---struct functions---

1.vehicle Fleet

```c
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<stdbool.h>

#include<math.h>

struct details {

    char reg_no[15];

    char model[20];

    int year;

    float milage;

    float fuel_efficiency;

};

int j=0;

void Add_vehicle(struct details *vehicles, int n) {

    printf("Enter the vehicle details:\n");

    printf("Enter the registration number: ");

    scanf(" %[^\n]", vehicles[j].reg_no);

    printf("Enter the model: ");

    scanf(" %[^\n]", vehicles[j].model);

    printf("Enter the year of manufacture: ");

    scanf("%d", &vehicles[j].year);

    printf("Enter the mileage: ");

    scanf("%f", &vehicles[j].milage);

    printf("Enter the fuel efficiency: ");

    scanf("%f", &vehicles[j].fuel_efficiency);

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

    j+=1;
```

```
}

void Update_milage(struct details *vehicles, int n) {
    float org_milage;
    char reg[30];
    printf("Enter the registration number of the vehicle: ");
    scanf(" %[^\n]", reg);
    for (int i = 0; i < n; i++) {
        if (strcmp(vehicles[i].reg_no, reg) == 0) {
            printf("Enter the new mileage: ");
            scanf("%f", &org_milage);
            vehicles[i].milage = org_milage;
            printf("Successfully updated the mileage!\n");
            return;
        }
    }
    printf("No vehicle found with the provided registration number!\n");
}

void Display_vehicles(struct details *vehicles, int n) {
    printf("Displaying all vehicles:\n");
    for (int i = 0; i < n; i++) {
        printf("RegNo: %s   Model: %s   Year of Manufacture: %d   Mileage: %.2f   Fuel Efficiency: %.2f\n",
               vehicles[i].reg_no, vehicles[i].model, vehicles[i].year, vehicles[i].milage,
vehicles[i].fuel_efficiency);
    }
}

void Highest_fuel_efficiency(struct details *vehicles, int n) {
    float max = -INFINITY;
    int maxIndex = -1;

    for (int i = 0; i < n; i++) {
        if (vehicles[i].fuel_efficiency > max) {
            max = vehicles[i].fuel_efficiency;
            maxIndex = i;
        }
    }

    if (maxIndex != -1) {
        printf("Vehicle with the highest fuel efficiency:\n");
        printf("RegNo: %s   Model: %s   Year: %d   Mileage: %.2f   Fuel Efficiency: %.2f\n",
               vehicles[maxIndex].reg_no, vehicles[maxIndex].model, vehicles[maxIndex].year,
               vehicles[maxIndex].milage, vehicles[maxIndex].fuel_efficiency);
    } else {
        printf("No vehicles found!\n");
    }
}

int main() {
    int n;
    printf("Enter the number of vehicles: ");
    scanf("%d", &n);
    struct details *vehicles = (struct details *)malloc(n * sizeof(struct details));

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


    bool is_on = true;
    while (is_on) {
        printf("\n1. Add vehicle\n2. Update mileage\n3. Display all vehicles\n4. Highest fuel efficiency\n5.
Exit\n");
        int user_input;
        scanf("%d", &user_input);


        switch (user_input) {
            case 1:
                Add_vehicle(vehicles, n);
                break;
            case 2:
                Update_milage(vehicles, n);
                break;
            case 3:
                Display_vehicles(vehicles, n);
                break;
            case 4:
                Highest_fuel_efficiency(vehicles, n);
                break;
            case 5:
                is_on = false;
                break;
            default:
                printf("Enter a valid option.\n");
        }
    }
    free(vehicles);
    vehicles = NULL;

    return 0;
}
```

2.

car rental reservation system

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

struct date {
    int day;
    int month;
    int year;
};

struct CarRental {
    char carID[10];
    char customerName[50];
    struct date rentalDate;
    struct date returnDate;
    float rentalPricePerDay;
};

int calculate_days(struct date start, struct date end) {
    int startDays = start.year * 365 + start.month * 30 + start.day;
    int endDays = end.year * 365 + end.month * 30 + end.day;
    return endDays - startDays;
}

void book_car(struct CarRental *cars, int n) {
    struct CarRental newRental;

    printf("Enter Car ID: ");
    scanf(" %[^\n]", newRental.carID);
    printf("Enter Customer Name: ");
    scanf(" %[^\n]", newRental.customerName);
    printf("Enter Rental Date (YYYY-MM-DD): ");
    scanf("%d-%d-%d", &newRental.rentalDate.year, &newRental.rentalDate.month,
&newRental.rentalDate.day);
    printf("Enter Return Date (YYYY-MM-DD): ");
    scanf("%d-%d-%d", &newRental.returnDate.year, &newRental.returnDate.month,
&newRental.returnDate.day);
    printf("Enter Rental Price Per Day: ");
    scanf("%f", &newRental.rentalPricePerDay);

    cars[n] = newRental;
}

float calculate_rental_price(struct CarRental car) {
    int rentalDays = calculate_days(car.rentalDate, car.returnDate);
    if (rentalDays < 0) {
        printf("Error: Return date cannot be before rental date.\n");
        return 0;
    }
    return rentalDays * car.rentalPricePerDay;
}

void display_current_rentals(struct CarRental *cars, int n) {
    printf("Current Rentals:\n");
    for (int i = 0; i < n; i++) {
        printf("Car ID: %s | Customer: %s | Rental Date: %d-%d-%d | Return Date: %d-%d-%d | Price Per Day: %.2f\n",
               cars[i].carID, cars[i].customerName,
               cars[i].rentalDate.year, cars[i].rentalDate.month, cars[i].rentalDate.day,
               cars[i].returnDate.year, cars[i].returnDate.month, cars[i].returnDate.day,
               cars[i].rentalPricePerDay);
    }
}

void search_by_name(struct CarRental *cars, int n, char *customerName) {
    printf("Searching rentals for Customer: %s\n", customerName);
    for (int i = 0; i < n; i++) {
        if (strcmp(cars[i].customerName, customerName) == 0) {
            printf("Car ID: %s | Rental Date: %d-%d-%d | Return Date: %d-%d-%d | Price Per Day: %.2f\n",
                   cars[i].carID, cars[i].rentalDate.year, cars[i].rentalDate.month, cars[i].rentalDate.day,
                   cars[i].returnDate.year, cars[i].returnDate.month, cars[i].returnDate.day,
                   cars[i].rentalPricePerDay);
        }
    }
}

int main() {
    struct CarRental carDetails[5];
    int n = 0;

    bool is_on = true;

    int user_input;


    while (is_on) {

        printf("\n1. Book a car\n2. Calculate rental price\n3. Display current rentals\n4. Search by customer
name\n5. Exit\n");

        scanf("%d", &user_input);

        switch (user_input) {

            case 1:

                book_car(carDetails, n);

                n++;

                break;

            case 2:

