles manufactured after: ");
    scanf("%d", &year);
    displayVehiclesManufacturedAfterYear(fleet, count, year);
}
else if (choice == 4) {
    findVehicleWithHighestFuelEfficiency(fleet, count);
}
else if (choice == 5) {
    free(fleet);
    printf("Exiting the system...\n");
    break;
```

```
        }
        else {
            printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

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];
    char returnDate[11];
    float rentalPricePerDay;
};

int calculateRentalDays(const char* rentalDate, const char* returnDate) {
    int rYear, rMonth, rDay;
    int rrYear, rrMonth, rrDay;

    sscanf(rentalDate, "%d-%d-%d", &rYear, &rMonth, &rDay);

    sscanf(returnDate, "%d-%d-%d", &rrYear, &rrMonth, &rrDay);

    //To Convert both dates
    int rentalDays = rYear * 365 + rMonth * 30 + rDay;
    int returnDays = rrYear * 365 + rrMonth * 30 + rrDay;

    return returnDays - rentalDays;
}

void bookCar(struct CarRental* rentals, int* rentalCount) {
    printf("Enter Car Rental Details:\n");

    printf("Car ID: ");
    scanf("%s", rentals[*rentalCount].carID);
```

```
    printf("Customer Name: ");
    scanf(" %[^\n]%*c", rentals[*rentalCount].customerName);  // Read full name with spaces

    printf("Rental Date (YYYY-MM-DD): ");
    scanf("%s", rentals[*rentalCount].rentalDate);

    printf("Return Date (YYYY-MM-DD): ");
    scanf("%s", rentals[*rentalCount].returnDate);

    printf("Rental Price Per Day: ");
    scanf("%f", &rentals[*rentalCount].rentalPricePerDay);

    (*rentalCount)++;
}

void calculateTotalPrice(struct CarRental* rentals, int rentalCount) {
    printf("\nCar Rental Details with Total Price:\n");
    for (int i = 0; i < rentalCount; i++) {
        int days = calculateRentalDays(rentals[i].rentalDate, rentals[i].returnDate);
        float totalPrice = days * rentals[i].rentalPricePerDay;
        printf("Car ID: %s, Customer: %s, Rental Days: %d, Total Price: %.2f\n",
               rentals[i].carID, rentals[i].customerName, days, totalPrice);
    }
}

void displayAllRentals(struct CarRental* rentals, int rentalCount) {
    printf("\nAll Current Rentals:\n");
    for (int i = 0; i < rentalCount; i++) {
```

```
        printf("Car ID: %s, Customer: %s, Rental Date: %s, Return Date: %s\n",
               rentals[i].carID, rentals[i].customerName, rentals[i].rentalDate,
rentals[i].returnDate);
    }
}

void searchByCustomerName(struct CarRental* rentals, int rentalCount, const char*
customerName) {
    int found = 0;
    printf("\nSearch Results for Customer: %s\n", customerName);
    for (int i = 0; i < rentalCount; i++) {
        if (strcmp(rentals[i].customerName, customerName) == 0) {
            printf("Car ID: %s, Rental Date: %s, Return Date: %s, Price per day: %.2f\n",
                   rentals[i].carID, rentals[i].rentalDate, rentals[i].returnDate,
rentals[i].rentalPricePerDay);
            found = 1;
        }
    }
    if (!found) {
        printf("No rentals found for customer %s.\n", customerName);
    }
}

int main() {
    struct CarRental* rentals = malloc(10 * sizeof(struct CarRental));  // Initial allocation for 10
rentals
    int rentalCount = 0;
    int rentalCapacity = 10;
```

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

    // Booking some cars
    bookCar(rentals, &rentalCount);
    bookCar(rentals, &rentalCount);

    // Calculate and display total rental price
    calculateTotalPrice(rentals, rentalCount);

    // Display all current rentals
    displayAllRentals(rentals, rentalCount);

    // Search rentals by customer name
    char searchName[50];
    printf("Enter customer name to search: ");
    scanf(" %[^\n]%*c", searchName);
    searchByCustomerName(rentals, rentalCount, searchName);

    free(rentals);

    return 0;
}
```

3: Autonomous Vehicle Sensor Data Logger

Requirements:

Create a structure SensorData with fields:

int sensorID

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

float speed

float latitude

float longitude

Functions to:

Log new sensor data.

Display sensor data for a specific time range.

Find the maximum speed recorded.

Calculate the average speed over a specific time period.

