Problem 1: Vehicle Fleet Management System

Requirements:

- Create a structure Vehicle with the following members:
  - char registrationNumber[15]
  - char model[30]
  - int yearOfManufacture
  - float mileage
  - float fuelEfficiency
- Implement functions to:
- Add a new vehicle to the fleet.
- Update the mileage and fuel efficiency for a vehicle.
- Display all vehicles manufactured after a certain year.
- Find the vehicle with the highest fuel efficiency.
- Use dynamic memory allocation to manage the fleet of vehicles.

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

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

```
    float fuelEfficiency;
};

//Function prototypes
void addVehicle(struct Vehicle** fleet, int* fleetSize);
void updateVehicle(struct Vehicle* fleet, int fleetSize, const char* registrationNumber);
void displayVehicles(struct Vehicle* fleet, int fleetSize, int year);
void findHighestFuelEfficiency(struct Vehicle* fleet, int fleetSize);

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

    int option;
    do
    {
        printf("\nVehicle Fleet Management System");
        printf("\n1. Add a new vehicle\n");
        printf("2. Update vehicle information\n");
        printf("3. Display vehicles manufactured after a certain year\n");
        printf("4. Find vehicle with highest fuel efficiency\n");
        printf("5. Exit\n");
        printf("Enter your option: ");
        scanf("%d", &option);

        switch (option)
```

```
        {
            case 1: addVehicle(&fleet, &fleetSize);
                    break;
            case 2: char regNumber[15];
                    printf("Enter registration number of the vehicle to update: ");
                    scanf("%s", regNumber);
                    updateVehicle(fleet, fleetSize, regNumber);
                    break;
            case 3: int year;
                    printf("Enter the year: ");
                    scanf("%d", &year);
                    displayVehicles(fleet, fleetSize, year);
                    break;
            case 4: findHighestFuelEfficiency(fleet, fleetSize);
                    break;
            case 5: free(fleet);
                    printf("Exit the program\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);
    return 0;
}

// Function to add a new vehicle to the fleet
void addVehicle(struct Vehicle** fleet, int* fleetSize)
{
```

```
    if (*fleet == 0)
        *fleet = malloc(sizeof(struct Vehicle));
    else
        *fleet = malloc((*fleetSize + 1) * sizeof(struct Vehicle));

    printf("Enter registration number: ");
    scanf("%s", (*fleet)[*fleetSize].registrationNumber);

    printf("Enter model: ");
    scanf("%s", (*fleet)[*fleetSize].model);

    printf("Enter year of manufacture: ");
    scanf("%d", &(*fleet)[*fleetSize].yearOfManufacture);

    printf("Enter mileage: ");
    scanf("%f", &(*fleet)[*fleetSize].mileage);

    printf("Enter fuel efficiency: ");
    scanf("%f", &(*fleet)[*fleetSize].fuelEfficiency);

    (*fleetSize)++;
}

// Function to update the mileage and fuel efficiency of a vehicle
void updateVehicle(struct Vehicle* fleet, int fleetSize, const char* registrationNumber)
{
    for (int i = 0; i < fleetSize; 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 not found!\n");
}

// Function to display all vehicles manufactured after a certain year
void displayVehicles(struct Vehicle* fleet, int fleetSize, int year)
{
    int found = 0;
    for (int i = 0; i < fleetSize; 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);
            found = 1;
```

```
        }
    }
    if (!found)
        printf("No vehicles found manufactured after year %d\n", year);
}

// Function to find the vehicle with the highest fuel efficiency
void findHighestFuelEfficiency(struct Vehicle* fleet, int fleetSize)
{
    if (fleetSize == 0)
    {
        printf("Fleet is empty\n");
        return;
    }

    struct Vehicle* highfuelveh = &fleet[0];
    for (int i = 1; i < fleetSize; i++)
    {
        if (fleet[i].fuelEfficiency > highfuelveh->fuelEfficiency)
            highfuelveh = &fleet[i];
    }

    printf("Vehicle with highest fuel efficiency:\n");
    printf("Registration Number: %s, Model: %s, Year: %d, Mileage: %.2f, Fuel Efficiency: %.2f\n",
        highfuelveh->registrationNumber, highfuelveh->model, highfuelveh->yearOfManufacture,
        highfuelveh->mileage, highfuelveh->fuelEfficiency);
```

}

Problem 2: Car Rental Reservation System

Requirements:

- Define a structure CarRental with members:
  - char carID[10]
  - char customerName[50]
  - char rentalDate[11] (format: YYYY-MM-DD)
  - char returnDate[11]
  - float rentalPricePerDay
- Write functions to:
- Book a car for a customer by inputting necessary details.
- Calculate the total rental price based on the number of rental days.
- Display all current rentals.
- Search for rentals by customer name.
- Implement error handling for invalid dates and calculate the number of rental days.

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

struct CarRental
{
   char carID[10];
   char customerName[50];
   char rentalDate[11];
   char returnDate[11];
```

```
    float rentalPricePerDay;
};

struct CarRental rentals[100];
int rentalCount = 0;

// Function prototypes
void bookCar();
void calculateTotalRentalPrice();
void displayRentals();
void searchRentalByCustomer();

int main()
{
    int option;
    do
    {
        printf("\nCar Rental Reservation System\n");
         printf("1. Book a Car\n2. Calculate Total Rental Price\n3. Display Rentals\n4. Search Rental by Customer name\n5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: bookCar();
                    break;
            case 2: calculateTotalRentalPrice();
                    break;
```

```
            case 3: displayRentals();
                    break;
            case 4: searchRentalByCustomer();
                    break;
            case 5: printf("Exit the program\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);
    return 0;
}

// Function to book a car
void bookCar()
{
    if (rentalCount >= 100)
    {
        printf("Maximum rental limit reached\n");
        return;
    }
    struct CarRental new;
    printf("Enter car ID: ");
    scanf("%s", new.carID);
    printf("Enter customer name: ");
    scanf(" %[^\n]", new.customerName);
    printf("Enter rental date (YYYY-MM-DD): ");
    scanf("%s", new.rentalDate);
```

```
    printf("Enter return date (YYYY-MM-DD): ");
    scanf("%s", new.returnDate);
    printf("Enter rental price per day: ");
    scanf("%f", &new.rentalPricePerDay);

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

// Function to calculate total rental price
void calculateTotalRentalPrice()
{
    char carID[10];
    printf("Enter Car ID to calculate total rental price: ");
    scanf("%s", carID);

    for (int i = 0; i < rentalCount; i++)
    {
        if (strcmp(rentals[i].carID, carID) == 0)
        {
            int rentalDays = 1;
            float totalCost = rentalDays * rentals[i].rentalPricePerDay;
            printf("Total rental price for car %s: %.2f\n", carID, totalCost);
            return;
        }
    }
    printf("Car ID not found\n");
```

```
}

// Function to display all current rentals
void displayRentals()
{
    if (rentalCount == 0)
    {
        printf("No rentals available\n");
        return;
    }
    for (int i = 0; i < rentalCount; i++)
    {
         printf("Car ID: %s Customer Name: %s  Rental Date: %s  Return Date: %s  Price Per Day: %.2f\n",
                rentals[i].carID, rentals[i].customerName, rentals[i].rentalDate,
                rentals[i].returnDate, rentals[i].rentalPricePerDay);
    }
}

