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

};

void yearSearch(struct Vehicle *vehiclePtr,int n);

void highFuelEfficiency(struct Vehicle *vehiclePtr,int n);



int main()

{

   int n;

   printf("Enter number of vehicles:");
   scanf("%d",&n);

   struct Vehicle *vehicles=(struct Vehicle *)malloc(n*sizeof(struct Vehicle));
   for(int i=0;i<n;i++){
      printf("Details of Vehicle: %d\n",i+1);
      printf("Registration Number:");
      scanf("%s",vehicles[i].registrationNumber);
      printf("Model:");
      scanf("%s",vehicles[i].model);
      printf("Year Of Manufacture:");
      scanf("%d",&vehicles[i].yearOfManufactur
      e); printf("Mileage:");
      scanf("%f",&vehicles[i].mileage);
      printf("fuelEfficiency:");
      scanf("%f",&vehicles[i].fuelEfficiency);
   }

   yearSearch(vehicles,n);
   highFuelEfficiency(vehicles,n);


   return 0;

}



void yearSearch(struct Vehicle *vehiclePtr,int
   n){ for(int i=0;i<n;i++){
      if(vehiclePtr[i].yearOfManufacture>2015)

      printf("%s manufactured after 2015\n",vehiclePtr[i].registrationNumber);
   }



}

void highFuelEfficiency(struct Vehicle *vehiclePtr,int
   n){ float highest=vehiclePtr[0].fuelEfficiency;
   int j;

   for(int i=0;i<n;i++){
     if(vehiclePtr[i].fuelEfficiency>highest
     )
     highest=vehiclePtr[i].fuelEfficiency;
     j=i;
   }

   printf("%s has highest fuel
efficiency(%0.2f)\n",vehiclePtr[j].registrationNumber,highest);