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

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

// Structure to store sensor data
struct SensorData {
    int sensorID;
    char timestamp[20];
    float speed;
    float latitude;
    float longitude;
};

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

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

    (*count)++;
}

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

// Function to find the maximum speed recorded
```

```
float findMaxSpeed(struct SensorData* data, int count) {
    float maxSpeed = data[0].speed;
    for (int i = 1; i < count; i++) {
        if (data[i].speed > maxSpeed) {
            maxSpeed = data[i].speed;
        }
    }
    return maxSpeed;
}

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

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

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

int main() {
    int count = 0;
```

```
    int capacity = 2;

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

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

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



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

    displayDataInRange(data, count, startTime, endTime);

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

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

    free(data);

    return 0;
}
```

4: Engine Performance Monitoring System

Requirements:

Define a structure EnginePerformance with members:

char engineID[10]

float temperature

float rpm

float fuelConsumptionRate

float oilPressure

Functions to:

Add performance data for a specific engine.

Display all performance data for a specific engine ID.

Calculate the average temperature and RPM for a specific engine.

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

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

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

// Function to add performance data
void addPerformanceData(struct EnginePerformance* data, int* count) {
    printf("Enter Engine ID: ");
    scanf("%s", data[*count].engineID);
    printf("Enter Temperature: ");
    scanf("%f", &data[*count].temperature);
    printf("Enter RPM: ");
    scanf("%f", &data[*count].rpm);
    printf("Enter Fuel Consumption Rate: ");
    scanf("%f", &data[*count].fuelConsumptionRate);
    printf("Enter Oil Pressure: ");
    scanf("%f", &data[*count].oilPressure);
    (*count)++;
}

// Function to display performance data for a specific engine ID
void displayPerformanceData(struct EnginePerformance* data, int count, const char*
engineID) {
    printf("Performance Data for Engine ID: %s\n", engineID);
    for (int i = 0; i < count; i++) {
```

```
        if (strcmp(data[i].engineID, engineID) == 0) {
            printf("Temperature: %.2f, RPM: %.2f, Fuel Consumption Rate: %.2f, Oil Pressure: %.2f\n",
                   data[i].temperature, data[i].rpm, data[i].fuelConsumptionRate, data[i].oilPressure);
        }
    }
}

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

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

    if (validCount > 0) {
        printf("Average Temperature: %.2f\n", totalTemp / validCount);
        printf("Average RPM: %.2f\n", totalRPM / validCount);
    } else {
        printf("No data found for Engine ID: %s\n", engineID);
    }
}
```

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

int main() {
    int count = 0;
    int maxRecords = 10;
    struct EnginePerformance* data = (struct EnginePerformance*)malloc(maxRecords * sizeof(struct EnginePerformance));

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

    // Add performance data for 3 engines
    addPerformanceData(data, &count);
    addPerformanceData(data, &count);
    addPerformanceData(data, &count);

    // Display performance data for a specific engine
```

```
    char engineID[10];
    printf("Enter Engine ID to display performance data: ");
    scanf("%s", engineID);
    displayPerformanceData(data, count, engineID);

    // Calculate average temperature and RPM for a specific engine
    printf("Enter Engine ID to calculate average temperature and RPM: ");
    scanf("%s", engineID);
    calculateAverage(data, count, engineID);

    // Identify engines with abnormal oil pressure
    float lowThreshold = 20.0, highThreshold = 80.0;
    identifyAbnormalOilPressure(data, count, lowThreshold, highThreshold);

    free(data);