// Function to search rentals by customer name
void searchRentalByCustomer()
{
    char customerName[50];
    printf("Enter Customer Name to search: ");
    scanf(" %[^\n]", customerName);

    int found = 0;
    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);
}
```

Problem 3: Autonomous Vehicle Sensor Data Logger

Requirements:

- Create a structure SensorData with fields:
  - int sensorID
  - char timestamp[20] (format: YYYY-MM-DD HH:MM:SS)
  - float speed
  - float latitude
  - float longitude
- Functions to:
- Log new sensor data.
- Display sensor data for a specific time range.
- Find the maximum speed recorded.
- Calculate the average speed over a specific time period.

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

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

// Define the structure for SensorData
struct SensorData
{
   int sensorID;
   char timestamp[20];
   float speed;
   float latitude;
   float longitude;
};

struct SensorData *sensor;
int recordCount = 0;
int recordCapacity = 10;

// Function prototypes
void logSensorData();
void displaySensorData();
void findMaximumSpeed();
void calculateAverageSpeed();

int main()
```

```
{
    sensor = (struct SensorData *)malloc(recordCapacity * sizeof(struct SensorData));

   int option;

   do
   {
      printf("\nAutonomous Vehicle Sensor Data Logger\n");
       printf("1. Log sensor data\n2. Display data for time range\n3. Find maximum speed\n4. Calculate average speed\n5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);

      switch (option)
      {
         case 1: logSensorData();
                 break;
         case 2: displaySensorData();
                 break;
         case 3: findMaximumSpeed();
                 break;
         case 4: calculateAverageSpeed();
                 break;
         case 5: printf("Exit the program\n");
                 break;
         default: printf("Invalid option\n");
      }
```

```
    } while (option != 5);

    free(sensor);
    return 0;
}

// Function to log new sensor data
void logSensorData()
{
    if (recordCount >= recordCapacity)
    {
        struct SensorData* new = (struct SensorData*)malloc(recordCapacity
* 2 * sizeof(struct SensorData));
        if (!new)
        {
            printf("Memory allocation failed\n");
            exit(1);
        }
        for (int i = 0; i < recordCapacity; i++)
            new[i] = sensor[i];
        free(sensor);
        sensor = new;
        recordCapacity *= 2;
    }

    struct SensorData newRecord;
    printf("Enter sensor ID: ");
    scanf("%d", &newRecord.sensorID);
```

```
    printf("Enter timestamp (YYYY-MM-DD HH:MM:SS): ");
    scanf(" %[^\n]", newRecord.timestamp);
    printf("Enter speed: ");
    scanf("%f", &newRecord.speed);
    printf("Enter latitude: ");
    scanf("%f", &newRecord.latitude);
    printf("Enter longitude: ");
    scanf("%f", &newRecord.longitude);

    sensor[recordCount++] = newRecord;
    printf("Sensor data logged successfully\n");
}

// Function to display sensor data for a specific time range
void displaySensorData()
{
    char start[20], end[20];
    printf("Enter start time (YYYY-MM-DD HH:MM:SS): ");
    scanf(" %[^\n]", start);
    printf("Enter End Time (YYYY-MM-DD HH:MM:SS): ");
    scanf(" %[^\n]", end);

    int found = 0;
    for (int i = 0; i < recordCount; i++)
    {
                if (strcmp(sensor[i].timestamp, start) >= 0 &&
strcmp(sensor[i].timestamp, end) <= 0)
        {
```

```
            printf("Sensor ID: %d  Timestamp: %s  Speed: %.2f  Latitude: %.2f  Longitude: %.2f\n",
                sensor[i].sensorID, sensor[i].timestamp,
                sensor[i].speed, sensor[i].latitude, sensor[i].longitude);
            found = 1;
        }
    }
    if (!found)
        printf("No sensor data found in the given time range\n");
}

// Function to find the maximum speed recorded
void findMaximumSpeed()
{
    if (recordCount == 0)
    {
        printf("No sensor data available\n");
        return;
    }

    float maxSpeed = sensor[0].speed;
    for (int i = 1; i < recordCount; i++)
    {
        if (sensor[i].speed > maxSpeed)
            maxSpeed = sensor[i].speed;
    }
    printf("Maximum speed recorded: %.2f\n", maxSpeed);
}
```

```
// Function to calculate the average speed over all records
void calculateAverageSpeed()
{
    if (recordCount == 0)
    {
        printf("No sensor data available\n");
        return;
    }

    float totalSpeed = 0;
    for (int i = 0; i < recordCount; i++)
        totalSpeed += sensor[i].speed;

    printf("Average Speed: %.2f\n", totalSpeed / recordCount);
}
```

Problem 4: Engine Performance Monitoring System

Requirements:

- Define a structure EnginePerformance with members:
    - char engineID[10]
    - float temperature
    - float rpm
    - float fuelConsumptionRate
    - float oilPressure
- Functions to:

- Add performance data for a specific engine.
- Display all performance data for a specific engine ID.
- Calculate the average temperature and RPM for a specific engine.
- Identify any engine with abnormal oil pressure (above or below specified thresholds).
- Use linked lists to store and manage performance data entries.

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

#define MAX_RECORDS 100
#define OIL_PRESSURE_MIN 20.0
#define OIL_PRESSURE_MAX 80.0

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

struct EnginePerformance performanceData[MAX_RECORDS];
int recordCount = 0;

//Function prototypes
void addPerformanceData();
```

```
void displayPerformanceData(const char* engineID);
void calculateAverageTemperatureAndRPM(const char* engineID);
void identifyAbnormalOilPressure();

int main()
{
   int option;
   char engineID[10];

   do
   {
      printf("\nEngine Performance Monitoring System\n");
      printf("1. Add performance data\n");
      printf("2. Display performance data for an engine\n");
      printf("3. Calculate average temperature and RPM for an engine\n");
      printf("4. Identify engines with abnormal oil pressure\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);

      switch (option)
      {
         case 1: addPerformanceData();
                 break;
         case 2: printf("Enter Engine ID: ");
                 scanf("%s", engineID);
                 displayPerformanceData(engineID);
```

```
            break;
        case 3: printf("Enter Engine ID: ");
            scanf("%s", engineID);
            calculateAverageTemperatureAndRPM(engineID);
            break;
        case 4: identifyAbnormalOilPressure();
            break;
        case 5: printf("Exit the program\n");
            break;
        default: printf("Invalid option\n");
      }
   } while (option != 5);
   return 0;
}

// Function to add performance data for a specific engine
void addPerformanceData()
{
   if (recordCount >= MAX_RECORDS)
   {
      printf("Maximum record limit reached\n");
      return;
   }

   printf("Enter engine ID: ");
   scanf("%s", performanceData[recordCount].engineID);
```

```
    printf("Enter temperature: ");
    scanf("%f", &performanceData[recordCount].temperature);

    printf("Enter RPM: ");
    scanf("%f", &performanceData[recordCount].rpm);

    printf("Enter fuel consumption rate: ");
    scanf("%f", &performanceData[recordCount].fuelConsumptionRate);

    printf("Enter oil pressure: ");
    scanf("%f", &performanceData[recordCount].oilPressure);

    recordCount++;
    printf("Performance data added successfully\n");
}

// Function to display all performance data for a specific engine ID
void displayPerformanceData(const char* engineID)
{
    int found = 0;

    for (int i = 0; i < recordCount; i++)
    {
        if (strcmp(performanceData[i].engineID, engineID) == 0)
        {
            printf("\nRecord %d:\n", i + 1);
            printf("Engine ID: %s\n", performanceData[i].engineID);
```

```
            printf("Temperature: %.2f\n", performanceData[i].temperature);
            printf("RPM: %.2f\n", performanceData[i].rpm);
                                printf("Fuel Consumption Rate: %.2f\n", performanceData[i].fuelConsumptionRate);
            printf("Oil Pressure: %.2f\n", performanceData[i].oilPressure);
            found = 1;
        }
    }
    if (!found)
        printf("No records found for engine ID: %s\n", engineID);
}

// Function to calculate the average temperature and RPM for a specific engine
void calculateAverageTemperatureAndRPM(const char* engineID)
{
    float totalTemperature = 0.0;
    float totalRPM = 0.0;
    int count = 0;

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

```
    }

    if (count > 0)
    {
        printf("Average temperature for engine %s: %.2f\n", engineID, totalTemperature / count);
        printf("Average RPM for engine %s: %.2f\n", engineID, totalRPM / count);
    }
    else
        printf("No records found for engine ID: %s\n", engineID);
}

// Function to identify engines with abnormal oil pressure
void identifyAbnormalOilPressure()
{
    int found = 0;

    for (int i = 0; i < recordCount; i++)
    {
        if (performanceData[i].oilPressure < OIL_PRESSURE_MIN || performanceData[i].oilPressure > OIL_PRESSURE_MAX)
        {
            printf("\nAbnormal oil pressure detected\n");
            printf("Engine ID: %s\n", performanceData[i].engineID);
            printf("Oil pressure: %.2f\n", performanceData[i].oilPressure);
            found = 1;
        }
```

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

Problem 5: Vehicle Service History Tracker

Requirements:

- Create a structure ServiceRecord with the following:
    - char serviceID[10]
    - char vehicleID[15]
    - char serviceDate[11]
    - char description[100]
    - float serviceCost
- Functions to:
- Add a new service record for a vehicle.
- Display all service records for a given vehicle ID.
- Calculate the total cost of services for a vehicle.
- Sort and display service records by service date.