}
```

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;
};



void bookCar(struct CarRental* rentals, int* size, int
capacity); void displayRentals(struct CarRental* rentals, int
size);
void searchByCustomerName(struct CarRental* rentals, int
size); void calculateTotalRentalPrice(struct CarRental*
rentals, int size); int calculateRentalDays(const char* start,
const char* end);
int validateDate(const char* date);



int main() {
   int size =
   0;
   int capacity = 10;

   struct CarRental* rentals = malloc(capacity * sizeof(struct CarRental));



   if (rentals == NULL) {

      printf("Memory allocation failed.\n");
      return 1;

   }



   int
   choice;
   do {
      printf("\n=== Car Rental Reservation System ===\n");
      printf("1. Book a Car\n");
      printf("2. Display All Rentals\n");

      printf("3. Search Rentals by Customer
      Name\n"); printf("4. Calculate Total Rental
      Price\n"); printf("5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);


      switch (choice) {
         case 1:
            bookCar(rentals, &size, capacity);
            break;
         case 2:

            displayRentals(rentals,
            size); break;
         case 3:

            searchByCustomerName(rentals,
            size); break;
         case 4:
            calculateTotalRentalPrice(rentals, size);
            break;
        case 5:

            printf("Exiting program.\n");
            break;
        default:

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

        }

    } while (choice != 5);



    free(rentals);
    return 0;
}





void bookCar(struct CarRental* rentals, int* size, int
    capacity) { if (*size == capacity) {
        printf("No more rentals can be booked. Capacity full.\n");
        return;
    }



    printf("\nEnter details for Rental %d:\n", *size + 1);
    printf("Car ID: ");
    scanf("%s",
    rentals[*size].carID);
    printf("Customer Name: ");
    scanf(" %[^\n]", rentals[*size].customerName);
    printf("Rental Date (YYYY-MM-DD): ");
    scanf("%s", rentals[*size].rentalDate);
    if (!validateDate(rentals[*size].rentalDate))
        { printf("Invalid rental date format.\n");
        return;
    }

    printf("Return Date (YYYY-MM-DD): ");
    scanf("%s", rentals[*size].returnDate);
    if (!validateDate(rentals[*size].returnDate))
        { printf("Invalid return date format.\n");
        return;
    }

    printf("Rental Price Per Day: ");

    scanf("%f", &rentals[*size].rentalPricePerDay);



    (*size)++;

    printf("Car rental booked successfully.\n");

}



void displayRentals(struct CarRental* rentals, int
    size) { if (size == 0) {
        printf("No rentals found.\n");
        return;
    }
    printf("\n=== Current Rentals ===\n");
    for (int i = 0; i < size; i++) {
        printf("Car ID: %s\n", rentals[i].carID);

        printf("Customer Name: %s\n",
        rentals[i].customerName); printf("Rental Date: %s\n",
        rentals[i].rentalDate); printf("Return Date: %s\n",
        rentals[i].returnDate);
        printf("Rental Price Per Day: %.2f\n\n", rentals[i].rentalPricePerDay);

    }

}





void searchByCustomerName(struct CarRental* rentals, int
    size) { if (size == 0) {
        printf("No rentals found.\n");
        return;
    }



    char name[50];

    printf("Enter Customer Name to search: ");
    scanf(" %[^\n]", name);


    int found = 0;

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

        if (strcmp(rentals[i].customerName, name) == 0) {
            printf("\nRental Found:\n");
            printf("Car ID: %s\n",
            rentals[i].carID);
            printf("Rental Date: %s\n",
            rentals[i].rentalDate); printf("Return Date:
            %s\n", rentals[i].returnDate);
            printf("Rental Price Per Day: %.2f\n",
            rentals[i].rentalPricePerDay); found = 1;
        }

    }



    if (!found) {

        printf("No rentals found for the customer.\n");

    }

}





void calculateTotalRentalPrice(struct CarRental* rentals, int
    size) { if (size == 0) {
        printf("No rentals found.\n");
        return;
    }



    char name[50];

    printf("Enter Customer Name to calculate total
    price: "); scanf(" %[^\n]", name);
    float totalPrice = 0;
    int found = 0;
    for (int i = 0; i < size; i++) {

        if (strcmp(rentals[i].customerName, name) == 0) {

            int days = calculateRentalDays(rentals[i].rentalDate,
            rentals[i].returnDate); if (days >= 0) {
                totalPrice += days * rentals[i].rentalPricePerDay;
                found = 1;
            } else {

                printf("Invalid date range for rental with Car ID: %s\n", rentals[i].carID);

            }

        }

    }



    if (found) {

        printf("Total Rental Price for %s: %.2f\n", name, totalPrice);

    } else {

        printf("No rentals found for the customer.\n");

    }

}



int calculateRentalDays(const char* start, const char*
    end) { int startYear, startMonth, startDay;
    int endYear, endMonth, endDay;
    sscanf(start, "%d-%d-%d", &startYear, &startMonth, &startDay);
    sscanf(end, "%d-%d-%d", &endYear, &endMonth, &endDay);




    int startTotalDays = startYear * 365 + startMonth * 30 +
    startDay; int endTotalDays = endYear * 365 + endMonth * 30
    + endDay;


    return endTotalDays - startTotalDays;

}



int validateDate(const char* date) {

    if (strlen(date) != 10 || date[4] != '-' || date[7] != '-') {
        return 0;
    }

    for (int i = 0; i < 10; i++) {

        if ((i == 4 || i == 7) && date[i] == '-')
        continue; if (date[i] < '0' || date[i] > '9')
        return 0;
    }

    return 1;

}
```

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
     > struct
 SensorData{ int
    sensorID;
        char timestamp[20];
        float speed;
        float latitude;
        float
        longitude;


};




void displayWithinTimerange(struct SensorData *dataptr,int
n); int compareTimestamps(char *ts1,char *ts2);
void findMaxSpeed(struct SensorData *dataptr,int n);

void calculateAverageSpeed(struct SensorData
*dataptr,int n); int main(){
   int n;

   printf("Enter the size:");
   scanf("%d",&n);
   struct SensorData *data=(struct SensorData *)malloc(n*sizeof(struct
   SensorData));
```

```
    if(data==NULL){

        printf("Memory allocation
        failed."); return 1;
    }



    printf("Log New Sensor Data\n");
    for(int i=0;i<n;i++){
        printf("Sensor %d\n",i+1);
        printf("SensorID:");
        scanf("%d",&data[i].sensorID);
        printf("TimeStamp:");
        scanf(" %[^\n]s",data[i].timestamp);
        printf("Speed:");
        scanf("%f",&data[i].speed);
        printf("Lattitude:");
        scanf("%f",&data[i].latitude);
        printf("Longitude:");
        scanf("%f",&data[i].longitude);


    }

    displayWithinTimerange(data,
    n); findMaxSpeed(data,n);
    calculateAverageSpeed(data,n
    ); return 0;
}
```

```
void findMaxSpeed(struct SensorData *dataptr,int
   n){ float maxSpeed=dataptr[0].speed;
   for(int i=0;i<n;i++){
      if(dataptr[i].speed>maxSpeed){
         maxSpeed=dataptr[i].speed;

      }

   }

   printf("Maximum speed=%0.2f\n",maxSpeed);