    return 0;
}
```

5: Vehicle Service History Tracker

Requirements:

Create a structure ServiceRecord with the following:

char serviceID[10]

char vehicleID[15]

char serviceDate[11]

char description[100]

float serviceCost

Functions to:

Add a new service record for a vehicle.

Display all service records for a given vehicle ID.

Calculate the total cost of services for a vehicle.

Sort and display service records by service date.

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

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

// Function to add a new service record
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 (YYYY-MM-DD): ");
    scanf("%s", records[*count].serviceDate);

    printf("Enter Description (no spaces): ");
```

```
    scanf("%s", records[*count].description);

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

    (*count)++;
}

// Function to display 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, Date: %s, Description: %s, 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 calculateTotalCost(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 sort service records by service date
void sortServiceRecordsByDate(struct ServiceRecord* records, int count) {
    for (int i = 0; i < count - 1; i++) {
        for (int j = i + 1; j < count; j++) {
            if (strcmp(records[i].serviceDate, records[j].serviceDate) > 0) {
                struct ServiceRecord temp = records[i];
                records[i] = records[j];
                records[j] = temp;
            }
        }
    }
}

int main() {
    int count = 0;
    int maxRecords = 10;
    struct ServiceRecord* records = (struct ServiceRecord*)malloc(maxRecords * sizeof(struct
ServiceRecord));

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

    // Add service records
```

```
    addServiceRecord(records, &count);
    addServiceRecord(records, &count);

    // Display service records for a specific vehicle ID
    char vehicleID[15];
    printf("Enter Vehicle ID to display records: ");
    scanf("%s", vehicleID);
    displayServiceRecords(records, count, vehicleID);

    // Calculate the total cost of services for a specific vehicle
    printf("Enter Vehicle ID to calculate total service cost: ");
    scanf("%s", vehicleID);
    float totalCost = calculateTotalCost(records, count, vehicleID);
    printf("Total Service Cost for Vehicle ID %s: %.2f\n", vehicleID, totalCost);

    // Sort and display service records by date
    sortServiceRecordsByDate(records, count);
    printf("Sorted Service Records by Date:\n");
    for (int i = 0; i < count; i++) {
        printf("Service ID: %s, Vehicle ID: %s, Date: %s, Description: %s, Cost: %.2f\n",
            records[i].serviceID, records[i].vehicleID, records[i].serviceDate,
records[i].description, records[i].serviceCost);
    }

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

set 2 programs

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>

// Structure to store player data
struct Player {
    char name[50];
    int age;
    char team[30];
    int matchesPlayed;
    int totalRuns;
```

```
    int totalWickets;
};

// Function to add a new player
void addPlayer(struct Player* players, int* count) {
    printf("Enter Player Name: ");
    scanf("%s", players[*count].name);
    printf("Enter Player Age: ");
    scanf("%d", &players[*count].age);
    printf("Enter Team Name: ");
    scanf("%s", players[*count].team);
    printf("Enter Matches Played: ");
    scanf("%d", &players[*count].matchesPlayed);
    printf("Enter Total Runs: ");
    scanf("%d", &players[*count].totalRuns);
    printf("Enter Total Wickets: ");
    scanf("%d", &players[*count].totalWickets);

    (*count)++;
}

// Function to update a player's statistics
void updatePlayerStats(struct Player* players, int count, const char* playerName) {
    for (int i = 0; i < count; i++) {
        if (strcmp(players[i].name, playerName) == 0) {
            printf("Enter Matches Played in this match: ");
            int matches;
            scanf("%d", &matches);
```

```
            players[i].matchesPlayed += matches;

            printf("Enter Runs Scored in this match: ");
            int runs;
            scanf("%d", &runs);
            players[i].totalRuns += runs;

            printf("Enter Wickets Taken in this match: ");
            int wickets;
            scanf("%d", &wickets);
            players[i].totalWickets += wickets;

            printf("Statistics updated for player: %s\n", players[i].name);
            return;
        }
    }
    printf("Player not found!\n");
}

// Function to display players from a specific team
void displayPlayersFromTeam(struct Player* players, int count, const char* teamName) {
    printf("Players from team: %s\n", teamName);
    for (int i = 0; i < count; i++) {
        if (strcmp(players[i].team, teamName) == 0) {
            printf("Name: %s, Age: %d, Matches: %d, Runs: %d, 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 most wickets
void findTopPlayers(struct Player* players, int count) {
    int maxRuns = 0, maxWickets = 0;
    int runsIndex = -1, wicketsIndex = -1;

    for (int i = 0; i < count; i++) {
        if (players[i].totalRuns > maxRuns) {
            maxRuns = players[i].totalRuns;
            runsIndex = i;
        }
        if (players[i].totalWickets > maxWickets) {
            maxWickets = players[i].totalWickets;
            wicketsIndex = i;
        }
    }

    if (runsIndex != -1) {
        printf("Player with highest runs: %s (Runs: %d)\n",
               players[runsIndex].name, players[runsIndex].totalRuns);
    }
    if (wicketsIndex != -1) {
        printf("Player with most wickets: %s (Wickets: %d)\n",
               players[wicketsIndex].name, players[wicketsIndex].totalWickets);
    }
}
```

```
int main() {
    int count = 0;
    int maxPlayers = 10;
    struct Player* players = (struct Player*)malloc(maxPlayers * sizeof(struct Player));

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

    // Adding sample players
    printf("Add a new player:\n");
    addPlayer(players, &count);

    printf("Add another player:\n");
    addPlayer(players, &count);

    // Update a player's statistics
    char playerName[50];
    printf("Enter Player Name to update statistics: ");
    scanf("%s", playerName);
    updatePlayerStats(players, count, playerName);

    // Display players from a specific team
    char teamName[30];
    printf("Enter Team Name to display players: ");
    scanf("%s", teamName);
```

```
    displayPlayersFromTeam(players, count, teamName);

    findTopPlayers(players, count);

    free(players);

    return 0;
}
```