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

// Define the structure for ServiceRecord
struct ServiceRecord
{
```

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

struct ServiceRecord *records;
int recordCount = 0;
int recordCapacity = 10;

// Function prototypes
void addServiceRecord();
void displayServiceRecords();
void calculateTotalServiceCost();
void sortAndDisplayRecords();
int compareDates(const void *a, const void *b);

int main()
{
    records = (struct ServiceRecord *)malloc(recordCapacity * sizeof(struct
ServiceRecord));
    if (!records)
    {
        printf("Memory allocation failed.\n");
        return 1;
    }
    int option;
```

```
    do
    {
        printf("\nVehicle Service History Tracker\n");
         printf("1. Add service record\n2. Display records by vehicle ID\n3. Calculate total service cost\n4. Sort and display records by date\n5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: addServiceRecord();
                    break;
            case 2: displayServiceRecords();
                    break;
            case 3: calculateTotalServiceCost();
                    break;
            case 4: sortAndDisplayRecords();
                    break;
            case 5: printf("Exit the program\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);

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

```
// Function to resize dynamic array
void resizeRecords() {
    recordCapacity *= 2;
     records = (struct ServiceRecord *)realloc(records, recordCapacity * sizeof(struct ServiceRecord));
    if (!records) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
}

// Function to add a new service record
void addServiceRecord()
{
    if (recordCount >= recordCapacity)
    {
                        struct ServiceRecord* new = (struct ServiceRecord*)malloc(recordCapacity * 2 * sizeof(struct ServiceRecord));
        if (!new)
        {
            printf("Memory allocation failed\n");
            exit(1);
        }
        for (int i = 0; i < recordCapacity; i++)
            new[i] = records[i];
        free(records);
        records = new;
```

```
        recordCapacity *= 2;
    }

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

    records[recordCount++] = newRecord;
    printf("Service record added successfully\n");
}

// Function to display all service records for a specific vehicle ID
void displayServiceRecords()
{
    char vehicleID[15];
    printf("Enter vehicle ID to display records: ");
    scanf("%s", vehicleID);

    int found = 0;
```

```
    for (int i = 0; i < recordCount; 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);
            found = 1;
        }
    }

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

// Function to calculate total service cost for a vehicle
void calculateTotalServiceCost()
{
    char vehicleID[15];
    printf("Enter vehicle ID to calculate total cost: ");
    scanf("%s", vehicleID);

    float totalCost = 0;
    int found = 0;

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

```
        if (strcmp(records[i].vehicleID, vehicleID) == 0)
        {
            totalCost += records[i].serviceCost;
            found = 1;
        }
    }

    if (found)
        printf("Total service cost for vehicle ID %s: %.2f\n", vehicleID, totalCost);
    else
        printf("No service records found for vehicle ID: %s\n", vehicleID);
}

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

void sortAndDisplayRecords()
{
    if (recordCount == 0)
    {
        printf("No service records available to sort\n");
        return;
```

```
    }

        qsort(records, recordCount, sizeof(struct ServiceRecord), compareDates);
    printf("Service records sorted by date\n");
    for (int i = 0; i < recordCount; 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);
    }
}
```

Problem 1: Player Statistics Management

- Requirements:

  Define a structure Player with the following members:

  - char name[50]
  - int age
  - char team[30]
  - int matchesPlayed

    - int totalRuns
    - int totalWickets
- Functions to:
    - Add a new player to the system.
    - Update a player's statistics after a match.
    - Display the details of players from a specific team.
    - Find the player with the highest runs and the player with the most wickets.
    - Use dynamic memory allocation to store player data in an array and expand it as needed.

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

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

```
// Function prototypes
void addPlayer(struct Player** players, int* playerCount, int* capacity);
void updatePlayer(struct Player* players, int playerCount);
void displayPlayers(struct Player* players, int playerCount, const char* team);
void highrunAndWicket(struct Player* players, int playerCount);

int main()
{
    struct Player* players;
    int playerCount = 0;
    int capacity = 3;
    players = malloc(capacity * sizeof(struct Player));
    if (!players)
    {
        printf("Memory allocation failed\n");
        return 1;
    }

    int option;
    do
    {
```

```
printf("\nPlayer Statistics Management System\n");
printf("1. Add a new player\n");
printf("2. Update player statistics\n");
printf("3. Display players by team\n");
printf("4. Find top players\n");
printf("5. Exit\n");
printf("Enter the option: ");
scanf("%d", &option);

switch (option)
{
   case 1: addPlayer(&players, &playerCount, &capacity);
           break;
   case 2: updatePlayer(players, playerCount);
           break;
   case 3: char team[30];
           printf("Enter team name: ");
           scanf("%s", team);
           displayPlayers(players, playerCount, team);
           break;
   case 4: highrunAndWicket(players, playerCount);
           break;
```

```
            case 5: printf("Exit the program\n");
                    break;
            default:printf("Invalid option\n");
        }
    } while (option != 5);

    free(players);
    return 0;
}

// Function to add a new player
void addPlayer(struct Player** players, int* playerCount, int* capacity)
{
    if(*playerCount == *capacity)
    {
        *capacity *= 2;
        struct Player* new = malloc((*capacity) * sizeof(struct Player));
        if (!new)
        {
            printf("Memory allocation failed\n");
            exit(1);
        }
```

```
    for (int i = 0; i < *playerCount; i++)
        new[i] = (*players)[i];
    free(*players);
    *players = new;
}

struct Player* newPlayer = &(*players)[*playerCount];
printf("Enter player name: ");
scanf("%s", newPlayer->name);
printf("Enter age: ");
scanf("%d", &newPlayer->age);
printf("Enter team: ");
scanf("%s", newPlayer->team);
printf("Enter matches played: ");
scanf("%d", &newPlayer->matchesPlayed);
printf("Enter total runs: ");
scanf("%d", &newPlayer->totalRuns);
printf("Enter total wickets: ");
scanf("%d", &newPlayer->totalWickets);

(*playerCount)++;
printf("Player added successfully\n");
```

```
}

// Function to update a player's statistics
void updatePlayer(struct Player* players, int playerCount)
{
    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)
        {
            int matches, runs, wickets;
            printf("Enter additional matches played: ");
            scanf("%d", &matches);
            players[i].matchesPlayed += matches;

            printf("Enter additional runs scored: ");
            scanf("%d", &runs);
            players[i].totalRuns += runs;
```

```
            printf("Enter additional wickets taken: ");
            scanf("%d", &wickets);
            players[i].totalWickets += wickets;

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

// Function to display players from a specific team
void displayPlayers(struct Player* players, int playerCount, const char* team)
{
    int found = 0;
    printf("\nPlayers from team %s\n", team);
    for (int i = 0; i < playerCount; i++)
    {
        if (strcmp(players[i].team, team) == 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);
            found = 1;
        }
    }
    if (!found)
        printf("No players found from team %s\n", team);
}

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

    struct Player* highRuns = &players[0];
    struct Player* mostWickets = &players[0];

    for (int i = 1; i < playerCount; i++) {
        if (players[i].totalRuns > highRuns->totalRuns)
```

```
            highRuns = &players[i];
        else if (players[i].totalWickets > mostWickets->totalWickets)
            mostWickets = &players[i];
    }

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

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

Problem 2: Tournament Fixture Scheduler

- Requirements:

  Create a structure Match with members:
  - char team1[30]
  - char team2[30]
  - char date[11] (format: YYYY-MM-DD)
  - char venue[50]
- Functions to:
  - Schedule a new match between two teams.

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

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

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

// Function prototypes
void scheduleMatch(struct Match** matches, int* matchCount, int* capacity);
void displayMatches(struct Match* matches, int matchCount);
void searchMatchesByDate(struct Match* matches, int matchCount, const char* date);
void cancelMatch(struct Match** matches, int* matchCount, const char* team1, const char* team2, const char* date);

int main()
{
```

```
struct Match* matches;
int matchCount = 0;
int capacity = 3;
matches = malloc(capacity * sizeof(struct Match));
if (!matches)
{
   printf("Memory allocation failed!\n");
   return 1;
}

int option;
do
{
   printf("\nTournament Fixture Scheduler\n");
   printf("1. Schedule a new match\n");
   printf("2. Display all matches\n");
   printf("3. Search matches by date\n");
   printf("4. Cancel a match\n");
   printf("5. Exit\n");
   printf("Enter the option: ");
   scanf("%d", &option);

   switch (option)
   {
      case 1: scheduleMatch(&matches, &matchCount, &capacity);
              break;
      case 2: displayMatches(matches, matchCount);
```

```
            break;
        case 3: char date[11];
            printf("Enter date (YYYY-MM-DD): ");
            scanf("%s", date);
            searchMatchesByDate(matches, matchCount, date);
            break;
        case 4: char team1[30], team2[30], date1[11];
            printf("Enter Team 1: ");
            scanf("%s", team1);
            printf("Enter Team 2: ");
            scanf("%s", team2);
            printf("Enter Date (YYYY-MM-DD): ");
            scanf("%s", date1);
            cancelMatch(&matches, &matchCount, team1, team2, date);
            break;
        case 5: printf("Exit the program.\n");
            break;
        default:printf("Invalid option\n");
      }
   } while (option != 5);

   free(matches);
   return 0;
}

// Function to schedule a new match
void scheduleMatch(struct Match** matches, int* matchCount, int* capacity)
```

```
{
    if (*matchCount == *capacity)
    {
        *capacity *= 2;
        struct Match *new = malloc((*capacity) * sizeof(struct Match));
        if (!new)
        {
            printf("Memory allocation failed\n");
            exit(1);
        }
        for (int i = 0; i < *matchCount; i++)
            new[i] = (*matches)[i];

        free(*matches);
        *matches = new;
    }

    struct Match* newMatch = &(*matches)[*matchCount];
    printf("Enter Team 1: ");
    scanf("%s", newMatch->team1);
    printf("Enter Team 2: ");
    scanf("%s", newMatch->team2);
    printf("Enter Date (YYYY-MM-DD): ");
    scanf("%s", newMatch->date);
    printf("Enter Venue: ");
    scanf("%s", newMatch->venue);
```

```
    (*matchCount)++;
    printf("Match scheduled successfully\n");
}

// Function to display all matches
void displayMatches(struct Match* matches, int matchCount)
{
    if (matchCount == 0)
    {
        printf("No matches scheduled\n");
        return;
    }
    printf("\nScheduled matches\n");
    for (int i = 0; i < matchCount; i++)
    {
        printf("Match %d: %s vs %s, Date: %s, Venue: %s\n",
                    i + 1, matches[i].team1, matches[i].team2, matches[i].date,
matches[i].venue);
    }
}

// Function to search matches by date
void searchMatchesByDate(struct Match* matches, int matchCount, const char*
date)
{
    int found = 0;
    printf("\nMatches scheduled on %s\n", date);
    for (int i = 0; i < matchCount; i++)
```

```
    {
        if (strcmp(matches[i].date, date) == 0)
        {
                printf("Match: %s vs %s  Venue: %s\n", matches[i].team1, matches[i].team2, matches[i].venue);
            found = 1;
        }
    }
    if (!found)
        printf("No matches found on %s\n", date);
}

// Function to cancel a match
void cancelMatch(struct Match** matches, int* matchCount, const char* team1, const char* team2, const char* 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");
}
```