}

void calculateAverageSpeed(struct SensorData
   *dataptr,int n){ char start[20],end[20];
   printf("Enter starting
   time:"); scanf("
   %[^\n]s",start); printf("Enter
   Ending time:"); scanf("
   %[^\n]s",end);
   float
   totalSpeed; int
   count;
   for(int i=0;i<n;i++){
      if(compareTimestamps(dataptr[i].timestamp,start)>=0 &&
      compareTimestamps(dataptr[i].timestamp,end)<=0){


      totalSpeed+=dataptr[i].speed
      ; count++;
      }
```

```
    }

    if(count>0){

        printf("Average Speed =%0.2f\n",totalSpeed/count);

    }

    else

    printf("No record found within the time range\n");

}




void displayWithinTimerange(struct SensorData *dataptr,int
    n){ char start[20],end[20];
    int found=0;

    printf("Enter Time Range.\nStart time:");
    scanf(" %[^\n]s",start);
    printf("End Time: ");

    scanf("
    %[^\n]s",end);
    for(int i=0;i<n;i++){
        if(compareTimestamps(dataptr[i].timestamp,start)>=0 &&
        compareTimestamps(dataptr[i].timestamp,end)<=0){
            printf("SensorID: %d\nTime Stamp:
%s\nSpeed:%0.2f\nLattitude:%0.2f\nLongitude:%0.2f\n",


dataptr[i].sensorID,dataptr[i].timestamp,dataptr[i].speed,dataptr[i].latitude,datapt
r[i].l ongitude);

            found=1;

        }

    }
    if(!found){

        printf("No data found\n");

    }



}

int compareTimestamps(char *ts1,char
    *ts2){ return strcmp(ts1,ts2);
}
```

Problem 4: Engine Performance Monitoring
System Requirements:

- Define a structure EnginePerformance with members:
    - char engineID[10]
    - float temperature
    - float rpm
    - float fuelConsumptionRate
    - float oilPressure
- Functions to:
- Add performance data for a specific engine.
- Display all performance data for a specific engine ID.
- Calculate the average temperature and RPM for a specific engine.
- Identify any engine with abnormal oil pressure (above or below specified thresholds).
- Use linked lists to store and manage performance data entries. #include <stdio.h>

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


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

    float fuelConsumptionRate;
    float oilPressure;
};



void addPerformanceData(struct EnginePerformance* data, int* size, int
capacity); void displayPerformanceData(struct EnginePerformance* data, int
size);
void calculateAverageTempAndRPM(struct EnginePerformance* data, int
size); void identifyAbnormalOilPressure(struct EnginePerformance* data, int
size);


int main() {

    int capacity = 10;
    int size = 0;
    struct EnginePerformance* data = malloc(capacity * sizeof(struct
EnginePerformance));



    if (data == NULL) {

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



    int
    choice;
    do {
        printf("\n=== Engine Performance Monitoring System
        ===\n"); printf("1. Add Performance Data\n");
        printf("2. Display Performance Data for Specific Engine ID\n");
printf("3. Calculate Average Temperature and RPM for Specific Engine
ID\n"); printf("4. Identify Engines with Abnormal Oil Pressure\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);


switch (choice) {
   case 1:
      addPerformanceData(data, &size, capacity);
      break;
   case 2:

      displayPerformanceData(data,
      size); break;
   case 3:

      calculateAverageTempAndRPM(data, size);
      break;
   case 4:

      identifyAbnormalOilPressure(data,
      size); break;
   case 5:

      printf("Exiting program.\n");
      break;
   default:

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

}
```

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



    free(data
    ); return
    0;
}



void addPerformanceData(struct EnginePerformance* data, int* size, int
    capacity) { if (*size == capacity) {
        printf("No more entries can be added. Capacity
        full.\n"); return;
    }



    printf("\nEnter details for Entry %d:\n", *size + 1);
    printf("Engine ID: ");
    scanf("%s", data[*size].engineID);
    printf("Temperature: ");
    scanf("%f",
    &data[*size].temperature);
    printf("RPM: ");
    scanf("%f", &data[*size].rpm);
    printf("Fuel Consumption Rate: ");
    scanf("%f",
    &data[*size].fuelConsumptionRate);
    printf("Oil Pressure: ");
    scanf("%f", &data[*size].oilPressure);



    (*size)++;
    printf("Performance data added successfully.\n");