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>

// Structure to store match details
```

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

// Function to schedule a new match
void scheduleMatch(struct Match* matches, int* count) {
    printf("Enter Team 1: ");
    scanf("%s", matches[*count].team1);
    printf("Enter Team 2: ");
    scanf("%s", matches[*count].team2);
    printf("Enter Date (YYYY-MM-DD): ");
    scanf("%s", matches[*count].date);
    printf("Enter Venue: ");
    scanf("%s", matches[*count].venue);
    (*count)++;
    printf("Match scheduled!\n");
}

// Function to display all scheduled matches
void displayMatches(struct Match* matches, int count) {
    for (int i = 0; i < count; i++) {
        printf("%s vs %s, Date: %s, Venue: %s\n",
               matches[i].team1, matches[i].team2, matches[i].date, matches[i].venue);
    }
    if (count == 0) printf("No matches scheduled.\n");
```

```
}

// Function to search for matches on a specific date
void searchMatchesByDate(struct Match* matches, int count, const char* date) {
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(matches[i].date, date) == 0) {
            printf("%s vs %s, Venue: %s\n", matches[i].team1, matches[i].team2,
matches[i].venue);
            found = 1;
        }
    }
    if (!found) printf("No matches found on this date.\n");
}

// Function to cancel a match
void cancelMatch(struct Match* matches, int* count, 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) {
            matches[i] = matches[--(*count)];  // Replace canceled match with the last match
            printf("Match canceled.\n");
            return;
        }
    }
    printf("Match not found!\n");
}
```

```
int main() {
    int count = 0, maxMatches = 5;
    struct Match* matches = malloc(maxMatches * sizeof(struct Match));

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

    int choice;
    do {
        printf("\n1. Schedule Match\n2. Display Matches\n3. Search Matches by Date\n4. Cancel Match\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            if (count == maxMatches) {
                maxMatches *= 2;
                matches = realloc(matches, maxMatches * sizeof(struct Match));
                if (!matches) {
                    printf("Memory reallocation failed!\n");
                    return 1;
                }
            }
            scheduleMatch(matches, &count);
        } else if (choice == 2) {
            displayMatches(matches, count);
```

```
        } else if (choice == 3) {
            char date[11];
            printf("Enter Date (YYYY-MM-DD): ");
            scanf("%s", date);
            searchMatchesByDate(matches, count, date);
        } else if (choice == 4) {
            char team1[30], team2[30], date[11];
            printf("Enter Team 1: ");
            scanf("%s", team1);
            printf("Enter Team 2: ");
            scanf("%s", team2);
            printf("Enter Date (YYYY-MM-DD): ");
            scanf("%s", date);
            cancelMatch(matches, &count, team1, team2, date);
        }
    } while (choice != 5);

    free(matches);
    printf("Goodbye!\n");
    return 0;
}
```

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>


// Structure to store country medal tally

struct CountryMedalTally {

    char country[30];

    int gold, silver, bronze;

};


// Function to add a new country's medal tally

void addCountry(struct CountryMedalTally* tally, int* count) {

    printf("Enter country name: ");

    scanf("%s", tally[*count].country);

    printf("Enter gold medals: ");

    scanf("%d", &tally[*count].gold);

    printf("Enter silver medals: ");

    scanf("%d", &tally[*count].silver);

    printf("Enter bronze medals: ");
```

```
    scanf("%d", &tally[*count].bronze);
    (*count)++;
}

// Function to display the medal tally
void displayMedalTally(struct CountryMedalTally* tally, int count) {
    printf("\nMedal Tally:\nCountry\t\tGold\tSilver\tBronze\n");
    for (int i = 0; i < count; i++) {
        printf("%-15s%d\t%d\t%d\n", tally[i].country, tally[i].gold, tally[i].silver, tally[i].bronze);
    }
}

// Function to find and display the country with the highest gold medals
void findHighestGold(struct CountryMedalTally* tally, int count) {
    int maxGoldIndex = 0;
    for (int i = 1; i < count; i++) {
        if (tally[i].gold > tally[maxGoldIndex].gold) maxGoldIndex = i;
    }
    printf("\nHighest Gold Medals: %s (%d gold medals)\n", tally[maxGoldIndex].country,
tally[maxGoldIndex].gold);
}

int main() {
    int count = 0, maxCountries = 5;
    struct CountryMedalTally* tally = malloc(maxCountries * sizeof(struct
CountryMedalTally));

    if (!tally) {
        printf("Memory allocation failed!\n");
```

```
    return 1;
  }

  int choice;
  while (1) {
    printf("\n1. Add Country\n2. Display Medal Tally\n3. Find Highest Gold Medals\n4. Exit\nEnter choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
      if (count == maxCountries) {
        maxCountries *= 2;
        tally = realloc(tally, maxCountries * sizeof(struct CountryMedalTally));
        if (!tally) {
          printf("Memory reallocation failed!\n");
          return 1;
        }
      }
      addCountry(tally, &count);
    } else if (choice == 2) {
      displayMedalTally(tally, count);
    } else if (choice == 3) {
      if (count > 0) findHighestGold(tally, count);
      else printf("No data available.\n");
    } else if (choice == 4) {
      break;
    } else {
      printf("Invalid choice! Try again.\n");
```

```
        }
    }

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

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>

// Structure to store athlete information
struct Athlete {
```

```
    char athleteID[10];
    char name[50];
    char sport[30];
    float personalBest;
    float lastPerformance;
};

// Function to add a new athlete
void addAthlete(struct Athlete* athletes, int* count) {
    printf("Enter Athlete ID, Name, Sport, Personal Best, and Last Performance:\n");
    scanf("%s %s %s %f %f", athletes[*count].athleteID, athletes[*count].name,
athletes[*count].sport,
        &athletes[*count].personalBest, &athletes[*count].lastPerformance);
    (*count)++;
}

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

// Function to display athletes in a specific sport
void displayAthletesInSport(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("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 display athletes with a new personal best
void displayNewPersonalBests(struct Athlete* athletes, int count) {
    printf("Athletes with New Personal Bests:\n");
    for (int i = 0; i < count; 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() {
    int count = 0, maxAthletes = 5;
    struct Athlete* athletes = malloc(maxAthletes * sizeof(struct Athlete));