Problem 3: Sports Event Medal Tally

- Requirements:

  Define a structure CountryMedalTally with members:
  - char country[30]
  - int gold
  - int silver
  - int bronze
- Functions to:
  - Add a new country's medal tally.
  - Update the medal count for a country.
  - Display the medal tally for all countries.
  - Find and display the country with the highest number of gold medals.
  - Use an array to store the medal tally, and resize the array dynamically as new countries are added.

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

struct CountryMedalTally
{
```

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

// Function prototypes
void addCountry(struct CountryMedalTally** medalTally, int* countryCount, int* capacity);
void updateMedalCount(struct CountryMedalTally* medalTally, int countryCount);
void displayMedalTally(struct CountryMedalTally* medalTally, int countryCount);
void findHighestGold(struct CountryMedalTally* medalTally, int countryCount);

int main()
{
    struct CountryMedalTally* medalTally;
    int countryCount = 0;
    int capacity = 3;
    medalTally = malloc(capacity * sizeof(struct CountryMedalTally));
    if (!medalTally)
```

```
{
    printf("Memory allocation failed\n");
    return 1;
}

int option;
do
{
    printf("\nSports Event Medal Tally\n");
    printf("1. Add a new country's medal tally\n");
    printf("2. Update medal count for a country\n");
    printf("3. Display the medal tally for all countries\n");
    printf("4. Find and display the country with the highest gold medals\n");
    printf("5. Exit\n");
    printf("Enter option: ");
    scanf("%d", &option);

    switch (option)
    {
        case 1: addCountry(&medalTally, &countryCount, &capacity);
                break;
        case 2: updateMedalCount(medalTally, countryCount);
```

```
            break;
        case 3: displayMedalTally(medalTally, countryCount);
            break;
        case 4: findHighestGold(medalTally, countryCount);
            break;
        case 5: printf("Exit the program.\n");
            break;
        default:printf("Invalid option\n");
      }
   } while (option != 5);

   free(medalTally);
   return 0;
}

// Function to add a new country's medal tally
void addCountry(struct CountryMedalTally** medalTally, int* countryCount, int* capacity)
{
   if (*countryCount == *capacity)
   {
      *capacity *= 2;
```

```
        struct CountryMedalTally* new = malloc((*capacity) * sizeof(struct CountryMedalTally));
    if (!new)
    {
        printf("Memory reallocation failed\n");
        exit(1);
    }
    for (int i = 0; i < *countryCount; i++)
        new[i] = (*medalTally)[i];

    free(*medalTally);
    *medalTally = new;
}

struct CountryMedalTally* newCountry = &(*medalTally)[*countryCount];
printf("Enter country name: ");
scanf("%s", newCountry->country);
printf("Enter number of gold medals: ");
scanf("%d", &newCountry->gold);
printf("Enter number of silver medals: ");
scanf("%d", &newCountry->silver);
printf("Enter number of bronze medals: ");
```

```
    scanf("%d", &newCountry->bronze);

    (*countryCount)++;
    printf("Country added successfully\n");
}

// Function to update the medal count for a specific country
void updateMedalCount(struct CountryMedalTally* medalTally, int countryCount)
{
    char country[30];
    printf("Enter the country name to update: ");
    scanf("%s", country);

    for (int i = 0; i < countryCount; i++)
    {
        if (strcmp(medalTally[i].country, country) == 0)
        {
            printf("Enter updated number of gold medals: ");
            scanf("%d", &medalTally[i].gold);
            printf("Enter updated number of silver medals: ");
            scanf("%d", &medalTally[i].silver);
```

```
            printf("Enter updated number of bronze medals: ");
            scanf("%d", &medalTally[i].bronze);
            printf("Medal tally updated successfully!\n");
            return;
        }
    }
    printf("Country not found\n");
}

// Function to display all countries medal tally
void displayMedalTally(struct CountryMedalTally* medalTally, int countryCount)
{
    if (countryCount == 0)
    {
        printf("No countries in the medal tally\n");
        return;
    }

    printf("\nMedal tally\n");
    for (int i = 0; i < countryCount; i++)
    {
```

```
        printf("Country: %s  Gold: %d  Silver: %d  Bronze: %d\n",
                    medalTally[i].country,  medalTally[i].gold,  medalTally[i].silver,
medalTally[i].bronze);
    }
}

// Function to find the country with the highest gold medals
void findHighestGold(struct CountryMedalTally* medalTally, int countryCount)
{
    if (countryCount == 0)
    {
        printf("No countries in the medal tally\n");
        return;
    }
    struct CountryMedalTally* highestGoldCountry = &medalTally[0];
    for (int i = 1; i < countryCount; i++)
    {
        if (medalTally[i].gold > highestGoldCountry->gold)
            highestGoldCountry = &medalTally[i];
    }
    printf("Country with highest gold medals: %s (Gold: %d  Silver: %d  Bronze:
%d)\n",
```

```
        highestGoldCountry->country, highestGoldCountry->gold,
        highestGoldCountry->silver, highestGoldCountry->bronze);
}
```

