1.

```c
#include <Stdio.h>
struct date{
    int days;
    int months;
    int years;
};


int main(){
    struct date CurrentDate;
    struct date *ptr;
    ptr = &CurrentDate;


    (*ptr).days=22;
    (*ptr).months =11;
    (*ptr).years =2024;

    printf("Todays date is = %d-%d-
%d",(*ptr).days,(*ptr).months,(*ptr).years);

    return 0;
}
```

2.

```c
//Instead of dot operator to store or print data here we are using arrow
operator



#include <Stdio.h>
struct date{
    int days;
    int months;
    int years;
};


int main(){
    struct date CurrentDate;
    struct date *ptr;
    ptr = &CurrentDate;


    (ptr)->days=22;
    (ptr)->months =11;
    (ptr)->years =2024;
```

```
    printf("Todays date is = %d-%d-%d",(ptr)->days,(ptr)->months,(ptr)-
>years);

    return 0;
}
```

3.

```
#include <stdio.h>
struct intPtrs{
    int *p1;
    int *p2;
};
int main(){
    struct intPtrs pointers;
    int i1 =100,i2;
    pointers.p1 =&i1;
    pointers.p2=&i2;


    *pointers.p2=-97;

    printf("i1 = %d , *pointer.p1 = %d \n",i1,*pointers.p1);
    printf("i2 = %d , *pointer.p2 = %d \n",i2,*pointers.p2);


    return 0;
}
```

4.

```
//character array and character pointer

#include <stdio.h>

struct names{
    char first[40];
    char last[40];
};


//This one is more efficient in terms of memory optimization than that of
character array declaration;
struct pNames{
    char *first;
    char *last;

};
```

```c
int main()
{
    struct names CAnames = {"Bhavana","Baiju"};
    struct pNames CPnames = {"Ekansh","Rathore"};


    printf("\n%s\t%s",CAnames.first,CAnames.last);


    printf("\nSize of CAnames = %d",sizeof(CAnames));
    printf("\nSize of CPnames = %d",sizeof(CPnames));



    return 0;
}
```

5.

```c
//Structures as argument to a fn.

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

struct names{
    char first[40];
    char last[40];
};

bool nameComparison(struct names,struct names);


int main()
{
    struct names CAnames = {"Bhavana","Baiju"};
    struct names CPnames = {"Bhavana","Rathore"};

    bool value = nameComparison(CAnames,CPnames);

    printf("\nb = %d",value);


    printf("\nSize of CAnames = %d",sizeof(CAnames));
    printf("\nSize of CPnames = %d",sizeof(CPnames));
```

```
    return 0;
}
bool nameComparison(struct names CAnames,struct names CPnames){
    if(strcmp(CAnames.first,CPnames.first)==0)
    {
        return true;
    }
    else{
        return false;
    }
}
```

6.

```
//Pointers to structures as a fn argument "Method1";

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

struct names{
    char first[40];
    char last[40];
};

bool nameComparison(struct names *,struct names *);


int main()
{
    struct names CAnames = {"Bhavana","Baiju"};
    struct names CPnames = {"Bhavana","Rathore"};


    struct names *ptr1,*ptr2;

    ptr1 = &CAnames;
    ptr2 = &CPnames;

    bool value = nameComparison(ptr1,ptr2);

    printf("\nb = %d",value);


    printf("\nSize of CAnames = %d",sizeof(CAnames));
    printf("\nSize of CPnames = %d",sizeof(CPnames));
```

```
    return 0;
}
bool nameComparison(struct names *ptr1,struct names *ptr2){
    if(strcmp(ptr1->first,ptr2->first)==0)//While using the pointers intead of
direct value use arrow operator instead of dot operator.
    {
        return true;
    }
    else{
        return false;
    }
}
```