}




void displayPerformanceData(struct EnginePerformance* data, int
    size) { if (size == 0) {
        printf("No data
        found.\n"); return;
    }



    char engineID[10];

    printf("Enter Engine ID to display data: ");
    scanf("%s", engineID);


    int found = 0;

    printf("\n=== Performance Data for Engine ID: %s ===\n",
    engineID); for (int i = 0; i < size; i++) {
        if (strcmp(data[i].engineID, engineID) == 0) {
            printf("Temperature: %.2f\n",
            data[i].temperature); printf("RPM: %.2f\n",
            data[i].rpm);
            printf("Fuel Consumption Rate: %.2f\n", data[i].fuelConsumptionRate);
            printf("Oil Pressure: %.2f\n", data[i].oilPressure);
            found = 1;

        }

    }
    if (!found) {

        printf("No data found for Engine ID: %s\n", engineID);

    }

}



void calculateAverageTempAndRPM(struct EnginePerformance* data, int
    size) { if (size == 0) {
        printf("No data
        found.\n"); return;
    }



    char engineID[10];

    printf("Enter Engine ID to calculate
    averages: "); scanf("%s", engineID);


    float totalTemp = 0, totalRPM =
    0; int count = 0;


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

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

        }
    }



    if (count > 0) {

        printf("\nAverage Temperature for Engine ID %s: %.2f\n", engineID,
        totalTemp
/ count);

        printf("Average RPM for Engine ID %s: %.2f\n", engineID, totalRPM /
        count);

    } else {

        printf("No data found for Engine ID: %s\n", engineID);

    }

}




void identifyAbnormalOilPressure(struct EnginePerformance* data, int
    size) { if (size == 0) {
        printf("No data
        found.\n"); return;
    }



    float lowThreshold, highThreshold;
    printf("Enter low oil pressure threshold:
    "); scanf("%f", &lowThreshold);
    printf("Enter high oil pressure threshold: ");
    scanf("%f", &highThreshold);


    int found = 0;

    printf("\n=== Engines with Abnormal Oil Pressure ===\n");
```

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

                if (data[i].oilPressure < lowThreshold || data[i].oilPressure >
                    highThreshold) { printf("Engine ID: %s\n", data[i].engineID);
                    printf("Oil Pressure: %.2f\n",
                    data[i].oilPressure); found = 1;
                }

            }



            if (!found) {

                printf("No engines found 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<string.h>


struct
   ServiceRecord {
   char
   serviceID[10];
   char
   vehicleID[15];
```

```
    char
    serviceDate[11];
    char
    description[100];
    float serviceCost;
};


void searchRecord(struct ServiceRecord records[], int n);


int main() {
    int n;
    printf("Enter number of new services:
    "); scanf("%d", &n);


    struct ServiceRecord records[n];
    printf("Add New Service Record\n");


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

        printf("Record of Vehicle %d\n", i +
        1); printf("Service ID: ");
        scanf("%s",
        records[i].serviceID);
        printf("Vehicle ID: ");
        scanf("%s",
        records[i].vehicleID);
        printf("Service Date (YYYY-MM-DD): ");
        scanf("%s", records[i].serviceDate);
        printf("Description: ");
        scanf(" %[^\n]%*c", records[i].description);
        printf("Service Cost: ");
        scanf("%f", &records[i].serviceCost);

    }


    searchRecord(records,
    n); return 0;
```

```
}




void searchRecord(struct ServiceRecord records[], int
  n) { char vehicleID_1[20];
  printf("Enter vehicle ID to get records: ");
  scanf("%s", vehicleID_1);


  int found = 0;

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

    if (strcmp(records[i].vehicleID, vehicleID_1) == 0) {
      printf("Service ID: %s\n", records[i].serviceID);
      printf("Vehicle ID: %s\n", records[i].vehicleID);
      printf("Service Date: %s\n", records[i].serviceDate);
      printf("Description: %s\n", records[i].description);
      printf("Service Cost: %.2f\n", records[i].serviceCost);
      found = 1;
    }

  }



  if (!found) {

    printf("No records found for vehicle ID: %s\n", vehicleID_1);

  }

}

  if (!found) {

    printf("No records found for the given vehicle ID.\n");

  }

}


float calculateTotalServiceCost(struct ServiceRecord records[], int n, const char
  vehicleID[]) { float totalCost = 0;
```

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

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

    }

    return totalCost;

}


void sortServiceRecordsByDate(struct ServiceRecord records[],
    int n) { for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {

            if (compareDates(records[i].serviceDate, records[j].serviceDate)
                > 0) { struct ServiceRecord temp = records[i];
                records[i] = records[j];
                records[j] = temp;
            }