Problem 4: Athlete Performance Tracker

- Requirements:

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

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

struct Athlete
```

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

// Function prototypes
void addAthlete(struct Athlete** athletes, int* athleteCount, int* capacity);
void updatePerformance(struct Athlete* athletes, int athleteCount);
void displayAthletesBySport(struct Athlete* athletes, int athleteCount, const char* sport);
void displayNewPersonalBestsInLast(struct Athlete* athletes, int athleteCount);

int main()
{
    struct Athlete* athletes;
    int athleteCount = 0;
    int capacity = 3;
    athletes = malloc(capacity * sizeof(struct Athlete));
    if (!athletes)
```

```
    {
        printf("Memory allocation failed\n");
        return 1;
    }

    int option;
    do
    {
        printf("\nAthlete Performance Tracker\n");
        printf("1. Add a new athlete\n");
        printf("2. Update an athlete's last performance\n");
        printf("3. Display all athletes in a specific sport\n");
            printf("4. Display athletes who set a new personal best in their last performance\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);

        switch (option)
        {
            case 1: addAthlete(&athletes, &athleteCount, &capacity);
                    break;
```

```
            case 2: updatePerformance(athletes, athleteCount);
                    break;
            case 3: char sport[30];
                    printf("Enter the sport: ");
                    scanf("%s", sport);
                    displayAthletesBySport(athletes, athleteCount, sport);
                    break;
            case 4: displayNewPersonalBestsInLast(athletes, athleteCount);
                    break;
            case 5: printf("Exit the program.\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);

    free(athletes);
    return 0;
}

// Function to add a new athlete
void addAthlete(struct Athlete** athletes, int* athleteCount, int* capacity)
{
```

```
if (*athleteCount == *capacity)
{
    *capacity *= 2;
    struct Athlete* new = malloc((*capacity) * sizeof(struct Athlete));
    if (!new)
    {
        printf("Memory reallocation failed\n");
        exit(1);
    }
    for (int i = 0; i < *athleteCount; i++)
        new[i] = (*athletes)[i];

    free(*athletes);
    *athletes = new;
}

struct Athlete* newAthlete = &(*athletes)[*athleteCount];
printf("Enter Athlete ID: ");
scanf("%s", newAthlete->athleteID);
printf("Enter Name: ");
scanf("%s", newAthlete->name);
printf("Enter Sport: ");
```

```
    scanf("%s", newAthlete->sport);
    printf("Enter Personal Best: ");
    scanf("%f", &newAthlete->personalBest);
    printf("Enter Last Performance: ");
    scanf("%f", &newAthlete->lastPerformance);

    (*athleteCount)++;
    printf("Athlete added successfully\n");
}

// Function to update an athlete's last performance
void updatePerformance(struct Athlete* athletes, int athleteCount)
{
    char athleteID[10];
    printf("Enter the 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;
                printf("New personal best achieved\n");
            }
            else
                printf("Last performance updated\n");
            return;
        }
    }
    printf("Athlete not found\n");
}

// Function to display all athletes in a specific sport
void displayAthletesBySport(struct Athlete* athletes, int athleteCount, const char* sport)
{
    int found = 0;
    printf("\nAthletes in sport %s\n", 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);
            found = 1;
        }
    }
    if (!found)
        printf("No athletes found in this sport\n");
}

// Function to display athletes who set a new personal best
void displayNewPersonalBestsInLast(struct Athlete* athletes, int athleteCount)
{
    int found = 0;
    printf("\nAthletes who set a new personal best\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);
            found = 1;
        }
    }
    if (!found)
        printf("No athletes set a new personal best\n");
}
```

Problem 5: Sports Equipment Inventory System

- Requirements:

  Define a structure Equipment with members:
  - char equipmentID[10]
  - char name[30]
  - char category[20] (e.g., balls, rackets)
  - int quantity
  - float pricePerUnit
- Functions to:
  - Add new equipment to the inventory.
  - Update the quantity of existing equipment.
  - Display all equipment in a specific category.
  - Calculate the total value of equipment in the inventory.

- Store the inventory data in a dynamically allocated array and ensure proper resizing when needed.

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

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

// Function prototypes
void addEquipment(struct Equipment** inventory, int* count, int* capacity);
void updateQuantity(struct Equipment* inventory, int count);
void displayByCategory(struct Equipment* inventory, int count, const char* category);
void calculateTotalValue(struct Equipment* inventory, int count);
```

```
int main()
{
    struct Equipment* inventory = NULL;
    int count = 0;
    int capacity = 3;
    inventory = malloc(capacity * sizeof(struct Equipment));
    if (!inventory)
    {
        printf("Memory allocation failed\n");
        return 1;
    }

    int option;
    do
    {
        printf("\nSports Equipment Inventory System\n");
        printf("1. Add new equipment\n");
        printf("2. Update quantity of existing equipment\n");
        printf("3. Display all equipment in a specific category\n");
        printf("4. Calculate total value of equipment in inventory\n");
        printf("5. Exit\n");
```

```
    printf("Enter the option: ");
    scanf("%d", &option);

    switch (option)
    {
        case 1: addEquipment(&inventory, &count, &capacity);
                break;
        case 2: updateQuantity(inventory, count);
                break;
        case 3: char category[20];
                printf("Enter category: ");
                scanf("%s", category);
                displayByCategory(inventory, count, category);
                break;
        case 4: calculateTotalValue(inventory, count);
                break;
        case 5: printf("Exiting the program.\n");
                break;
        default:printf("Invalid option\n");
    }
} while (option != 5);
```

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

// Function to add new equipment to the inventory
void addEquipment(struct Equipment** inventory, int* count, int* capacity)
{
    if (*count == *capacity)
    {
        *capacity *= 2;
        struct Equipment* new = malloc((*capacity) * sizeof(struct Equipment));
        if (!new)
        {
            printf("Memory reallocation failed\n");
            exit(1);
        }

        for (int i = 0; i < *count; i++)
            new[i] = (*inventory)[i];

        free(*inventory);
        *inventory = new;
```

```
    }

    struct Equipment* newEquipment = &(*inventory)[*count];
    printf("Enter Equipment ID: ");
    scanf("%s", newEquipment->equipmentID);
    printf("Enter Name: ");
    scanf("%s", newEquipment->name);
    printf("Enter Category: ");
    scanf("%s", newEquipment->category);
    printf("Enter Quantity: ");
    scanf("%d", &newEquipment->quantity);
    printf("Enter Price per Unit: ");
    scanf("%f", &newEquipment->pricePerUnit);

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

// Function to update the quantity of existing equipment
void updateQuantity(struct Equipment* inventory, int count)
{
    char equipmentID[10];
```

```
    printf("Enter the equipment ID to update: ");
    scanf("%s", equipmentID);

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

// Function to display all equipment in a specific category
void displayByCategory(struct Equipment* inventory, int count, const char* category)
{
    int found = 0;
    printf("\nEquipment in category: %s\n", category);
```

```
    for (int i = 0; i < count; 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);
            found = 1;
        }
    }
    if (!found)
        printf("No equipment found in this category\n");
}

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

Problem 1: Research Paper Database Management

Requirements:

- Define a structure ResearchPaper with the following members:
    - char title[100]
    - char author[50]
    - char journal[50]
    - int year
    - char DOI[30]
- Functions to:
- Add a new research paper to the database.
- Update the details of an existing paper using its DOI.
- Display all papers published in a specific journal.
- Find and display the most recent papers published by a specific author.
- Use dynamic memory allocation to store and manage the research papers in an array, resizing it as needed.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

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

// Function prototypes
void addPaper(struct ResearchPaper** papers, int* count, int* capacity);
void updatePaper(struct ResearchPaper* papers, int count, const char* DOI);
void displayPapersByJournal(struct ResearchPaper* papers, int count, const char* journal);
void displayRecentPapersByAuthor(struct ResearchPaper* papers, int count, const char* author);

int main()
{
    struct ResearchPaper* papers;
    int count = 0;
```

```
int capacity = 3;
papers = malloc(capacity * sizeof(struct ResearchPaper));
if (!papers)
{
    printf("Memory allocation failed\n");
    return 1;
}

int option;
do
{
    printf("\nResearch Paper Database Management\n");
    printf("1. Add a new research paper\n");
    printf("2. Update research paper details\n");
    printf("3. Display papers from a specific journal\n");
    printf("4. Display recent papers by a specific author\n");
    printf("5. Exit\n");
    printf("Enter the option: ");
    scanf("%d", &option);

    switch (option)
    {
```

```
        case 1: addPaper(&papers, &count, &capacity);
                break;
        case 2: char DOI[30];
                printf("Enter DOI of the paper to update: ");
                scanf("%s", DOI);
                updatePaper(papers, count, DOI);
                break;
        case 3: char journal[50];
                printf("Enter journal name: ");
                scanf(" %[^\n]s", journal); // To read strings with spaces
                displayPapersByJournal(papers, count, journal);
                break;
        case 4: char author[50];
                printf("Enter author name: ");
                scanf(" %[^\n]s", author);
                displayRecentPapersByAuthor(papers, count, author);
                break;
        case 5: printf("Exit the program\n");
                break;
        default: printf("Invalid option\n");
    }
} while (option != 5);
```