                {

                    char carID[10];

                    printf("Enter Car ID to calculate rental price: ");

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

                    int found = 0;

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

                        if (strcmp(carDetails[i].carID, carID) == 0) {

                            float price = calculate_rental_price(carDetails[i]);

                            if (price > 0) {

                                printf("Total rental price for car ID %s: %.2f\n", carID, price);

                            }

                            found = 1;

                            break;

                        }

                    }
                    if (!found) {
                        printf("Car ID not found.\n");
                    }
                }
                break;
            case 3:
                display_current_rentals(carDetails, n);
                break;
            case 4:
                {
                    char customerName[50];
                    printf("Enter customer name to search: ");
                    scanf(" %[^\n]", customerName);
                    search_by_name(carDetails, n, customerName);
                }
                break;
            case 5:
                is_on = false;
                printf("Thank you! Come back again.\n");
                break;
            default:
                printf("Invalid option. Please try again.\n");
        }
    }

    return 0;
}
```

3.

vehicle sensor data logger

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

#define MAX_TIMESTAMP_LENGTH 20

struct SensorData {
    int sensorID;
    char timestamp[MAX_TIMESTAMP_LENGTH];
    float speed;
    float latitude;
    float longitude;
};

int calculate_days(char *startTime, char *endTime) {
    int startYear, startMonth, startDay;
    int endYear, endMonth, endDay;

    sscanf(startTime, "%d-%d-%d", &startYear, &startMonth, &startDay);
    sscanf(endTime, "%d-%d-%d", &endYear, &endMonth, &endDay);

    return (endYear - startYear) * 365 + (endMonth - startMonth) * 30 + (endDay - startDay);
}

void logSensorData(struct SensorData **data, int *size, int *capacity, struct SensorData newData) {
    if (*size >= *capacity) {
        *capacity *= 2;
        *data = realloc(*data, *capacity * sizeof(struct SensorData));
    }
    (*data)[*size] = newData;
    (*size)++;
}

void displayDataInRange(struct SensorData *data, int size, char *startTime, char *endTime) {
    for (int i = 0; i < size; i++) {
        if (calculate_days(data[i].timestamp, startTime) >= 0 && calculate_days(data[i].timestamp, 
endTime) <= 0) {
            printf("SensorID: %d | Timestamp: %s | Speed: %.2f | Latitude: %.6f | Longitude: %.6f\n",
                   data[i].sensorID, data[i].timestamp, data[i].speed, data[i].latitude, data[i].longitude);
        }
    }
}

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

float calculateAvgSpeed(struct SensorData *data, int size, char *startTime, char *endTime) {
    float totalSpeed = 0;
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (calculate_days(data[i].timestamp, startTime) >= 0 && calculate_days(data[i].timestamp,
endTime) <= 0) {
            totalSpeed += data[i].speed;
            count++;
        }
    }
    return (count > 0) ? totalSpeed / count : 0;
}

int main() {
    struct SensorData *data = NULL;
    int size = 0;
    int capacity = 2;
    data = malloc(capacity * sizeof(struct SensorData));
    int choice;

    while (1) {
        printf("\n1. Log new sensor data\n2. Display sensor data for a specific time range\n3. Find maximum
speed\n4. Calculate average speed\n5. Exit\n");
        scanf("%d", &choice);

        if (choice == 1) {
            struct SensorData newData;
            printf("Enter Sensor ID: ");
            scanf("%d", &newData.sensorID);
            printf("Enter Timestamp (YYYY-MM-DD HH:MM:SS): ");
            scanf(" %[^\n]", newData.timestamp);
            printf("Enter Speed: ");
            scanf("%f", &newData.speed);
            printf("Enter Latitude: ");
            scanf("%f", &newData.latitude);
            printf("Enter Longitude: ");
            scanf("%f", &newData.longitude);
            logSensorData(&data, &size, &capacity, newData);
        }
        else if (choice == 2) {
            char startTime[MAX_TIMESTAMP_LENGTH], endTime[MAX_TIMESTAMP_LENGTH];
            printf("Enter start time (YYYY-MM-DD HH:MM:SS): ");
            scanf(" %[^\n]", startTime);
            printf("Enter end time (YYYY-MM-DD HH:MM:SS): ");
            scanf(" %[^\n]", endTime);
            displayDataInRange(data, size, startTime, endTime);
        }
        else if (choice == 3) {
            printf("Maximum speed recorded: %.2f\n", findMaxSpeed(data, size));
        }
        else if (choice == 4) {
            char startTime[MAX_TIMESTAMP_LENGTH], endTime[MAX_TIMESTAMP_LENGTH];
            printf("Enter start time (YYYY-MM-DD HH:MM:SS): ");
            scanf(" %[^\n]", startTime);
            printf("Enter end time (YYYY-MM-DD HH:MM:SS): ");
            scanf(" %[^\n]", endTime);
            printf("Average speed: %.2f\n", calculateAvgSpeed(data, size, startTime, endTime));
        }
        else if (choice == 5) {
            free(data);
```

```
            printf("Exiting...\n");
            break;
        }
        else {
            printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

4.

Engine performance monitoring system

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

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

void add_data(struct EnginePerformance a[], int n) {
    for (int i = 0; i < n; i++) {
        printf("Enter the Engine ID: ");
        scanf("%s", a[i].engineID);
        printf("Enter the temperature: ");
        scanf("%f", &a[i].temperature);
        printf("Enter the rpm: ");
        scanf("%d", &a[i].rpm);
        printf("Enter the fuel consumption rate: ");
        scanf("%f", &a[i].fuelConsumptionRate);
        printf("Enter the oil pressure: ");
        scanf("%f", &a[i].oilPressure);
    }
}

void display_data(struct EnginePerformance a[], int n, char engineID[]) {
    for (int i = 0; i < n; i++) {
        if (strcmp(a[i].engineID, engineID) == 0) {
            printf("Engine ID: %s\n", a[i].engineID);
            printf("Temperature: %.2f\n", a[i].temperature);
            printf("RPM: %d\n", a[i].rpm);
            printf("Fuel Consumption Rate: %.2f\n", a[i].fuelConsumptionRate);
            printf("Oil Pressure: %.2f\n", a[i].oilPressure);
            return;
        }
    }
    printf("Engine ID not found.\n");
}

void calculate_average(struct EnginePerformance a[], int n, char engineID[]) {
    float totalTemperature = 0;
    int totalRPM = 0;
    int count = 0;

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

    if (count > 0) {
        printf("Average Temperature: %.2f\n", totalTemperature / count);
        printf("Average RPM: %d\n", totalRPM / count);
    } else {
        printf("Engine ID not found.\n");
    }
}

void identify_abnormal_oil_pressure(struct EnginePerformance a[], int n, float minOilPressure, float
maxOilPressure) {
    printf("Identifying engines with abnormal oil pressure:\n");
    for (int i = 0; i < n; i++) {
        if (a[i].oilPressure < minOilPressure || a[i].oilPressure > maxOilPressure) {
            printf("Engine ID: %s | Oil Pressure: %.2f (Abnormal)\n", a[i].engineID, a[i].oilPressure);
        }
    }
}

int main() {
    int n = 1;

    struct EnginePerformance *arr = (struct EnginePerformance *)malloc(n * sizeof(struct
EnginePerformance));

    bool is_on = true;

    char engineID[10];

    float minOilPressure = 10.0;

    float maxOilPressure = 80.0;


    while (is_on) {

        int user_input;

        printf("\n1. Add performance data\n2. Display performance data for a specific engine\n3. Calculate
average temperature and RPM for a specific engine\n4. Identify abnormal oil pressure\n5. Exit\n");

        scanf("%d", &user_input);


        switch (user_input) {

            case 1:

                printf("Enter the number of engines to add data for: ");

                scanf("%d", &n);

                arr = realloc(arr, n * sizeof(struct EnginePerformance));

                add_data(arr, n);

                break;

            case 2:

                printf("Enter Engine ID to display data: ");

                scanf("%s", engineID);

                display_data(arr, n, engineID);

                break;

            case 3:

                printf("Enter Engine ID to calculate average temperature and RPM: ");

                scanf("%s", engineID);
            calculate_average(arr, n, engineID);
            break;
          case 4:
            identify_abnormal_oil_pressure(arr, n, minOilPressure, maxOilPressure);
            break;
          case 5:
            is_on = false;
            printf("Exiting the program...\n");
            break;
          default:
            printf("Invalid option, please try again.\n");
        }
    }

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

5.vehicle service history tracker

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

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