7.

```
//Pointers to structures as a fn argument "Method2" .This method is faster
than method 1;
//Time taken to execute is less than that of method 1.
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

struct names{
    char first[40];
    char last[40];
};

bool nameComparison(struct names *,struct names *);


int main()
{
    struct names CAnames = {"Bhavana","Baiju"};
    struct names CPnames = {"Bhavana","Rathore"};


    /*struct names *ptr1,*ptr2;

    ptr1 = &CAnames;
    ptr2 = &CPnames;*/

    bool value = nameComparison(&CAnames,&CPnames);//Passing addre4ss using
"Reference operator i.e &"

    printf("\nb = %d",value);
```

```
    printf("\nSize of CAnames = %d",sizeof(CAnames));
    printf("\nSize of CPnames = %d",sizeof(CPnames));



    return 0;
}
bool nameComparison(struct names *ptr1,struct names *ptr2){
    if(strcmp(ptr1->first,ptr2->first)==0)//While using the pointers intead of
direct value use arrow operator instead of dot operator.
    {
        return true;
    }
    else{
        return false;
    }
}
```

8.

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
struct names{
    char first[40];
    char last[40];
};

bool nameCOmparison(struct names *, struct names *);

int main(){
    clock_t start, end;
    double cpu_used_time;
    start = clock();
    struct names CAnames ={"Abhinav", "Karan"};
    struct names CPnames = {"Abhinav","Karan"};
    //struct names *ptr1, *ptr2;
    //ptr1 = &CAnames;
    //ptr2 = &CPnames;

    bool b = nameCOmparison(&CAnames, &CPnames);
    printf("b = %d",b);
    end = clock();
    cpu_used_time = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("\ncpu_used_time = %f \n",cpu_used_time);
```

```
    return 0;
}

bool nameCOmparison(struct names *p1, struct names *p2){
    if(strcmp(p1->first,p2->first) == 0){
        return true;
    }
    else{
        return false;
    }
}
```

9.

```
/****ASSIGNMENT***

Problem 1: Dynamic Student Record Management


Objective: Manage student records using pointers to structures and dynamically
allocate memory for student names.

**Description**:
1.Define a structure Student with fields:
    ->int roll_no: Roll number
    ->char *name: Pointer to dynamically allocated memory for the student's
name
    ->float marks: Marks obtained
2.Write a program to:
    ->Dynamically allocate memory for n students.
    ->Accept details of each student, dynamically allocating memory for their
names.
    ->Display all student details.
    ->Free all allocated memory before exiting.


*/

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

struct Student{
    int roll_no;
    char *name;
```

```c
    float marks;
};


int main()
{

    int n;

    printf("\nEnter the number of students : ");
    scanf("%d",&n);


    //Dynamically allocating memory for n students using pointers
    struct Student *newStudent = (struct Student *)malloc(n * sizeof(struct
Student));

    if (newStudent == NULL) {
        printf("Memory allocation failed.\n");
        return 1; // Exit if memory allocation fails
    }

    //Collecting and storing the details of n students.
    printf("\nEnter the roll number,name and marks of the students :  \n");
    for(int i = 0;i<n;i++)
    {
        printf("\nStudent %d : \n",i+1);

        printf("\nRoll number : ");
        scanf("%d",&newStudent[i].roll_no);


        printf("\nName : ");
        char temp[50];//Temporary buffer to store the name for further memory
allocation process.
        scanf(" %[^\n]s", temp); // To handle spaces in names
        newStudent[i].name = (char *)malloc((strlen(temp) + 1) *
sizeof(char));
        if (newStudent[i].name == NULL) {
            printf("Memory allocation for name failed.\n");
            return 1; // Exit if memory allocation for name fails
        }


        strcpy(newStudent[i].name,temp);


        printf("\nMarks : ");
        scanf("%f",&newStudent[i].marks);
```

```
    }


    //Display the details of Student in the console
    printf("\nDetails of students : \n");

    for(int i=0;i<n;i++){
        printf("\nRoll_no : %d\tName : %s\tMarks :
%f",newStudent[i].roll_no,newStudent[i].name,newStudent[i].marks);
    }


    //De-allocating or removing all allocated memory
    for(int i=0;i<n;i++)
    {
        free(newStudent[i].name);
    }
    free(newStudent);


    return 0;