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

    while (1) {
        int choice;
        printf("\n1. Add Athlete\n2. Update Performance\n3. Display Athletes in Sport\n");
        printf("4. Display New Personal Bests\n5. Exit\nEnter choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            if (count == maxAthletes) {
                maxAthletes *= 2;
                athletes = realloc(athletes, maxAthletes * sizeof(struct Athlete));
            }
            addAthlete(athletes, &count);
        } else if (choice == 2) {
            updatePerformance(athletes, count);
        } else if (choice == 3) {
            char sport[30];
            printf("Enter Sport: ");
            scanf("%s", sport);
```

```
            displayAthletesInSport(athletes, count, sport);
        } else if (choice == 4) {
            displayNewPersonalBests(athletes, count);
        } else if (choice == 5) {
            break;
        } else {
            printf("Invalid choice! Try again.\n");
        }
    }

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

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>

// Structure to represent equipment
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));
        if (!*inventory) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
    }
    printf("Enter Equipment ID, Name, Category, Quantity, Price Per Unit:\n");
    scanf("%s %s %s %d %f",
          (*inventory)[*count].equipmentID,
```

```
        (*inventory)[*count].name,
        (*inventory)[*count].category,
        &(*inventory)[*count].quantity,
        &(*inventory)[*count].pricePerUnit);
    (*count)++;
}

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

// Function to display all equipment in a specific category
void displayByCategory(struct Equipment* inventory, int count) {
    char category[20];
    printf("Enter Category: ");
    scanf("%s", category);
    for (int i = 0; i < count; i++) {
```

```
        if (strcmp(inventory[i].category, category) == 0) {
            printf("ID: %s, Name: %s, Quantity: %d, Price: %.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
float calculateTotalValue(struct Equipment* inventory, int count) {
    float total = 0;
    for (int i = 0; i < count; i++) {
        total += inventory[i].quantity * inventory[i].pricePerUnit;
    }
    return total;
}

int main() {
    int count = 0, capacity = 2;
    struct Equipment* inventory = malloc(capacity * sizeof(struct Equipment));
    if (!inventory) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    int choice;
    do {
        printf("\n1. Add Equipment\n2. Update Quantity\n3. Display by Category\n");
```

```
        printf("4. Calculate Total Value\n5. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addEquipment(&inventory, &count, &capacity); break;
            case 2: updateQuantity(inventory, count); break;
            case 3: displayByCategory(inventory, count); break;
            case 4:
                printf("Total Inventory Value: %.2f\n", calculateTotalValue(inventory, count));
                break;
            case 5: break;
            default: printf("Invalid choice!\n");
        }
    } while (choice != 5);

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

set 3 programs

1: Research Paper Database Management

Requirements:

Define a structure ResearchPaper with the following members:

char title[100]

char author[50]

char journal[50]

int year

char DOI[30]

Functions to:

Add a new research paper to the database.

Update the details of an existing paper using its DOI.

Display all papers published in a specific journal.

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

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

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Structure to represent a research paper

struct ResearchPaper {

    char title[100];

    char author[50];

    char journal[50];

    int year;

    char DOI[30];

};


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

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

    if (*count == *capacity) {

        *capacity *= 2;

        *papers = realloc(*papers, (*capacity) * sizeof(struct ResearchPaper));

        if (*papers == NULL) {

            printf("Memory allocation failed!\n");
```

```
            exit(1);
        }
    }
    printf("Enter Title: ");
    scanf(" %[^\n]%*c", (*papers)[*count].title);
    printf("Enter Author: ");
    scanf(" %[^\n]%*c", (*papers)[*count].author);
    printf("Enter Journal: ");
    scanf(" %[^\n]%*c", (*papers)[*count].journal);
    printf("Enter Year: ");
    scanf("%d", &(*papers)[*count].year);
    printf("Enter DOI: ");
    scanf("%s", (*papers)[*count].DOI);
    (*count)++;
}

// Function to update the details of an existing paper using its DOI
void updateResearchPaper(struct ResearchPaper* papers, int count, const char* DOI) {
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].DOI, DOI) == 0) {
            printf("Enter New Title: ");
            scanf(" %[^\n]%*c", papers[i].title);
            printf("Enter New Author: ");
            scanf(" %[^\n]%*c", papers[i].author);
            printf("Enter New Journal: ");
            scanf(" %[^\n]%*c", papers[i].journal);
            printf("Enter New Year: ");
            scanf("%d", &papers[i].year);
```

```
            printf("Enter New DOI: ");
            scanf("%s", papers[i].DOI);
            printf("Research Paper updated successfully!\n");
            return;
        }
    }
    printf("No paper found with DOI %s.\n", DOI);
}

// Function to display all papers published in a specific journal
void displayPapersByJournal(struct ResearchPaper* papers, int count, const char* journal) {
    printf("Papers in Journal: %s\n", journal);
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].journal, journal) == 0) {
            printf("Title: %s, Author: %s, Year: %d, DOI: %s\n",
                   papers[i].title, papers[i].author, papers[i].year, papers[i].DOI);
        }
    }
}