```
};

void Add_Service_Record(struct ServiceRecord *records, int *n) {
    printf("Enter Service ID: ");
    scanf(" %[^\n]", records[*n].serviceID);
    printf("Enter Vehicle ID: ");
    scanf(" %[^\n]", records[*n].vehicleID);
    printf("Enter Service Date (DD/MM/YYYY): ");
    scanf(" %[^\n]", records[*n].serviceDate);
    printf("Enter Service Description: ");
    scanf(" %[^\n]", records[*n].description);
    printf("Enter Service Cost: ");
    scanf("%f", &records[*n].serviceCost);
    (*n)++;
}

void Display_Service_Records(struct ServiceRecord *records, int n, char *vehicleID) {
    for (int i = 0; i < n; i++) {
        if (strcmp(records[i].vehicleID, vehicleID) == 0) {
            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);
        }
    }
}

float Calculate_Total_Cost(struct ServiceRecord *records, int n, 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;
}

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

void Sort_Service_Records(struct ServiceRecord *records, int n) {
    qsort(records, n, sizeof(struct ServiceRecord), Compare_ServiceDate);
}

int main() {
    int n = 0;
    struct ServiceRecord *records = (struct ServiceRecord *)malloc(100 * sizeof(struct ServiceRecord));

    while (1) {
        printf("\n1. Add Service Record\n2. Display Service Records\n3. Calculate Total Cost\n4. Sort Service
Records by Date\n5. Exit\n");
        int choice;
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                Add_Service_Record(records, &n);
                break;
            case 2: {
                char vehicleID[15];
                printf("Enter Vehicle ID to search: ");
                scanf(" %[^\n]", vehicleID);
                Display_Service_Records(records, n, vehicleID);
                break;
            }
            case 3: {
                char vehicleID[15];
                printf("Enter Vehicle ID to calculate total cost: ");
                scanf(" %[^\n]", vehicleID);
                printf("Total Service Cost: %.2f\n", Calculate_Total_Cost(records, n, vehicleID));
                break;
            }
            case 4:
                Sort_Service_Records(records, n);
                printf("Service records sorted by date.\n");
                break;
            case 5:
                free(records);
                return 0;
            default:
                printf("Invalid option. Please try again.\n");
        }
    }
}
```

6.player statitics managment.

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

```
#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include<math.h>

struct datas {

    char name[50];

    int age;

    char team[30];

    int matchesPlayed;

    int totalRuns;

    int totalWickets;

};


void add_player(struct datas **arr, int *n) {

    *arr = realloc(*arr, (*n + 1) * sizeof(struct datas));

    if (*arr == NULL) {

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

        exit(1);

    }


    int index = *n;

    printf("Enter the player name: ");

    getchar();

    scanf("%[^\n]", (*arr)[index].name);


    printf("Enter the player age: ");

    scanf("%d", &(*arr)[index].age);


    printf("Enter the team in which the player is playing: ");
    getchar();
    scanf("%[^\n]", (*arr)[index].team);

    printf("Enter the total matches played: ");
    scanf("%d", &(*arr)[index].matchesPlayed);

    printf("Enter the total runs scored: ");
    scanf("%d", &(*arr)[index].totalRuns);

    printf("Enter the total wickets taken: ");
    scanf("%d", &(*arr)[index].totalWickets);

    *n += 1;
    printf("Player added successfully!\n");
}
void update_player(struct datas **arr, int *n) {
    int u_o;
    printf("Enter the details which need to be updated:\n1. Total Runs\n2. Total Wickets\n3. Total Matches
Played\n");
    scanf("%d", &u_o);
    getchar();
    char str[30];
    printf("Enter the Player name: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';
    for (int i = 0; i < *n; i++) {
        if (strcmp(arr[i]->name, str) == 0) {
            int sc;
            switch (u_o) {
```