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

// Function to add a new research paper to the database
void addPaper(struct ResearchPaper** papers, int* count, int* capacity)
{
    if (*count == *capacity)
    {
        *capacity *= 2;
            struct ResearchPaper* new = malloc((*capacity) * sizeof(struct ResearchPaper));
        if (!new)
        {
            printf("Memory reallocation failed\n");
            exit(1);
        }
        for (int i = 0; i < *count; i++)
            new[i] = (*papers)[i];

        free(*papers);
```

```
        *papers = new;
    }

    struct ResearchPaper* newPaper = &(*papers)[*count];
    printf("Enter title: ");
    scanf(" %[^\n]s", newPaper->title);
    printf("Enter author: ");
    scanf(" %[^\n]s", newPaper->author);
    printf("Enter journal: ");
    scanf(" %[^\n]s", newPaper->journal);
    printf("Enter year of publication: ");
    scanf("%d", &newPaper->year);
    printf("Enter DOI: ");
    scanf("%s", newPaper->DOI);

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

// Function to update the details of an existing paper using its DOI
void updatePaper(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]s", papers[i].title);
            printf("Enter new author: ");
            scanf(" %[^\n]s", papers[i].author);
            printf("Enter new journal: ");
            scanf(" %[^\n]s", papers[i].journal);
            printf("Enter new year of publication: ");
            scanf("%d", &papers[i].year);
            printf("Details updated successfully\n");
            return;
        }
    }
    printf("Paper with DOI %s not found\n", DOI);
}

// Function to display all papers published in a specific journal
void displayPapersByJournal(struct ResearchPaper* papers, int count, const char* journal)
```

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

// Function to find and display the most recent papers by a specific author
void displayRecentPapersByAuthor(struct ResearchPaper* papers, int count, const char* author)
{
    int found = 0;
    int maxYear = -1;
```

```
    for (int i = 0; i < count; i++)
    {
        if (strcmp(papers[i].author, author) == 0)
        {
            if (papers[i].year > maxYear)
                maxYear = papers[i].year;
            found = 1;
        }
    }

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

```
    else
        printf("No papers found for author %s\n", author);
}
```

Problem 2: Experimental Data Logger

Requirements:

- Create a structure Experiment with members:
    - char experimentID[10]
    - char researcher[50]
    - char startDate[11] (format: YYYY-MM-DD)
    - char endDate[11]
    - float results[10] (store up to 10 result readings)
- Functions to:
- Log a new experiment.
- Update the result readings of an experiment.
- Display all experiments conducted by a specific researcher.
- Calculate and display the average result for a specific experiment.
- Use a dynamically allocated array for storing experiments and manage resizing as more data is logged.

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

```
struct Experiment
{
    char experimentID[10];
    char researcher[50];
    char startDate[11];
    char endDate[11];
    float results[10];
};

// Function prototypes
void logExperiment(struct Experiment** experiments, int* count, int* capacity);
void updateResults(struct Experiment* experiments, int count, const char* experimentID);
void displayExperimentsByResearcher(struct Experiment* experiments, int count, const char* researcher);
void calculateSpecificAverageResult(struct Experiment* experiments, int count, const char* experimentID);

int main()
{
    struct Experiment* experiments;
```

```
int count = 0;
int capacity = 3;
experiments = malloc(capacity * sizeof(struct Experiment));
if (!experiments)
{
    printf("Memory allocation failed\n");
    return 1;
}

int option;
do
{
    printf("\nExperimental Data Logger\n");
    printf("1. Log a new experiment\n");
    printf("2. Update result readings of an experiment\n");
    printf("3. Display experiments by a specific researcher\n");
    printf("4. Calculate average result for an experiment\n");
    printf("5. Exit\n");
    printf("Enter the option: ");
    scanf("%d", &option);

    switch (option)
```

```
{
   case 1: logExperiment(&experiments, &count, &capacity);
           break;
   case 2: char experimentID[10];
           printf("Enter experiment ID: ");
           scanf("%s", experimentID);
           updateResults(experiments, count, experimentID);
           break;
   case 3: char researcher[50];
           printf("Enter researcher name: ");
           scanf(" %[^\n]s", researcher);
           displayExperimentsByResearcher(experiments, count, researcher);
           break;
   case 4: char experimentID1[10];
           printf("Enter experiment ID: ");
           scanf("%s", experimentID1);
           calculateSpecificAverageResult(experiments, count, experimentID);
           break;
   case 5: printf("Exit the program.\n");
           break;
   default:printf("Invalid option\n");
}
```

```
    } while (option != 5);

    free(experiments);
    return 0;
}

// Function to log a new experiment
void logExperiment(struct Experiment** experiments, int* count, int* capacity)
{
    if (*count == *capacity)
    {
        *capacity *= 2;
        struct Experiment* new = malloc((*capacity) * sizeof(struct Experiment));
        if (!new)
        {
            printf("Memory reallocation failed\n");
            exit(1);
        }
        for (int i = 0; i < *count; i++)
            new[i] = (*experiments)[i];

        free(*experiments);
```

```
    *experiments = new;
}

struct Experiment* newExperiment = &(*experiments)[*count];
printf("Enter experiment ID: ");
scanf("%s", newExperiment->experimentID);
printf("Enter researcher name: ");
scanf(" %[^\n]s", newExperiment->researcher);
printf("Enter start date (YYYY-MM-DD): ");
scanf("%s", newExperiment->startDate);
printf("Enter end date (YYYY-MM-DD): ");
scanf("%s", newExperiment->endDate);

printf("Enter up to 10 results (enter -1 to stop):\n");
for (int i = 0; i < 10; i++)
{
    float result;
    printf("Result %d: ", i + 1);
    scanf("%f", &result);
    if (result == -1)
        break;
    newExperiment->results[i] = result;
```

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

// Function to update the result readings of an experiment
void updateResults(struct Experiment* experiments, int count, const char* experimentID)
{
    for (int i = 0; i < count; i++)
    {
        if (strcmp(experiments[i].experimentID, experimentID) == 0)
        {
            printf("Updating results for experiment ID: %s\n", experimentID);
            printf("Enter up to 10 results (enter -1 to stop):\n");
            for (int j = 0; j < 10; j++)
            {
                float result;
                printf("Result %d: ", j + 1);
                scanf("%f", &result);
                if (result == -1)
                    break;
```

```
            experiments[i].results[j] = result;
        }
        printf("Results updated successfully\n");
        return;
    }
  }
  printf("Experiment with ID %s not found\n", experimentID);
}

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