        }

    }

}


int compareDates(const char date1[], const char date2[]) {
    return strcmp(date1, date2);
}
```

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 **playerData, int *n, int
*capacity); void updatePlayerStats(struct Player
*playerData, int n);
void displayPlayersFromTeam(struct Player *playerData, int
n); void findTopPlayers(struct Player *playerData, int n);


int main() {

   int n = 0; // Current number of players

   int capacity = 10; // Initial capacity of the array
```

```
    struct Player *playerData = (struct Player *)malloc(capacity * sizeof(struct
Player));

    if (playerData == NULL) {
        printf("Memory Allocation Failed\n");
        return 1;
    }



    int
    choice;
    do {
        printf("\n--- Player Management System ---\n");
        printf("1. Add new player\n");
        printf("2. Update player statistics\n");

        printf("3. Display players from a specific team\n");

        printf("4. Find top players (highest runs and most
        wickets)\n"); printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                addPlayer(&playerData, &n,
                &capacity); break;
            case 2:

                updatePlayerStats(playerData
                , n); break;
            case 3:
```

```
            displayPlayersFromTeam(playerData
            , n); break;
         case 4:

            findTopPlayers(playerData,
            n); break;
         case 5:

            printf("Exiting...\n");
            break;
         default:

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

      }

   } while (choice != 5);



   free(playerData);
   return 0;
}



// Function to add a new player

void addPlayer(struct Player **playerData, int *n, int
   *capacity) { if (*n == *capacity) {
      // Double the capacity

      *capacity *= 2;

      *playerData = realloc(*playerData, *capacity * sizeof(struct
      Player)); if (*playerData == NULL) {
         printf("Memory reallocation failed.\n");
```

```
            exit(1);

        }

        printf("Capacity increased to %d.\n", *capacity);

    }



    printf("\nEnter Player Details:\n");
    printf("Name: ");
    scanf(" %[^\n]s", (*playerData)[*n].name);
    printf("Age: ");
    scanf("%d",
    &(*playerData)[*n].age);
    printf("Team: ");
    scanf(" %[^\n]s", (*playerData)[*n].team);
    printf("Matches Played: ");
    scanf("%d",
    &(*playerData)[*n].matchesPlayed);
    printf("Total Runs: ");
    scanf("%d", &(*playerData)[*n].totalRuns);
    printf("Total Wickets: ");
    scanf("%d", &(*playerData)[*n].totalWickets);



    (*n)++;

    printf("Player added successfully!\n");

}



// Function to update a player's statistics

void updatePlayerStats(struct Player *playerData, int n) {
     if (n == 0) {

        printf("No players to evaluate.\n");
        return;
    }

    char name[50];

    printf("\nEnter the name of the player to update: ");
    scanf(" %[^\n]s", name);


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

        if (strcmp(playerData[i].name, name) == 0) {
            printf("\nUpdating statistics for %s:\n",
            playerData[i].name); printf("Matches Played: ");
            scanf("%d",
            &playerData[i].matchesPlayed);
            printf("Total Runs: ");
            scanf("%d", &playerData[i].totalRuns);
            printf("Total Wickets: ");
            scanf("%d", &playerData[i].totalWickets);
            printf("Player statistics updated
            successfully!\n"); return;
        }

    }

    printf("Player not found.\n");

}



// Function to display players from a specific team
```

```
void displayPlayersFromTeam(struct Player *playerData, int
    n) { if (n == 0) {
      printf("No players to evaluate.\n");
      return;
   }

   char team[30];

   printf("\nEnter the team name: ");
   scanf(" %[^\n]s", team);


   printf("\nPlayers from team %s:\n",
   team); int found = 0;
   for (int i = 0; i < n; i++) {

      if (strcmp(playerData[i].team, team) == 0) {
         printf("\nName: %s\n", playerData[i].name);
         printf("Age: %d\n", playerData[i].age);
         printf("Matches Played: %d\n",
         playerData[i].matchesPlayed); printf("Total Runs: %d\n",
         playerData[i].totalRuns); printf("Total Wickets: %d\n",
         playerData[i].totalWickets); found = 1;
      }

   }

   if (!found) {

      printf("No players found in team %s.\n", team);

   }

}
```

```
// Function to find the top players

void findTopPlayers(struct Player *playerData, int
   n) { if (n == 0) {
      printf("No players to evaluate.\n");
      return;
   }



   int maxRuns = 0, maxWickets =
   0; int runsIndex = 0, wicketsIndex
   = 0;


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

      if (playerData[i].totalRuns >
         playerData[runsIndex].totalRuns) { runsIndex = i;
      }

      if (playerData[i].totalWickets > playerData[wicketsIndex].totalWickets) {
         wicketsIndex = i;
      }

   }



   printf("\nPlayer with the highest runs:\n");
   printf("Name: %s\n",
   playerData[runsIndex].name); printf("Team:
   %s\n", playerData[runsIndex].team);
   printf("Total Runs: %d\n", playerData[runsIndex].totalRuns);
```

```
    printf("\nPlayer with the most wickets:\n");
    printf("Name: %s\n",
    playerData[wicketsIndex].name); printf("Team:
    %s\n", playerData[wicketsIndex].team);
    printf("Total Wickets: %d\n", playerData[wicketsIndex].totalWickets);

}
```