// Function to find and display the most recent papers published by a specific author
void displayRecentPapersByAuthor(struct ResearchPaper* papers, int count, const char*
author) {
    int latestYear = -1;
    for (int i = 0; i < count; i++) {
        if (strcmp(papers[i].author, author) == 0 && papers[i].year > latestYear) {
            latestYear = papers[i].year;
        }
```

```
    }

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

int main() {
    int count = 0, capacity = 2;
    struct ResearchPaper* papers = malloc(capacity * sizeof(struct ResearchPaper));
    if (papers == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    int choice;
    do {
        printf("\n1. Add Research Paper\n2. Update Research Paper\n3. Display Papers by
Journal\n");
        printf("4. Display Recent Papers by Author\n5. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addResearchPaper(&papers, &count, &capacity); break;
            case 2: {
```

```
            char DOI[30];
            printf("Enter DOI of paper to update: ");
            scanf("%s", DOI);
            updateResearchPaper(papers, count, DOI);
            break;
        }
        case 3: {
            char journal[50];
            printf("Enter Journal name: ");
            scanf(" %[^\n]%*c", journal);
            displayPapersByJournal(papers, count, journal);
            break;
        }
        case 4: {
            char author[50];
            printf("Enter Author name: ");
            scanf(" %[^\n]%*c", author);
            displayRecentPapersByAuthor(papers, count, author);
            break;
        }
        case 5: break;
        default: printf("Invalid choice! Please try again.\n");
      }
   } while (choice != 5);

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

2: Experimental Data Logger

Requirements:

Create a structure Experiment with members:

char experimentID[10]

char researcher[50]

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

char endDate[11]

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

Functions to:

Log a new experiment.

Update the result readings of an experiment.

Display all experiments conducted by a specific researcher.

Calculate and display the average result for a specific experiment.

Use a dynamically allocated array for storing experiments and manage resizing as more data is logged.

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

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

```
// Function to log a new experiment
void logExperiment(struct Experiment* experiments, int* count, int* capacity) {
    if (*count == *capacity) {
        *capacity *= 2;
        experiments = realloc(experiments, (*capacity) * sizeof(struct Experiment));
        if (!experiments) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
    }

    printf("Enter Experiment ID, Researcher, 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 results (up to 10, -1 to stop):\n");
    for (int i = 0; i < 10; i++) {
        scanf("%f", &experiments[*count].results[i]);
        if (experiments[*count].results[i] == -1) break;
    }

    (*count)++;
}

// Function to display experiments by researcher
```

```
void displayExperimentsByResearcher(struct Experiment* experiments, int count, const char* researcher) {
    printf("Experiments by %s:\n", researcher);
    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].researcher, researcher) == 0) {
            printf("%s | %s to %s | Results: ", experiments[i].experimentID, experiments[i].startDate, experiments[i].endDate);
            for (int j = 0; j < 10; j++) {
                if (experiments[i].results[j] == -1) break;
                printf("%.2f ", experiments[i].results[j]);
            }
            printf("\n");
        }
    }
}

// Function to calculate average result for an experiment
void calculateAverageResult(struct Experiment* experiments, int count, const char* experimentID) {
    for (int i = 0; i < count; i++) {
        if (strcmp(experiments[i].experimentID, experimentID) == 0) {
            float total = 0;
            int validResults = 0;
            for (int j = 0; j < 10; j++) {
                if (experiments[i].results[j] == -1) break;
                total += experiments[i].results[j];
                validResults++;
            }
            if (validResults > 0)
```

```
            printf("Average result for %s: %.2f\n", experimentID, total / validResults);
        else
            printf("No results available for %s.\n", experimentID);
        return;
    }
  }
  printf("Experiment ID not found!\n");
}

int main() {
  int count = 0, capacity = 2;
  struct Experiment* experiments = malloc(capacity * sizeof(struct Experiment));
  if (!experiments) {
    printf("Memory allocation failed!\n");
    return 1;
  }

  int choice;
  while (1) {
    printf("\n1. Log Experiment\n2. Display Experiments by Researcher\n3. Calculate
Average Result\n4. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
      logExperiment(experiments, &count, &capacity);
    } else if (choice == 2) {
      char researcher[50];
```

```
            printf("Enter Researcher Name: ");
            scanf(" %[^\n]", researcher);
            displayExperimentsByResearcher(experiments, count, researcher);
        } else if (choice == 3) {
            char experimentID[10];
            printf("Enter Experiment ID: ");
            scanf("%s", experimentID);
            calculateAverageResult(experiments, count, experimentID);
        } else if (choice == 4) {
            break;
        } else {
            printf("Invalid choice!\n");
        }
    }

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

3: Grant Application Tracker

Requirements:

Define a structure GrantApplication with the following members:

char applicationID[10]

char applicantName[50]

char projectTitle[100]

float requestedAmount

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

Functions to:

Add a new grant application.

Update the status of an application.

Display all applications requesting an amount greater than a specified value.

Find and display applications that are currently "Approved."

Store the grant applications in a dynamically allocated array, resizing it as necessary.

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

// Structure to represent a grant application
struct GrantApplication {
    char applicationID[10];
    char applicantName[50];
    char projectTitle[100];
    float requestedAmount;
    char status[20]; // e.g., "Submitted", "Approved", "Rejected"
};

// Function to add a new grant application
void addGrantApplication(struct GrantApplication* applications, int* count, int* capacity) {
    if (*count == *capacity) {
        *capacity *= 2; // Double the size of the array
        applications = realloc(applications, (*capacity) * sizeof(struct GrantApplication));
        if (!applications) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
```

```
    }

    printf("Enter Application ID: ");
    scanf("%s", applications[*count].applicationID);

    printf("Enter Applicant Name: ");
    scanf(" %[^\n]", applications[*count].applicantName);

    printf("Enter Project Title: ");
    scanf(" %[^\n]", applications[*count].projectTitle);

    printf("Enter Requested Amount: ");
    scanf("%f", &applications[*count].requestedAmount);

    printf("Enter Status (e.g., Submitted, Approved, Rejected): ");
    scanf("%s", applications[*count].status);

    (*count)++;
}

// Function to update the status of an application
void updateStatus(struct GrantApplication* applications, int count, const char*
applicationID, const char* newStatus) {
    for (int i = 0; i < count; i++) {
        if (strcmp(applications[i].applicationID, applicationID) == 0) {
            strcpy(applications[i].status, newStatus);
            printf("Application status updated to %s.\n", newStatus);
            return;
```

```
        }
    }
    printf("Application ID not found!\n");
}

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

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