```
            found = 1;
        }
    }
    if (!found)
        printf("No experiments found for researcher %s\n", researcher);
}

// Function to calculate and display the average result for a specific experiment
void calculateSpecificAverageResult(struct Experiment* experiments, int count,
const 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] == 0)
                    break;
                sum += experiments[i].results[j];
```

```
            validResults++;
        }
        if (validResults > 0)
            printf("Average result for experiment ID %s: %.2f\n",
                experimentID, sum / validResults);
        else
            printf("No valid results found for experiment ID %s\n", experimentID);
        return;
      }
   }
   printf("Experiment with ID %s not found\n", experimentID);
}
```

Problem 3: Grant Application Tracker

Requirements:

- Define a structure GrantApplication with the following members:
    - char applicationID[10]
    - char applicantName[50]
    - char projectTitle[100]
    - float requestedAmount
    - char status[20] (e.g., Submitted, Approved, Rejected)

- Functions to:
- Add a new grant application.
- Update the status of an application.
- Display all applications requesting an amount greater than a specified value.
- Find and display applications that are currently "Approved."
- Store the grant applications in a dynamically allocated array, resizing it as necessary.

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

struct GrantApplication
{
    char applicationID[10];
    char applicantName[50];
    char projectTitle[100];
    float requestedAmount;
    char status[20];
};
```

```
// Function prototypes
void addApplication(struct GrantApplication** applications, int* count, int* capacity);
void updateStatus(struct GrantApplication* applications, int count);
void displayApplicationsByAmount(struct GrantApplication* applications, int count, float amount);
void displayApprovedApplications(struct GrantApplication* applications, int count);

int main()
{
    struct GrantApplication* applications;
    int count = 0;
    int capacity = 3;
    applications = malloc(capacity * sizeof(struct GrantApplication));
    if (!applications)
    {
        printf("Memory allocation failed\n");
        return 1;
    }

    int option;
```

```
    do
    {
        printf("\nGrant Application Tracker\n");
        printf("1. Add a new grant application\n");
        printf("2. Update the status of an application\n");
        printf("3. Display all applications requesting amount grater than a specified value\n");
        printf("4. Display approved applications\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);

        switch (option)
        {
            case 1: addApplication(&applications, &count, &capacity);
                    break;
            case 2: updateStatus(applications, count);
                    break;
            case 3: float amount;
                    printf("Enter the amount: ");
                    scanf("%f", &amount);
                    displayApplicationsByAmount(applications, count, amount);
```

```
            break;
        case 4: displayApprovedApplications(applications, count);
            break;
        case 5: printf("Exiting the program.\n");
            break;
        default: printf("Invalid option\n");
      }
   } while (option != 5);
   free(applications);
   return 0;
}

// Add a new grant application
void addApplication(struct GrantApplication** applications, int* count, int* capacity)
{
   if (*count == *capacity)
   {
      *capacity *= 2;
          struct GrantApplication* new = malloc(*capacity * sizeof(struct GrantApplication));
```

```
    if (!new)
    {
        printf("Memory allocation failed\n");
        exit(1);
    }
    for (int i = 0; i < *count; i++)
        new[i] = (*applications)[i];
    free(*applications);
    *applications = new;
}

struct GrantApplication* newApplication = &(*applications)[*count];
printf("Enter application ID: ");
scanf("%s", newApplication->applicationID);

printf("Enter applicant name: ");
scanf(" %[^\n]", newApplication->applicantName);

printf("Enter project title: ");
scanf(" %[^\n]", newApplication->projectTitle);

printf("Enter requested amount: ");
```

```
    scanf("%f", newApplication->requestedAmount);

    printf("Enter status (Submitted/Approved/Rejected): ");
    scanf("%s", newApplication->status);

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

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

    for (int i = 0; i < count; i++)
    {
        if (strcmp(applications[i].applicationID, id) == 0)
        {
            printf("Enter the new status (Submitted/Approved/Rejected): ");
            scanf("%s", applications[i].status);
```

```
            printf("Application status updated successfully\n");
            return;
        }
    }
    printf("Application with ID %s not found\n", id);
}

// Display applications requesting an amount greater than a specified value
void displayApplicationsByAmount(struct GrantApplication* applications, int count, float amount)
{
    int found = 0;
    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  Amount: %.2f  Status: %s\n",
                applications[i].applicationID, applications[i].applicantName,
                applications[i].projectTitle, applications[i].requestedAmount,
                applications[i].status);
            found = 1;
```

```
        }
    }
    if (!found)
        printf("No applications found requesting more than %.2f\n", amount);
}

// Display all approved applications
void displayApprovedApplications(struct GrantApplication* applications, int count)
{
    int found = 0;
    printf("Approved applications\n");
    for (int i = 0; i < count; i++)
    {
        if (strcmp(applications[i].status, "Approved") == 0)
        {
            printf("ID: %s  Applicant: %s  Project: %s  Amount: %.2f\n",
                applications[i].applicationID, applications[i].applicantName,
                applications[i].projectTitle, applications[i].requestedAmount);
            found = 1;
        }
    }
```

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

Problem 4: Research Collaborator Management

Requirements:

- Create a structure Collaborator with members:
    - char collaboratorID[10]
    - char name[50]
    - char institution[50]
    - char expertiseArea[30]
    - int numberOfProjects
- Functions to:
- Add a new collaborator to the database.
- Update the number of projects a collaborator is involved in.
- Display all collaborators from a specific institution.
- Find collaborators with expertise in a given area.
- Use dynamic memory allocation to manage the list of collaborators, allowing for expansion as more are added.

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

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

// Function Prototypes
void addCollaborator(struct Collaborator** collaborators, int* collaboratorCount, int* capacity);
void updateCollaboratorProjects(struct Collaborator* collaborators, int collaboratorCount, const char* collaboratorID);
void displayCollaboratorsByInstitution(struct Collaborator* collaborators, int collaboratorCount, const char* institution);
void findCollaboratorsByExpertiseInArea(struct Collaborator* collaborators, int collaboratorCount, const char* expertiseArea);

int main()
{
    struct Collaborator* collaborators = malloc(10 * sizeof(struct Collaborator));
```

```
if (collaborators == 0)
{
    printf("Memory allocation failed\n");
    exit(1);
}

int collaboratorCount = 0;
int capacity = 10;

int option;
do
{
    printf("\nResearch Collaborator Management System\n");
    printf("1. Add new collaborator\n");
    printf("2. Update number of projects collaborator is involved\n");
    printf("3. Display all collaborators from a specific institution\n");
    printf("4. Find collaborators with expertise in given area\n");
    printf("5. Exit\n");
    printf("Enter the option: ");
    scanf("%d", &option);

    switch (option)
```

```
{
  case 1: addCollaborator(&collaborators, &collaboratorCount, &capacity);
          break;
  case 2: char collaboratorID[10];
          printf("Enter Collaborator ID to update projects: ");
          scanf("%s", collaboratorID);
                updateCollaboratorProjects(collaborators,  collaboratorCount, collaboratorID);
          break;
  case 3: char institution[50];
          printf("Enter institution name: ");
          scanf(" %[^\n]", institution);
          displayCollaboratorsByInstitution(collaborators, collaboratorCount, institution);
          break;
  case 4: char expertiseArea[30];
          printf("Enter expertise area: ");
          scanf(" %[^\n]", expertiseArea);
                      findCollaboratorsByExpertiseInArea(collaborators, collaboratorCount, expertiseArea);
          break;
  case 5: printf("Exit the program...\n");
```

```
                break;
            default: printf("Invalid choice\n");
        }
    } while (option != 5);
    free(collaborators);
    return 0;
}

// Function to add a new collaborator
void addCollaborator(struct Collaborator** collaborators, int* collaboratorCount, int* capacity)
{
    if (*collaboratorCount == *capacity)
    {
        *capacity *= 2;
        struct Collaborator* new = malloc(*capacity * sizeof(struct Collaborator));
        if (new == 0)
        {
            printf("Memory allocation failed\n");
            exit(1);
        }
        for (int i = 0; i < *collaboratorCount; i++)
```