Problem 2: Tournament Fixture Scheduler
Requirements:

- Create a structure Match with members:
  - char team1[30]
  - char team2[30]
  - char date[11] (format: YYYY-MM-DD)
  - char venue[50]
- Functions to:
- Schedule a new match between two teams.
- Display all scheduled matches.
- Search for matches scheduled on a specific date.
- Cancel a match by specifying both team names and the date.
- Ensure that the match schedule is stored in an array, with the ability to dynamically adjust its size.

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


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


};
```

```
    // Function prototypes

    void addMatch(struct Match *matchPtr,  int *n);

    void displayAllMatches(struct Match *matchPtr,int
    n); void searchMatch(struct Match *matchPtr,int n);
void cancelMatch(struct Match *matchPtr,int *n);



    int main() {
      int n = 0;
      int capacity = 10;

      struct Match *matchData = (struct Match *)malloc(capacity * sizeof(struct
    Match));

      if (matchData == NULL) {
        printf("Memory Allocation Failed\n");
        return 1;
      }



      int
      choice;
      do {
        printf("\n--- Tournament Fixture Scheduler ---\n");
        printf("1. Add new match\n");
     printf("2. Display all scheduled matches\n");

        printf("3. Search for matches scheduled on a specific date\n");

        printf("4. Cancel a match by specifying both team names and the date\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                addMatch(matchData,
                &n); break;
            case 2:

                displayAllMatches(matchData,n);
                break;
            case 3:

                searchMatch(matchData, n);
                break;
            case 4:

                cancelMatch(matchData,&n);
                break;
            case 5:

                printf("Exiting...\n");
                break;
            default:

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

        }

    } while (choice != 5);