```
            case 1:
                printf("Enter the new runs scored: ");
                scanf("%d", &sc);
                arr[i]->totalRuns = sc;
                printf("Total Runs updated successfully!\n");
                return;

            case 2:
                printf("Enter the new wickets taken: ");
                scanf("%d", &sc);
                arr[i]->totalWickets = sc;
                printf("Total Wickets updated successfully!\n");
                return;

            case 3:
                printf("Enter the new matches played: ");
                scanf("%d", &sc);
                arr[i]->matchesPlayed = sc;
                printf("Total Matches Played updated successfully!\n");
                return;

            default:
                printf("Enter a valid option!\n");
                return;
        }
    }
}

printf("Player with the name '%s' not found.\n", str);
```

```
}
void display_details(struct datas **arr, int *n) {
   char str3[20];
   printf("Enter the name of the player: ");
   getchar();
   scanf("%[^\n]",str3);
   int found = 0;
   for (int i = 0; i < *n; i++) {
      if (strcmp((*arr)[i].name, str3) == 0) {
         printf("Name: %s, Age: %d, Team: %s, Matches Played: %d, Total Runs: %d, Total Wickets: %d\n",
            (*arr)[i].name, (*arr)[i].age, (*arr)[i].team,
            (*arr)[i].matchesPlayed, (*arr)[i].totalRuns, (*arr)[i].totalWickets);
         found = 1;
         break;
      }
   }

   if (!found) {
      printf("Player with the name '%s' not found.\n", str3);
   }
}
void highest_runs(struct datas **arr,int *n){
   int highest_run=0;
   for(int i=0;i<*n;i++){
      if((*arr)[i].totalRuns>highest_run){
         highest_run=(*arr)[i].totalRuns;
      }
   }
   for(int i=0;i<*n;i++){
        if((*arr)[i].totalRuns==highest_run){
            printf("%s scored the highest run %d",(*arr)[i].name,highest_run);
        }
    }
}

int main() {
    int n = 0;
    struct datas *arr = (struct datas *)malloc(n * sizeof(struct datas));

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

    bool is_on = true;
    while (is_on) {
        int user_input;
        printf("\nChoose your option:\n");
        printf("1. Add a new player\n2. Update player statistics\n3. Display details of player\n4. Highest
runs\n5. Exit\n");
        scanf("%d", &user_input);

        switch (user_input) {
            case 1:
                add_player(&arr, &n);
                break;
            case 2:
                update_player(&arr,&n);
                break;

            case 3:
                display_details(&arr,&n);
                break;
            case 4:
                highest_runs(&arr,&n);
                break;

            case 5:
                is_on = false;
                printf("Exiting program...\n");
                break;

            default:
                printf("Enter a valid option.\n");
        }
    }
    free(arr);

    return 0;
}
```

7.Tournament fixter

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

struct Date {
    int year;
    int month;
    int day;
};

struct Match {
    char team1[30];
    char team2[30];
    struct Date matchDate;
    char venue[50];
};

struct Match matches[100];
int matchCount = 0;

void scheduleMatch(char team1[], char team2[], struct Date matchDate, char venue[]) {
    strcpy(matches[matchCount].team1, team1);
    strcpy(matches[matchCount].team2, team2);
    matches[matchCount].matchDate = matchDate;
    strcpy(matches[matchCount].venue, venue);
    matchCount++;
}

void displayMatches() {
    for (int i = 0; i < matchCount; i++) {
        printf("Match: %s vs %s, Date: %04d-%02d-%02d, Venue: %s\n",
                matches[i].team1, matches[i].team2, matches[i].matchDate.year,
                matches[i].matchDate.month, matches[i].matchDate.day, matches[i].venue);
    }
}
```

```
void searchMatchesByDate(struct Date searchDate) {
  for (int i = 0; i < matchCount; i++) {
    if (matches[i].matchDate.year == searchDate.year &&
      matches[i].matchDate.month == searchDate.month &&
      matches[i].matchDate.day == searchDate.day) {
      printf("Match: %s vs %s, Date: %04d-%02d-%02d, Venue: %s\n",
          matches[i].team1, matches[i].team2, matches[i].matchDate.year,
          matches[i].matchDate.month, matches[i].matchDate.day, matches[i].venue);
    }
  }
}

void cancelMatch(char team1[], char team2[], struct Date matchDate) {
  for (int i = 0; i < matchCount; i++) {
    if (strcmp(matches[i].team1, team1) == 0 && strcmp(matches[i].team2, team2) == 0 &&
      matches[i].matchDate.year == matchDate.year && matches[i].matchDate.month ==
matchDate.month &&
      matches[i].matchDate.day == matchDate.day) {
      for (int j = i; j < matchCount - 1; j++) {
        matches[j] = matches[j + 1];
      }
      matchCount--;
      printf("Match between %s and %s on %04d-%02d-%02d has been canceled.\n",
          team1, team2, matchDate.year, matchDate.month, matchDate.day);
      return;
    }
  }
  printf("Match not found.\n");
}

int main() {
  struct Date date1 = {2025, 5, 10};
  struct Date date2 = {2025, 5, 15};

  scheduleMatch("Team A", "Team B", date1, "Stadium 1");
  scheduleMatch("Team C", "Team D", date2, "Stadium 2");

  printf("All Matches:\n");
  displayMatches();

  printf("\nSearch for matches on 2025-05-10:\n");
  searchMatchesByDate(date1);

  printf("\nCancel match between Team A and Team B on 2025-05-10:\n");
  cancelMatch("Team A", "Team B", date1);

  printf("\nAll Matches after cancellation:\n");
  displayMatches();

  return 0;
}
```

8.

sports event medal tally

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

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

struct CountryMedalTally *tally = NULL;
int countryCount = 0;
int capacity = 0;

void resizeArray() {
    capacity = (capacity == 0) ? 1 : capacity * 2;
    tally = realloc(tally, capacity * sizeof(struct CountryMedalTally));
}

void addCountryMedalTally(char country[], int gold, int silver, int bronze) {
    if (countryCount == capacity) {
        resizeArray();
    }
    strcpy(tally[countryCount].country, country);
    tally[countryCount].gold = gold;
    tally[countryCount].silver = silver;
    tally[countryCount].bronze = bronze;
    countryCount++;
}

void updateMedals(char country[], int gold, int silver, int bronze) {
    for (int i = 0; i < countryCount; i++) {
        if (strcmp(tally[i].country, country) == 0) {
            tally[i].gold += gold;
            tally[i].silver += silver;
            tally[i].bronze += bronze;
            return;
        }
    }
    printf("Country not found.\n");
}

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

void displayCountryWithMostGold() {
    if (countryCount == 0) {
        printf("No countries added.\n");
        return;
    }

    int maxGold = -1;
    char countryWithMostGold[30];

    for (int i = 0; i < countryCount; i++) {
        if (tally[i].gold > maxGold) {
            maxGold = tally[i].gold;
            strcpy(countryWithMostGold, tally[i].country);
        }
    }

    printf("Country with the most gold medals: %s (%d Gold)\n", countryWithMostGold, maxGold);
}

int main() {
    addCountryMedalTally("USA", 10, 15, 5);
    addCountryMedalTally("China", 12, 8, 6);
    addCountryMedalTally("Germany", 5, 10, 8);

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

    printf("\nUpdating medals for China (2 gold, 3 silver, 1 bronze):\n");
    updateMedals("China", 2, 3, 1);

    printf("\nUpdated Medal Tally:\n");
    displayMedalTally();

    printf("\n");
    displayCountryWithMostGold();