```
    }
}

int main() {
    int count = 0, capacity = 2;
    struct GrantApplication* applications = malloc(capacity * sizeof(struct GrantApplication));

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

    int choice;
    while (1) {
        printf("\n1. Add Grant Application\n2. Update Application Status\n3. Display
Applications by Amount\n");
        printf("4. Display Approved Applications\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            addGrantApplication(applications, &count, &capacity);
        } else if (choice == 2) {
            char applicationID[10], newStatus[20];
            printf("Enter Application ID to update: ");
            scanf("%s", applicationID);
            printf("Enter new status: ");
            scanf("%s", newStatus);
```

```
            updateStatus(applications, count, applicationID, newStatus);
        } else if (choice == 3) {
            float amount;
            printf("Enter the requested amount to filter: ");
            scanf("%f", &amount);
            displayApplicationsByAmount(applications, count, amount);
        } else if (choice == 4) {
            displayApprovedApplications(applications, count);
        } else if (choice == 5) {
            break;
        } else {
            printf("Invalid choice!\n");
        }
    }

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

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.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Structure to represent a research collaborator

struct Collaborator {

   char collaboratorID[10];

   char name[50];

   char institution[50];

   char expertiseArea[30];

   int numberOfProjects;

};


// Function to add a new collaborator to the database

void addCollaborator(struct Collaborator* collaborators, int* count, int* capacity) {

   if (*count == *capacity) {

      *capacity *= 2; // Double the size of the array if it reaches capacity

      collaborators = realloc(collaborators, (*capacity) * sizeof(struct Collaborator));

      if (!collaborators) {

         printf("Memory allocation failed!\n");

         exit(1);
```

```
        }
    }

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

    printf("Enter Name: ");
    scanf(" %[^\n]", collaborators[*count].name);

    printf("Enter Institution: ");
    scanf(" %[^\n]", collaborators[*count].institution);