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

```
//Cancel a Match

 void cancelMatch(struct Match *matchPtr, int
    *n) { if (*n == 0) {
        printf("No matches scheduled to
        cancel!\n"); return;
    }

    char team1[30], team2[30], date[11];
    printf("\nEnter details of the match to
    cancel:\n"); printf("Team1: ");
    scanf(" %[^\n]s", team1);

    printf("Team2: ");

    scanf(" %[^\n]s", team2);

    printf("Date (YYYY-MM-DD): ");

    scanf(" %[^\n]s", date);



    int found = 0;

    for (int i = 0; i < *n; i++) {

        if (strcmp(matchPtr[i].team1, team1) == 0
            && strcmp(matchPtr[i].team2, team2) ==
            0 && strcmp(matchPtr[i].date, date) ==
            0) {
            // Shift matches to remove the canceled
            match for (int j = i; j < *n - 1; j++) {
                matchPtr[j] = matchPtr[j + 1];

            }
            (*n)--;

            printf("Match canceled successfully!\n");
            found = 1;
            break;

        }

    }

    if (!found) {

        printf("No match found with the specified details.\n");

    }

}



//Search Match by date

void searchMatch(struct Match *matchPtr,int
    n){ if(n==0){
        printf("No matches scheduled!");
        return;
    }

    char searchDate[10];

    printf("Enter     Date     to     search
    Match\n");                          scanf("
    %[^\n]s",searchDate);
    for(int                              i=0;i<n;i++){
        if(strcmp(matchPtr[i].date,searchDate)==
        0){
            printf("Match              details              on
            %s\n",searchDate);
            printf("\nTeam1:%s\n",matchPtr[i].team1
            );                              printf("Team2:
            %s\n",matchPtr[i].team2);
```

```
         printf("Date: %s\n",matchPtr[i].date);
         printf("Venue: %s\n",matchPtr[i].venue);


      }

   }

}



//Displays all scheduledmatches

void displayAllMatches(struct Match *matchPtr,int n){
   if(n==0){
      printf("No scheduled
      Matches!\n"); return;
   }

   for(int i=0;i<n;i++){
      printf("Match
      %d\n",i+1);
      printf("\nTeam1: %s\n",matchPtr[i].team1);
      printf("Team2: %s\n",matchPtr[i].team2);
      printf("Date: %s\n",matchPtr[i].date);
      printf("Venue: %s\n",matchPtr[i].venue);
   }



}

// Function to add a new player

void addMatch(struct Match *matchPtr,  int *n) {
    printf("\nEnter Match %d Details:\n",*n+1);
    printf("Team1: ");
    scanf(" %[^\n]s", matchPtr[*n].team1);
    printf("Team2: ");
    scanf(" %[^\n]s",
    matchPtr[*n].team2); printf("Date: ");
    scanf(" %[^\n]s", matchPtr[*n].date);
    printf("Venue: ");
    scanf(" %[^\n]s", matchPtr[*n].venue);



    (*n)++;

    printf("Match added successfully!\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 *tally, int *n, int
*capacity); void updateMedalCount(struct CountryMedalTally *tally,
int n);
void displayTally(const struct CountryMedalTally *tally, int n);

void findCountryWithMostGold(const struct CountryMedalTally *tally, int n);



int main() {
   int n = 0;
   int capacity = 5;

   struct CountryMedalTally *tally = (struct CountryMedalTally
*)malloc(capacity * sizeof(struct CountryMedalTally));

   if (tally == NULL) {

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



   int choice;
```

```
    do {

        printf("\n--- Sports 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 number of gold
medals\n");

        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                addCountry(tally, &n, &capacity);
                break;
            case 2:

                updateMedalCount(tally,
                n); break;
            case 3:

                displayTally(tally,
                n); break;
            case 4:

                findCountryWithMostGold(tally,
                n); break;
            case 5:

                printf("Exiting...\n");
            break
         ;
         default:
            printf("Invalid choice! Please try again.\n");

      }

   } while (choice != 5);



   free(tally);
   return 0;
}



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

void addCountry(struct CountryMedalTally *tally, int *n, int
   *capacity) { if (*n == *capacity) {
      // Resize the array if capacity is full

      *capacity *= 2;

      struct CountryMedalTally *newTally = realloc(tally, (*capacity) *
sizeof(struct CountryMedalTally));

      if (newTally == NULL) {

         printf("Memory reallocation failed!\n");
         return;
      }

      tally = newTally;

   }



   printf("\nEnter details for country %d:\n", *n + 1);
   printf("Country Name: ");
```

```
    scanf(" %[^\n]s",
    tally[*n].country); printf("Gold
    Medals: "); scanf("%d",
    &tally[*n].gold); printf("Silver
    Medals: "); scanf("%d",
    &tally[*n].silver); printf("Bronze
    Medals: "); scanf("%d",
    &tally[*n].bronze);


    (*n)++;

    printf("Country added successfully!\n");

}



// Function to update medal count for a country

void updateMedalCount(struct CountryMedalTally *tally, int
    n) { if (n == 0) {
        printf("No countries in the tally to
        update.\n"); return;
    }



    char country[30];

    printf("Enter the name of the country to update: ");
    scanf(" %[^\n]s", country);


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

        if (strcmp(tally[i].country, country) == 0) {
            printf("Updating medal count for %s:\n", tally[i].country);
            printf("Gold Medals: ");
            scanf("%d", &tally[i].gold);
            printf("Silver Medals: ");
            scanf("%d", &tally[i].silver);
            printf("Bronze Medals: ");
            scanf("%d",
            &tally[i].bronze);
            printf("Medal count updated successfully!\n");
            return;
        }

    }

    printf("Country '%s' not found in the tally.\n", country);

}



// Function to display the medal tally for all countries

void displayTally(const struct CountryMedalTally *tally, int
    n) { if (n == 0) {
        printf("No countries in the tally.\n");
        return;
    }