    // Free dynamically allocated memory
    free(tally);

    return 0;
```

```
}
```

9. Athlete performance tracker.

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

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

struct Athlete *athletes = NULL;
int athleteCount = 0;
int capacity = 0;

void resizeArray() {
    capacity = (capacity == 0) ? 1 : capacity * 2;
    athletes = realloc(athletes, capacity * sizeof(struct Athlete));
}

void addAthlete(char athleteID[], char name[], char sport[], float personalBest) {
    if (athleteCount == capacity) {
        resizeArray();
    }
    strcpy(athletes[athleteCount].athleteID, athleteID);
    strcpy(athletes[athleteCount].name, name);
    strcpy(athletes[athleteCount].sport, sport);
    athletes[athleteCount].personalBest = personalBest;
    athletes[athleteCount].lastPerformance = personalBest;  // Assuming initial performance is equal to
personal best
    athleteCount++;
}

void updatePerformance(char athleteID[], float lastPerformance) {
    for (int i = 0; i < athleteCount; i++) {
        if (strcmp(athletes[i].athleteID, athleteID) == 0) {
            athletes[i].lastPerformance = lastPerformance;
            return;
        }
    }
    printf("Athlete not found.\n");
}

void displayAthletesInSport(char sport[]) {
    int found = 0;
    for (int i = 0; i < athleteCount; i++) {
        if (strcmp(athletes[i].sport, sport) == 0) {
            printf("ID: %s, Name: %s, Sport: %s, Personal Best: %.2f, Last Performance: %.2f\n",
                   athletes[i].athleteID, athletes[i].name, athletes[i].sport,
                   athletes[i].personalBest, athletes[i].lastPerformance);
            found = 1;
        }
    }
    if (!found) {
        printf("No athletes found in sport: %s\n", sport);
    }
}

void displayAthletesWithNewPersonalBest() {
    int found = 0;
    for (int i = 0; i < athleteCount; i++) {
        if (athletes[i].lastPerformance < athletes[i].personalBest) {
            athletes[i].personalBest = athletes[i].lastPerformance;
            printf("Athlete %s has set a new personal best: %.2f\n",
                   athletes[i].name, athletes[i].personalBest);
            found = 1;
        }
    }
    if (!found) {
        printf("No athlete has set a new personal best.\n");
    }
}

int main() {
    addAthlete("A123", "John Doe", "Track", 9.58);
    addAthlete("A124", "Jane Smith", "Swimming", 52.56);
    addAthlete("A125", "Michael Johnson", "Track", 19.32);

    printf("Displaying all athletes in Track:\n");
    displayAthletesInSport("Track");

    printf("\nUpdating performance for John Doe to 9.52:\n");
    updatePerformance("A123", 9.52);

    printf("\nDisplaying all athletes in Track after update:\n");
    displayAthletesInSport("Track");

    printf("\nDisplaying athletes with new personal bests:\n");
    displayAthletesWithNewPersonalBest();

    // Free dynamically allocated memory
    free(athletes);

    return 0;
}
```

10.sports equipment inventory system.

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

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

void add_equipment(struct Equipment **inventory, int *n) {
    *inventory = realloc(*inventory, (*n + 1) * sizeof(struct Equipment));
    if (*inventory == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    printf("Enter Equipment ID: ");
    scanf("%s", (*inventory)[*n].equipmentID);
    printf("Enter Equipment Name: ");
    scanf(" %[^\n]", (*inventory)[*n].name);
    printf("Enter Equipment Category: ");
    scanf(" %[^\n]", (*inventory)[*n].category);
    printf("Enter Quantity: ");
    scanf("%d", &(*inventory)[*n].quantity);
    printf("Enter Price per Unit: ");
    scanf("%f", &(*inventory)[*n].pricePerUnit);
    (*n)++;
}

void update_quantity(struct Equipment *inventory, int n) {
    char id[10];
    int quantity;
    printf("Enter Equipment ID to update: ");
    scanf("%s", id);
    printf("Enter new quantity: ");
    scanf("%d", &quantity);
    for (int i = 0; i < n; i++) {
        if (strcmp(inventory[i].equipmentID, id) == 0) {
            inventory[i].quantity = quantity;
            printf("Quantity updated successfully.\n");
            return;
        }
    }
    printf("Equipment ID not found.\n");
}

void display_category(struct Equipment *inventory, int n, const char *category) {
    printf("Equipment in category %s:\n", category);
    for (int i = 0; i < n; 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);
        }
    }
}

float total_inventory_value(struct Equipment *inventory, int n) {
    float total = 0;
    for (int i = 0; i < n; i++) {
        total += inventory[i].quantity * inventory[i].pricePerUnit;
    }
    return total;
}

int main() {
    int n = 0;
    struct Equipment *inventory = malloc(n * sizeof(struct Equipment));
    int choice;
    while (1) {
        printf("\n1. Add new equipment\n2. Update quantity\n3. Display equipment by category\n4.
Calculate total inventory value\n5. Exit\n");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                add_equipment(&inventory, &n);

                break;

            case 2:

                update_quantity(inventory, n);

                break;

            case 3:

                {

                    char category[20];

                    printf("Enter category to display: ");

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

                    display_category(inventory, n, category);

                }

                break;

            case 4:

                printf("Total inventory value: %.2f\n", total_inventory_value(inventory, n));

                break;

            case 5:

                free(inventory);

                return 0;

            default:

                printf("Invalid option.\n");