```
            new[i] = (*collaborators)[i];
        free(*collaborators);
        *collaborators = new;
    }

                struct Collaborator* newCollaborator =
&(*collaborators)[*collaboratorCount];
    printf("Enter Collaborator ID: ");
    scanf("%s", newCollaborator->collaboratorID);

    printf("Enter Name: ");
    scanf(" %[^\n]", newCollaborator->name);

    printf("Enter Institution: ");
    scanf(" %[^\n]", newCollaborator->institution);

    printf("Enter Expertise Area: ");
    scanf(" %[^\n]", newCollaborator->expertiseArea);

    printf("Enter Number of Projects: ");
    scanf("%d", newCollaborator->numberOfProjects);
```

```
    (*collaboratorCount)++;
    printf("Collaborator added successfully\n");
}

// Function to update the number of projects
void updateCollaboratorProjects(struct Collaborator* collaborators, int collaboratorCount, const char* collaboratorID)
{
    for (int i = 0; i < collaboratorCount; i++)
    {
        if (strcmp(collaborators[i].collaboratorID, collaboratorID) == 0) {
                printf("Enter new number of projects for collaborator %s: ", collaboratorID);
            scanf("%d", &collaborators[i].numberOfProjects);
            printf("Projects updated successfully\n");
            return;
        }
    }
    printf("Collaborator not found\n");
}

// Function to display collaborators from a specific institution
```

```
void displayCollaboratorsByInstitution(struct Collaborator* collaborators, int collaboratorCount, const char* institution)
{
    int found = 0;
    for (int i = 0; i < collaboratorCount; i++)
    {
        if (strcmp(collaborators[i].institution, institution) == 0)
        {
            printf("Collaborator ID: %s  Name: %s  Expertise: %s  Projects: %d\n",
                   collaborators[i].collaboratorID, collaborators[i].name,
                   collaborators[i].expertiseArea, collaborators[i].numberOfProjects);
            found = 1;
        }
    }
    if (!found)
        printf("No collaborators found from institution %s\n", institution);
}

// Function to find collaborators with expertise in a given area
void findCollaboratorsByExpertiseInArea(struct Collaborator* collaborators, int collaboratorCount, const char* expertiseArea)
{
```

```
    int found = 0;
    for (int i = 0; i < collaboratorCount; i++)
    {
        if (strcmp(collaborators[i].expertiseArea, expertiseArea) == 0)
        {
            printf("Collaborator ID: %s  Name: %s  Institution: %s  Projects: %d\n",
                    collaborators[i].collaboratorID, collaborators[i].name,
                    collaborators[i].institution, collaborators[i].numberOfProjects);
            found = 1;
        }
    }
    if (!found)
        printf("No collaborators found with expertise in %s\n", expertiseArea);
}
```

Problem 5: Scientific Conference Submission Tracker

Requirements:

- Define a structure ConferenceSubmission with the following:
    - char submissionID[10]
    - char authorName[50]
    - char paperTitle[100]

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

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

struct ConferenceSubmission
{
   char submissionID[10];
   char authorName[50];
   char paperTitle[100];
   char conferenceName[50];
   char submissionDate[11];
```

```
    char status[20];
};

// Function prototypes
void addSubmission(struct ConferenceSubmission** submissions, int* submissionCount, int* capacity);
void updateSubmissionStatus(struct ConferenceSubmission* submissions, int submissionCount, const char* submissionID);
void displaySubmissionsByConference(struct ConferenceSubmission* submissions, int submissionCount, const char* conferenceName);
void findSubmissionsByAuthor(struct ConferenceSubmission* submissions, int submissionCount, const char* authorName);

int main()
{
    struct ConferenceSubmission* submissions = malloc(10 * sizeof(struct ConferenceSubmission));  // Initial allocation for 10 submissions
    if (submissions == 0)
    {
        printf("Memory allocation failed\n");
        exit(1);
    }
```

```
int submissionCount = 0;
int capacity = 10;

int option;
do
{
   printf("\nScientific Conference Submission Tracker\n");
   printf("1. Add new submission\n");
   printf("2. Update the status of submission\n");
   printf("3. Display all submissions to a apecific conference\n");
   printf("4. Find and display submissions by a specific author\n");
   printf("5. Exit\n");
   printf("Enter the option: ");
   scanf("%d", &option);

   switch (option)
   {
      case 1: addSubmission(&submissions, &submissionCount, &capacity);
            break;
      case 2: char submissionID[10];
            printf("Enter submission ID to update status: ");
            scanf("%s", submissionID);
```

```
                updateSubmissionStatus(submissions, submissionCount, submissionID);
                break;
            case 3: char conferenceName[50];
                printf("Enter conference name: ");
                scanf(" %[^\n]", conferenceName);
                displaySubmissionsByConference(submissions, submissionCount, conferenceName);
                break;
            case 4: char authorName[50];
                printf("Enter author name: ");
                scanf(" %[^\n]", authorName);
                findSubmissionsByAuthor(submissions, submissionCount, authorName);
                break;
            case 5: printf("Exiting the program...\n");
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);
    free(submissions);
    return 0;
```

```
}

// Function to add a new conference submission
void addSubmission(struct ConferenceSubmission** submissions, int* submissionCount, int* capacity)
{
    if (*submissionCount == *capacity)
    {
        *capacity *= 2;
            struct ConferenceSubmission* new = malloc(*capacity * sizeof(struct ConferenceSubmission));
        if (new == 0)
        {
            printf("Memory allocation failed\n");
            exit(1);
        }
        for (int i = 0; i < *submissionCount; i++)
            new[i] = (*submissions)[i];
        free(*submissions);
        *submissions = new;
    }
```

```
    struct ConferenceSubmission* newSubmission =
&(*submissions)[*submissionCount];
    printf("Enter submission ID: ");
    scanf("%s", newSubmission->submissionID);

    printf("Enter author name: ");
    scanf(" %[^\n]", newSubmission->authorName);

    printf("Enter paper title: ");
    scanf(" %[^\n]", newSubmission->paperTitle);

    printf("Enter conference name: ");
    scanf(" %[^\n]", newSubmission->conferenceName);

    printf("Enter submission date (YYYY-MM-DD): ");
    scanf("%s", newSubmission->submissionDate);

    printf("Enter status (Pending/Accepted/Rejected): ");
    scanf("%s", newSubmission->status);

    (*submissionCount)++;
    printf("Submission added successfully\n");
```

```
}

// Function to update the status of a submission
void updateSubmissionStatus(struct ConferenceSubmission* submissions, int submissionCount, const char* submissionID)
{
    for (int i = 0; i < submissionCount; i++)
    {
        if (strcmp(submissions[i].submissionID, submissionID) == 0)
        {
            printf("Enter new status (Pending/Accepted/Rejected): ");
            scanf("%s", submissions[i].status);
            printf("Status updated successfully\n");
            return;
        }
    }
    printf("Submission not found\n");
}

// Function to display all submissions to a specific conference
void displaySubmissionsByConference(struct ConferenceSubmission* submissions, int submissionCount, const char* conferenceName)
```

```
{
    int found = 0;
    for (int i = 0; i < submissionCount; i++)
    {
        if (strcmp(submissions[i].conferenceName, conferenceName) == 0)
        {
            printf("Submission ID: %s  Author: %s  Paper Title: %s  Date: %s  Status: %s\n",
                   submissions[i].submissionID, submissions[i].authorName,
                                submissions[i].paperTitle,   submissions[i].submissionDate, submissions[i].status);
            found = 1;
        }
    }
    if (!found)
        printf("No submissions found for conference %s\n", conferenceName);
}

// Function to find and display submissions by a specific author
void findSubmissionsByAuthor(struct ConferenceSubmission* submissions, int submissionCount, const char* authorName)
{
```

```
    int found = 0;
    for (int i = 0; i < submissionCount; i++)
    {
        if (strcmp(submissions[i].authorName, authorName) == 0)
        {
            printf("Submission ID: %s  Paper Title: %s  Conference: %s  Date: %s  Status: %s\n",
                submissions[i].submissionID, submissions[i].paperTitle,
                submissions[i].conferenceName, submissions[i].submissionDate, submissions[i].status);
            found = 1;
        }
    }
    if (!found)
        printf("No submissions found by author %s\n", authorName);
}
```