    printf("Enter Expertise Area: ");
    scanf(" %[^\n]", 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 updateProjects(struct Collaborator* collaborators, int count, const char* collaboratorID, int newProjects) {
    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].collaboratorID, collaboratorID) == 0) {
            collaborators[i].numberOfProjects = newProjects;
            printf("Updated number of projects for %s to %d.\n", collaborators[i].name, newProjects);
```

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

// Function to display all collaborators from a specific institution
void displayByInstitution(struct Collaborator* collaborators, int count, const char* institution) {
    printf("Collaborators from %s:\n", institution);
    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].institution, institution) == 0) {
            printf("ID: %s, Name: %s, Expertise Area: %s, Projects: %d\n",
                collaborators[i].collaboratorID, collaborators[i].name,
                collaborators[i].expertiseArea, collaborators[i].numberOfProjects);
        }
    }
}

// Function to find and display collaborators with expertise in a given area
void displayByExpertise(struct Collaborator* collaborators, int count, const char* expertiseArea) {
    printf("Collaborators with expertise in %s:\n", expertiseArea);
    for (int i = 0; i < count; i++) {
        if (strcmp(collaborators[i].expertiseArea, expertiseArea) == 0) {
            printf("ID: %s, Name: %s, Institution: %s, Projects: %d\n",
                collaborators[i].collaboratorID, collaborators[i].name,
                collaborators[i].institution, collaborators[i].numberOfProjects);
        }
```

```
    }
}

int main() {
    int count = 0, capacity = 2;
    struct Collaborator* collaborators = malloc(capacity * sizeof(struct Collaborator));

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

    int choice;
    while (1) {
        printf("\n1. Add Collaborator\n2. Update Number of Projects\n3. Display Collaborators by Institution\n");
        printf("4. Display Collaborators by Expertise Area\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            addCollaborator(collaborators, &count, &capacity);
        } else if (choice == 2) {
            char collaboratorID[10];
            int newProjects;
            printf("Enter Collaborator ID to update: ");
            scanf("%s", collaboratorID);
            printf("Enter new number of projects: ");
```

```
            scanf("%d", &newProjects);
            updateProjects(collaborators, count, collaboratorID, newProjects);
        } else if (choice == 3) {
            char institution[50];
            printf("Enter Institution to filter by: ");
            scanf(" %[^\n]", institution);
            displayByInstitution(collaborators, count, institution);
        } else if (choice == 4) {
            char expertiseArea[30];
            printf("Enter Expertise Area to filter by: ");
            scanf(" %[^\n]", expertiseArea);
            displayByExpertise(collaborators, count, expertiseArea);
        } else if (choice == 5) {
            break;
        } else {
            printf("Invalid choice!\n");
        }
    }

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

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>


// Structure to represent a conference paper submission

struct ConferenceSubmission {

   char submissionID[10];

   char authorName[50];

   char paperTitle[100];

   char conferenceName[50];

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

   char status[20];         // Pending, Accepted, Rejected

};


// Function to add a new conference submission

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

```
if (*count == *capacity) {
    *capacity *= 2;  // Double the size of the array if it reaches capacity
    submissions = realloc(submissions, (*capacity) * sizeof(struct ConferenceSubmission));
    if (!submissions) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}

printf("Enter Submission ID: ");
scanf("%s", submissions[*count].submissionID);

printf("Enter Author Name: ");
scanf(" %[^\n]", submissions[*count].authorName);

printf("Enter Paper Title: ");
scanf(" %[^\n]", submissions[*count].paperTitle);

printf("Enter Conference Name: ");
scanf(" %[^\n]", submissions[*count].conferenceName);

printf("Enter Submission Date (YYYY-MM-DD): ");
scanf("%s", submissions[*count].submissionDate);

printf("Enter Status (Pending, Accepted, Rejected): ");
scanf(" %[^\n]", submissions[*count].status);

(*count)++;
```

```
}

// Function to update the status of a submission
void updateStatus(struct ConferenceSubmission* submissions, int count, const char* submissionID, const char* newStatus) {
    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].submissionID, submissionID) == 0) {
            strcpy(submissions[i].status, newStatus);
            printf("Updated submission status to %s for Submission ID %s.\n", newStatus, submissionID);
            return;
        }
    }
    printf("Submission with ID %s not found!\n", submissionID);
}

// Function to display all submissions to a specific conference
void displayByConference(struct ConferenceSubmission* submissions, int count, const char* conferenceName) {
    printf("Submissions to %s:\n", conferenceName);
    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].conferenceName, conferenceName) == 0) {
            printf("ID: %s, Author: %s, Title: %s, Date: %s, Status: %s\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 displayByAuthor(struct ConferenceSubmission* submissions, int count, const char* authorName) {
    printf("Submissions by %s:\n", authorName);
    for (int i = 0; i < count; i++) {
        if (strcmp(submissions[i].authorName, authorName) == 0) {
            printf("ID: %s, Conference: %s, Title: %s, Date: %s, Status: %s\n",
                   submissions[i].submissionID, submissions[i].conferenceName,
                   submissions[i].paperTitle, submissions[i].submissionDate,
                   submissions[i].status);
        }
    }
}

int main() {
    int count = 0, capacity = 2;
    struct ConferenceSubmission* submissions = malloc(capacity * sizeof(struct ConferenceSubmission));

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

    int choice;
    while (1) {
        printf("\n1. Add Submission\n2. Update Submission Status\n3. Display Submissions by Conference\n");
```

```
printf("4. Display Submissions by Author\n5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

if (choice == 1) {
    addSubmission(submissions, &count, &capacity);
} else if (choice == 2) {
    char submissionID[10], newStatus[20];
    printf("Enter Submission ID to update: ");
    scanf("%s", submissionID);
    printf("Enter new status: ");
    scanf(" %[^\n]", newStatus);
    updateStatus(submissions, count, submissionID, newStatus);
} else if (choice == 3) {
    char conferenceName[50];
    printf("Enter Conference Name to filter by: ");
    scanf(" %[^\n]", conferenceName);
    displayByConference(submissions, count, conferenceName);
} else if (choice == 4) {
    char authorName[50];
    printf("Enter Author Name to filter by: ");
    scanf(" %[^\n]", authorName);
    displayByAuthor(submissions, count, authorName);
} else if (choice == 5) {
    break;
} else {
    printf("Invalid choice!\n");
}
```

```
    }

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