        }

    }

}
```

11.Research paper database management

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

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

void add_paper(struct ResearchPaper **arr, int *n) {
    *arr = realloc(*arr, (*n + 1) * sizeof(struct ResearchPaper));
    if (*arr == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    int index = *n;
    printf("Enter the title: ");
    getchar();
    fgets((*arr)[index].title, 100, stdin);
    printf("Enter the author: ");
    fgets((*arr)[index].author, 50, stdin);
    printf("Enter the journal: ");
    fgets((*arr)[index].journal, 50, stdin);
    printf("Enter the year: ");
    scanf("%d", &(*arr)[index].year);
    printf("Enter the DOI: ");
    getchar();
    fgets((*arr)[index].DOI, 30, stdin);

    *n += 1;
}

void update_paper(struct ResearchPaper **arr, int *n) {
    char DOI[30];
    printf("Enter the DOI of the paper to update: ");
    fgets(DOI, 30, stdin);
    DOI[strcspn(DOI, "\n")] = '\0';

    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].DOI, DOI) == 0) {
            printf("Enter new title: ");
            fgets((*arr)[i].title, 100, stdin);
            printf("Enter new author: ");
            fgets((*arr)[i].author, 50, stdin);
            printf("Enter new journal: ");
            fgets((*arr)[i].journal, 50, stdin);
            printf("Enter new year: ");
            scanf("%d", &(*arr)[i].year);
            printf("Enter new DOI: ");
            getchar();
            fgets((*arr)[i].DOI, 30, stdin);
            return;
        }
    }
    printf("Paper with the given DOI not found.\n");
}

void display_journal_papers(struct ResearchPaper **arr, int *n) {
    char journal[50];
    printf("Enter the journal name: ");
    fgets(journal, 50, stdin);
    journal[strcspn(journal, "\n")] = '\0';

    int found = 0;
    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].journal, journal) == 0) {
            printf("Title: %sAuthor: %sYear: %dDOI: %s\n", (*arr)[i].title, (*arr)[i].author, (*arr)[i].year,
(*arr)[i].DOI);
            found = 1;
        }
    }
    if (!found) {
        printf("No papers found for the given journal.\n");
    }
}

void display_recent_papers(struct ResearchPaper **arr, int *n) {
    char author[50];
    printf("Enter the author's name: ");
    fgets(author, 50, stdin);
    author[strcspn(author, "\n")] = '\0';
    int found = 0;

    int recent_year = 0;

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

        if (strcmp((*arr)[i].author, author) == 0 && (*arr)[i].year > recent_year) {

            recent_year = (*arr)[i].year;

        }

    }


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

        if (strcmp((*arr)[i].author, author) == 0 && (*arr)[i].year == recent_year) {

            printf("Title: %sAuthor: %sYear: %dDOI: %s\n", (*arr)[i].title, (*arr)[i].author, (*arr)[i].year,
(*arr)[i].DOI);

            found = 1;

        }

    }


    if (!found) {

        printf("No recent papers found for the given author.\n");

    }

}


int main() {

    int n = 0;

    struct ResearchPaper *arr = (struct ResearchPaper *)malloc(n * sizeof(struct ResearchPaper));


    if (arr == NULL) {

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

        return 1;
    }


    int choice;

    bool is_on = true;

    while (is_on) {

        printf("\n1. Add a new paper\n2. Update a paper\n3. Display papers from a journal\n4. Display most
recent papers by an author\n5. Exit\n");

        scanf("%d", &choice);

        getchar();


        switch (choice) {

            case 1:

                add_paper(&arr, &n);

                break;

            case 2:

                update_paper(&arr, &n);

                break;

            case 3:

                display_journal_papers(&arr, &n);

                break;

            case 4:

                display_recent_papers(&arr, &n);

                break;

            case 5:

                is_on = false;

                break;

            default:

                printf("Invalid option.\n");

        }
    }

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

12.Experimental Data logger

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

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

void new_experiment(struct experiment **arr, int *n) {
    *arr = realloc(*arr, (*n + 1) * sizeof(struct experiment));
    if (*arr == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    int index = *n;
    printf("Enter Experiment ID: ");
    getchar();  // To consume the leftover newline character
    fgets((*arr)[index].experimentID, sizeof((*arr)[index].experimentID), stdin);
    (*arr)[index].experimentID[strcspn((*arr)[index].experimentID, "\n")] = '\0';

    printf("Enter Researcher Name: ");
    fgets((*arr)[index].researcher, sizeof((*arr)[index].researcher), stdin);
    (*arr)[index].researcher[strcspn((*arr)[index].researcher, "\n")] = '\0';

    printf("Enter Start Date (YYYY-MM-DD): ");
    fgets((*arr)[index].startDate, sizeof((*arr)[index].startDate), stdin);
    (*arr)[index].startDate[strcspn((*arr)[index].startDate, "\n")] = '\0';

    printf("Enter End Date (YYYY-MM-DD): ");
    fgets((*arr)[index].endDate, sizeof((*arr)[index].endDate), stdin);
    (*arr)[index].endDate[strcspn((*arr)[index].endDate, "\n")] = '\0';

    for (int i = 0; i < 10; i++) {
        printf("Enter result #%d (or -1 to stop): ", i + 1);
        scanf("%f", &(*arr)[index].results[i]);
        if ((*arr)[index].results[i] == -1) {
            break;  // Stop if the user enters -1
        }
    }

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

void update_result(struct experiment **arr, int *n) {
    char experimentID[10];
    printf("Enter Experiment ID to update results: ");
    getchar();  // To consume the leftover newline character
    fgets(experimentID, sizeof(experimentID), stdin);
    experimentID[strcspn(experimentID, "\n")] = '\0';

    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].experimentID, experimentID) == 0) {
            printf("Experiment found. Update results.\n");
            for (int j = 0; j < 10; j++) {
                printf("Enter new result #%d (or -1 to stop): ", j + 1);
                scanf("%f", &(*arr)[i].results[j]);
                if ((*arr)[i].results[j] == -1) {
                    break;  // Stop if the user enters -1
                }
            }
            printf("Results updated successfully!\n");
            return;
        }
    }
    printf("Experiment with ID '%s' not found.\n", experimentID);
}

void display_experiments(struct experiment **arr, int *n) {
    char researcherName[50];
    printf("Enter Researcher Name to display their experiments: ");
    getchar();  // To consume the leftover newline character
    fgets(researcherName, sizeof(researcherName), stdin);
    researcherName[strcspn(researcherName, "\n")] = '\0';
    int found = 0;
    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].researcher, researcherName) == 0) {
            printf("\nExperiment ID: %s\nResearcher: %s\nStart Date: %s\nEnd Date: %s\nResults: ",
                   (*arr)[i].experimentID, (*arr)[i].researcher, (*arr)[i].startDate, (*arr)[i].endDate);
            for (int j = 0; j < 10; j++) {
                if ((*arr)[i].results[j] != -1) {
                    printf("%.2f ", (*arr)[i].results[j]);
                } else {
                    break;
                }
            }
            printf("\n");
            found = 1;
        }
    }
    if (!found) {
        printf("No experiments found for researcher '%s'.\n", researcherName);
    }
}

void average_result(struct experiment **arr, int *n) {
    char experimentID[10];
    printf("Enter Experiment ID to calculate average result: ");
    getchar();  // To consume the leftover newline character
    fgets(experimentID, sizeof(experimentID), stdin);
    experimentID[strcspn(experimentID, "\n")] = '\0';
    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].experimentID, experimentID) == 0) {
            float sum = 0;
            int count = 0;
            for (int j = 0; j < 10; j++) {
                if ((*arr)[i].results[j] == -1) {
                    break;
                }
                sum += (*arr)[i].results[j];
                count++;
            }
            if (count > 0) {
                printf("Average result for experiment ID '%s' is: %.2f\n", experimentID, sum / count);
            } else {
                printf("No results available for this experiment.\n");
            }
            return;
        }
    }
    printf("Experiment with ID '%s' not found.\n", experimentID);
}

int main() {
    int n = 0;
    int user_input;
    bool is_on = true;
    struct experiment *arr = (struct experiment *)malloc(n * sizeof(struct experiment));

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


    while (is_on) {
        printf("\n1. Log a new Experiment\n2. Update result\n3. Display Experiments\n4. Display average
result\n5. Exit\nEnter your choice: ");
        scanf("%d", &user_input);


        switch (user_input) {
            case 1:
                new_experiment(&arr, &n);
                break;
            case 2:
                update_result(&arr, &n);
                break;
            case 3:
                display_experiments(&arr, &n);
                break;
            case 4:
                average_result(&arr, &n);
                break;
            case 5:
                is_on = false;
                break;
            default:
                printf("Enter a valid option.\n");
        }
    }
    free(arr);
    return 0;
}
```