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



// Function to find and display the country with the highest number of gold
medals void findCountryWithMostGold(const struct CountryMedalTally
*tally, int n) {
    if (n == 0) {

        printf("No countries in the tally.\n");
        return;
    }



    int maxGold = -1;
    int index = -1;
    for (int i = 0; i < n; i++) {

        if (tally[i].gold > maxGold) {
            maxGold = tally[i].gold;
            index = i;
        }

    }



    if (index != -1) {

        printf("\nCountry with the highest number of gold
        medals:\n"); printf("Country: %s\n", tally[index].country);
        printf("Gold Medals: %d\n", tally[index].gold);
        printf("Silver Medals: %d\n", tally[index].silver);
        printf("Bronze Medals: %d\n", tally[index].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;

};
```

```
void addAthlete(struct Athlete **athletes, int *n, int
*capacity); void updatePerformance(struct Athlete
*athletes, int n);
void displayAthletesBySport(const struct Athlete *athletes, int
n); void displayNewPersonalBest(const struct Athlete *athletes,
int n);


int main() {

   int n = 0;          // Current number of athletes
   int capacity = 5;    // Initial capacity of the
   array
   struct Athlete *athletes = (struct Athlete *)malloc(capacity * sizeof(struct
   Athlete));



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



   int
   choice;
   do {
      printf("\n--- Athlete 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\n"); printf("5. Exit\n");
      printf("Enter your choice: ");
```

```
    scanf("%d", &choice);



    switch (choice) {
        case 1:
            addAthlete(&athletes, &n,
            &capacity); break;
        case 2:

            updatePerformance(athletes
            , n); break;
        case 3:

            displayAthletesBySport(athletes,
            n); break;
        case 4:

            displayNewPersonalBest(athletes
            , n); break;
        case 5:

            printf("Exiting...\n");
            break;
        default:

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

    }

} while (choice != 5);



free(athletes);
return 0;
```

```
}



// Function to add a new athlete

void addAthlete(struct Athlete **athletes, int *n, int
   *capacity) { if (*n == *capacity) {
      *capacity *= 2; // Double the capacity

      struct Athlete *newAthletes = (struct Athlete *)realloc(*athletes, (*capacity)
* sizeof(struct Athlete));

      if (newAthletes == NULL) {
         printf("Memory reallocation failed!\n");
         return;
      }

      *athletes = newAthletes;

   }



   printf("\nEnter details for athlete %d:\n", *n + 1);
   printf("Athlete ID: ");
   scanf(" %[^\n]s", (*athletes)[*n].athleteID);
   printf("Name: ");
   scanf(" %[^\n]s", (*athletes)[*n].name);
   printf("Sport: ");
   scanf(" %[^\n]s", (*athletes)[*n].sport);
   printf("Personal Best: ");
   scanf("%f",
   &(*athletes)[*n].personalBest);
   printf("Last Performance: ");
   scanf("%f", &(*athletes)[*n].lastPerformance);
```

```
    (*n)++;

    printf("Athlete added successfully!\n");

}



// Function to update an athlete's last performance

void updatePerformance(struct Athlete *athletes, int
    n) { if (n == 0) {
        printf("No athletes in the system.\n");
        return;
    }



    char athleteID[10];

    printf("Enter the Athlete ID to update performance:
    "); scanf(" %[^\n]s", athleteID);


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

        if (strcmp(athletes[i].athleteID, athleteID) == 0) {

            printf("Current Last Performance: %.2f\n", athletes[i].lastPerformance);
            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
                set!\n");
            } else {

            printf("Performance updated but no new personal best.\n");

            }

            return;

        }

    }

    printf("Athlete ID '%s' not found.\n", athleteID);

}



// Function to display all athletes in a specific sport

void displayAthletesBySport(const struct Athlete *athletes,
    int n) { if (n == 0) {
        printf("No athletes in the system.\n");
        return;
    }



    char sport[30];

    printf("Enter the sport to display athletes: ");
    scanf(" %[^\n]s", sport);


    printf("\n--- Athletes in %s ---\n", sport);
    int found = 0;
    for (int i = 0; i < n; i++) {

        if (strcmp(athletes[i].sport, sport) == 0) {
            printf("Athlete ID: %s\n", athletes[i].athleteID);
            printf("Name: %s\n", athletes[i].name);

            printf("Personal Best: %.2f\n", athletes[i].personalBest);
            printf("Last Performance: %.2f\n",
            athletes[i].lastPerformance); printf("        \n");
            found = 1;
                    ----------------
        }

    }



    if (!found) {

        printf("No athletes found in the sport '%s'.\n", sport);

    }

}



// Function to display athletes who set a new personal best

void displayNewPersonalBest(const struct Athlete *athletes,
    int n) { if (n == 0) {
        printf("No athletes in the system.\n");
        return;
    }



    printf("\n--- Athletes Who Set a New Personal Best ---
    \n"); int found = 0;
    for (int i = 0; i < n; i++) {

        if (athletes[i].lastPerformance == athletes[i].personalBest) {
            printf("Athlete ID: %s\n", athletes[i].athleteID);
            printf("Name: %s\n", athletes[i].name);
            printf("Sport: %s\n", athletes[i].sport);
            printf("New Personal Best: %.2f\n",
            athletes[i].personalBest); printf("   \n");
            found = 1;---------------

        }

    }



    if (!found) {

        printf("No athletes have set a new personal best.\n");

    }

}
```

Problem 5: Sports Equipment Inventory System
Requirements:

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

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.

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.

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.

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.

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.