13.Grant application tracker

```
#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];
};

void add_application(struct GrantApplication **arr, int *n) {
    *arr = realloc(*arr, (*n + 1) * sizeof(struct GrantApplication));
    if (*arr == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    int index = *n;
    printf("Enter Application ID: ");
    fgets((*arr)[index].applicationID, 10, stdin);
    printf("Enter Applicant Name: ");
    fgets((*arr)[index].applicantName, 50, stdin);
    printf("Enter Project Title: ");
    fgets((*arr)[index].projectTitle, 100, stdin);
    printf("Enter Requested Amount: ");
    scanf("%f", &(*arr)[index].requestedAmount);
    getchar();  // Consume newline character left by scanf
    printf("Enter Status: ");
    fgets((*arr)[index].status, 20, stdin);

    (*arr)[index].status[strcspn((*arr)[index].status, "\n")] = '\0';  // Remove newline from status string
    *n += 1;
}

void update_status(struct GrantApplication **arr, int *n) {
    char appID[10];
    printf("Enter the Application ID to update the status: ");
    fgets(appID, 10, stdin);
    appID[strcspn(appID, "\n")] = '\0';

    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].applicationID, appID) == 0) {
            printf("Enter the new status (Submitted, Approved, Rejected): ");
            fgets((*arr)[i].status, 20, stdin);
            (*arr)[i].status[strcspn((*arr)[i].status, "\n")] = '\0';
            printf("Status updated successfully.\n");
            return;
        }
    }
    printf("Application ID not found.\n");
```

```
}


void display_applications_greater_than(struct GrantApplication **arr, int *n, float value) {
  int found = 0;
  for (int i = 0; i < *n; i++) {
    if ((*arr)[i].requestedAmount > value) {
      printf("Application ID: %s\nApplicant: %sProject Title: %sRequested Amount: %.2f\nStatus:
%s\n\n",
           (*arr)[i].applicationID, (*arr)[i].applicantName, (*arr)[i].projectTitle,
           (*arr)[i].requestedAmount, (*arr)[i].status);
      found = 1;
    }
  }
  if (!found) {
    printf("No applications found with requested amount greater than %.2f.\n", value);
  }
}


void display_approved_applications(struct GrantApplication **arr, int *n) {
  int found = 0;
  for (int i = 0; i < *n; i++) {
    if (strcmp((*arr)[i].status, "Approved") == 0) {
      printf("Application ID: %s\nApplicant: %sProject Title: %sRequested Amount: %.2f\nStatus:
%s\n\n",
           (*arr)[i].applicationID, (*arr)[i].applicantName, (*arr)[i].projectTitle,
           (*arr)[i].requestedAmount, (*arr)[i].status);
      found = 1;
    }
  }
```

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

int main() {
    int n = 0;
    struct GrantApplication *arr = (struct GrantApplication *)malloc(n * sizeof(struct GrantApplication));

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

    int choice;
    bool is_on = 1;
    while (is_on) {
        printf("\n1. Add a new grant application\n2. Update application status\n3. Display applications with
requested amount > value\n4. Display approved applications\n5. Exit\n");
        scanf("%d", &choice);
        getchar();  // Consume newline character

        switch (choice) {
            case 1:
                add_application(&arr, &n);
                break;
            case 2:
                update_status(&arr, &n);
                break;
            case 3:
                {
                    float value;
                    printf("Enter the amount value: ");
                    scanf("%f", &value);
                    display_applications_greater_than(&arr, &n, value);
                }
                break;
            case 4:
                display_approved_applications(&arr, &n);
                break;
            case 5:
                is_on = 0;
                break;
            default:
                printf("Invalid option.\n");
        }
    }

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

14. Research collbrator management

```
#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;

};


void add_collaborator(struct Collaborator **arr, int *n) {

    *arr = realloc(*arr, (*n + 1) * sizeof(struct Collaborator));

    if (*arr == NULL) {

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

        exit(1);

    }


    int index = *n;

    printf("Enter Collaborator ID: ");

    fgets((*arr)[index].collaboratorID, 10, stdin);

    printf("Enter Collaborator Name: ");

    fgets((*arr)[index].name, 50, stdin);

    printf("Enter Institution: ");

    fgets((*arr)[index].institution, 50, stdin);

    printf("Enter Expertise Area: ");

    fgets((*arr)[index].expertiseArea, 30, stdin);

    printf("Enter Number of Projects: ");

    scanf("%d", &(*arr)[index].numberOfProjects);

    getchar();  // Consume newline character

    *n += 1;

}
```

```
void update_projects(struct Collaborator **arr, int *n) {
    char collabID[10];
    printf("Enter Collaborator ID to update number of projects: ");
    fgets(collabID, 10, stdin);
    collabID[strcspn(collabID, "\n")] = '\0';

    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].collaboratorID, collabID) == 0) {
            int newProjects;
            printf("Enter the new number of projects: ");
            scanf("%d", &newProjects);
            (*arr)[i].numberOfProjects = newProjects;
            printf("Number of projects updated successfully.\n");
            return;
        }
    }
    printf("Collaborator with ID '%s' not found.\n", collabID);
}

void display_by_institution(struct Collaborator **arr, int *n, const char *institution) {
    int found = 0;
    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].institution, institution) == 0) {
            printf("Collaborator ID: %s\nName: %sInstitution: %sExpertise Area: %sNumber of Projects: 
%d\n\n",
                   (*arr)[i].collaboratorID, (*arr)[i].name, (*arr)[i].institution, (*arr)[i].expertiseArea,
                   (*arr)[i].numberOfProjects);
            found = 1;
        }
    }
    if (!found) {
        printf("No collaborators found from institution '%s'.\n", institution);
    }
}

void find_expertise_area(struct Collaborator **arr, int *n, const char *expertiseArea) {
    int found = 0;
    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].expertiseArea, expertiseArea) == 0) {
            printf("Collaborator ID: %s\nName: %sInstitution: %sExpertise Area: %sNumber of Projects:
%d\n\n",
                   (*arr)[i].collaboratorID, (*arr)[i].name, (*arr)[i].institution, (*arr)[i].expertiseArea,
                   (*arr)[i].numberOfProjects);
            found = 1;
        }
    }
    if (!found) {
        printf("No collaborators found with expertise in '%s'.\n", expertiseArea);
    }
}

int main() {
    int n = 0;
    struct Collaborator *arr = (struct Collaborator *)malloc(n * sizeof(struct Collaborator));

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


    int choice;

    bool is_on = 1;

    while (is_on) {

        printf("\n1. Add a new collaborator\n2. Update number of projects\n3. Display collaborators from a
specific institution\n4. Find collaborators with expertise in a given area\n5. Exit\n");

        scanf("%d", &choice);

        getchar();


        switch (choice) {

            case 1:

                add_collaborator(&arr, &n);

                break;

            case 2:

                update_projects(&arr, &n);

                break;

            case 3:

                {

                    char institution[50];

                    printf("Enter institution name: ");

                    fgets(institution, 50, stdin);

                    institution[strcspn(institution, "\n")] = '\0';  // Remove newline from string

                    display_by_institution(&arr, &n, institution);

                }

                break;

            case 4:

                {

                    char expertiseArea[30];
                    printf("Enter expertise area: ");
                    fgets(expertiseArea, 30, stdin);
                    expertiseArea[strcspn(expertiseArea, "\n")] = '\0';  // Remove newline from string
                    find_expertise_area(&arr, &n, expertiseArea);
                }
                break;
            case 5:
                is_on = 0;
                break;
            default:
                printf("Invalid option.\n");
        }
    }

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

15. Scientific Conference Submission Tracker

```
#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];
};

void add_submission(struct ConferenceSubmission **arr, int *n) {
    *arr = realloc(*arr, (*n + 1) * sizeof(struct ConferenceSubmission));
    if (*arr == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    int index = *n;
    printf("Enter Submission ID: ");
    fgets((*arr)[index].submissionID, 10, stdin);
    printf("Enter Author Name: ");
    fgets((*arr)[index].authorName, 50, stdin);
    printf("Enter Paper Title: ");
    fgets((*arr)[index].paperTitle, 100, stdin);
    printf("Enter Conference Name: ");
    fgets((*arr)[index].conferenceName, 50, stdin);
    printf("Enter Submission Date (DD/MM/YYYY): ");
    fgets((*arr)[index].submissionDate, 11, stdin);
    printf("Enter Status (Pending/Accepted/Rejected): ");
    fgets((*arr)[index].status, 20, stdin);

    // Remove newline characters
    (*arr)[index].submissionID[strcspn((*arr)[index].submissionID, "\n")] = '\0';
    (*arr)[index].authorName[strcspn((*arr)[index].authorName, "\n")] = '\0';
    (*arr)[index].paperTitle[strcspn((*arr)[index].paperTitle, "\n")] = '\0';
    (*arr)[index].conferenceName[strcspn((*arr)[index].conferenceName, "\n")] = '\0';
    (*arr)[index].submissionDate[strcspn((*arr)[index].submissionDate, "\n")] = '\0';
    (*arr)[index].status[strcspn((*arr)[index].status, "\n")] = '\0';

    *n += 1;
    printf("Conference Submission added successfully!\n");
}

void update_status(struct ConferenceSubmission **arr, int *n) {
    char submissionID[10];
    printf("Enter Submission ID to update status: ");
    fgets(submissionID, 10, stdin);
    submissionID[strcspn(submissionID, "\n")] = '\0';

    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].submissionID, submissionID) == 0) {
            printf("Enter new status (Pending/Accepted/Rejected): ");
            fgets((*arr)[i].status, 20, stdin);
            (*arr)[i].status[strcspn((*arr)[i].status, "\n")] = '\0';
            printf("Status updated successfully!\n");
            return;
        }
    }
    printf("Submission with ID '%s' not found.\n", submissionID);
}

void display_conference_submissions(struct ConferenceSubmission **arr, int *n, const char
*conferenceName) {
    int found = 0;
    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].conferenceName, conferenceName) == 0) {
            printf("\nSubmission ID: %s\nAuthor: %s\nPaper Title: %s\nConference: %s\nDate: %s\nStatus:
%s\n",
                   (*arr)[i].submissionID, (*arr)[i].authorName, (*arr)[i].paperTitle,
                   (*arr)[i].conferenceName, (*arr)[i].submissionDate, (*arr)[i].status);
            found = 1;
        }
    }
    if (!found) {
        printf("No submissions found for conference '%s'.\n", conferenceName);
    }
}

void display_author_submissions(struct ConferenceSubmission **arr, int *n, const char *authorName) {
    int found = 0;
    for (int i = 0; i < *n; i++) {
        if (strcmp((*arr)[i].authorName, authorName) == 0) {
            printf("\nSubmission ID: %s\nAuthor: %s\nPaper Title: %s\nConference: %s\nDate: %s\nStatus:
%s\n",
                   (*arr)[i].submissionID, (*arr)[i].authorName, (*arr)[i].paperTitle,
                   (*arr)[i].conferenceName, (*arr)[i].submissionDate, (*arr)[i].status);
            found = 1;
        }
    }
    if (!found) {
        printf("No submissions found by author '%s'.\n", authorName);
    }
}
```

```
int main() {
   int n = 0;
   struct ConferenceSubmission *arr = (struct ConferenceSubmission *)malloc(n * sizeof(struct
ConferenceSubmission));

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

   int choice;
   bool is_on = 1;
   while (is_on) {
      printf("\n1. Add a new conference submission\n2. Update submission status\n3. Display all
submissions to a specific conference\n4. Find and display submissions by a specific author\n5. Exit\n");
      scanf("%d", &choice);
      getchar();  // Consume newline character

      switch (choice) {
         case 1:
            add_submission(&arr, &n);
            break;
         case 2:
            update_status(&arr, &n);
            break;
         case 3:
            {
               char conferenceName[50];
               printf("Enter the conference name: ");
```

```
                    fgets(conferenceName, 50, stdin);
                    conferenceName[strcspn(conferenceName, "\n")] = '\0';  // Remove newline
                    display_conference_submissions(&arr, &n, conferenceName);
                }
                break;
            case 4:
                {
                    char authorName[50];
                    printf("Enter the author name: ");
                    fgets(authorName, 50, stdin);
                    authorName[strcspn(authorName, "\n")] = '\0';  // Remove newline
                    display_author_submissions(&arr, &n, authorName);
                }
                break;
            case 5:
                is_on = 0;
                break;
            default:
                printf("Invalid option.\n");
        }
    }

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