}
```

10.

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

struct Book {
    char *title;   // Pointer for dynamically allocated memory for the title
    char *author;  // Pointer for dynamically allocated memory for the author
    int *copies;   // Pointer for dynamically allocated memory for the number
of copies
};

int main() {
    int n;

    // Accept the number of books
    printf("\nEnter the number of books: ");
    scanf("%d", &n);

    // Dynamically allocate memory for n books
    struct Book *newBook = (struct Book *)malloc(n * sizeof(struct Book));
```

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

    // Collect details for each book
    printf("\nEnter the details of the books:\n");
    for (int i = 0; i < n; i++) {
        printf("\nBook %d:\n", i + 1);

        // Accept title
        printf("Title: ");
        char tempTitle[100]; // Temporary buffer
        scanf(" %[^\n]s", tempTitle);
        newBook[i].title = (char *)malloc((strlen(tempTitle) + 1) *
sizeof(char));
        if (newBook[i].title == NULL) {
            printf("Memory allocation for title failed.\n");
            return 1;
        }
        strcpy(newBook[i].title, tempTitle);

        // Accept author
        printf("Author: ");
        char tempAuthor[100]; // Temporary buffer
        scanf(" %[^\n]s", tempAuthor);
        newBook[i].author = (char *)malloc((strlen(tempAuthor) + 1) *
sizeof(char));
        if (newBook[i].author == NULL) {
            printf("Memory allocation for author failed.\n");
            return 1;
        }
        strcpy(newBook[i].author, tempAuthor);

        // Accept number of copies
        printf("Copies: ");
        newBook[i].copies = (int *)malloc(sizeof(int));
        if (newBook[i].copies == NULL) {
            printf("Memory allocation for copies failed.\n");
            return 1;
        }
        scanf("%d", newBook[i].copies);
    }

    // Display book details
    printf("\nDetails of Books:\n");
    for (int i = 0; i < n; i++) {
        printf("\nBook %d:\n", i + 1);
```

```
        printf("Title: %s\n", newBook[i].title); // Prints the entire string
        printf("Author: %s\n", newBook[i].author); // Prints the entire string
        printf("Copies: %d\n", *newBook[i].copies);
    }

    // Free allocated memory
    for (int i = 0; i < n; i++) {
        free(newBook[i].title);
        free(newBook[i].author);
        free(newBook[i].copies);
    }
    free(newBook);

    return 0;
}
```

11.

```
/*
Problem 5: Employee Tax Calculation

Objective: Calculate income tax for an employee based on their salary by
passing a structure to a function.

Description:
1.  Define a structure Employee with fields:
    o   char name[50]: Employee name
    o   int emp_id: Employee ID
    o   float salary: Employee salary
    o   float tax: Tax to be calculated (initialized to 0)
2.  Write a function to:
    o   Calculate tax based on salary slabs (e.g., 10% for salaries below
$50,000, 20% otherwise).
    o   Modify the tax field of the structure.
3.  Pass the structure by reference to the function and display the updated
tax in main.




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

```c
struct Employee
{
    int emp_id;
    float tax;
    char name[50];
    float salary;
};
void calcTax(struct Employee *,int);


int main(){

     int n;

    printf("\nEnter the number of employees: ");
    scanf("%d", &n);

    // Dynamically allocate memory for n employees
    struct Employee *emp = (struct Employee *)malloc(n * sizeof(struct
Employee));
    if (emp == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    // Collect employee details
    printf("\nEnter the details of employees:\n");
    for (int i = 0; i < n; i++) {
        printf("\nEmployee %d:\n", i + 1);

        printf("Name: ");
        scanf(" %[^\n]s", emp[i].name); // Read string with spaces

        printf("Employee ID: ");
        scanf("%d", &emp[i].emp_id);

        printf("Salary (in $): ");
        scanf("%f", &emp[i].salary);

        emp[i].tax = 0; // Initialize tax to 0
    }

    // Calculate tax for each employee
    calcTax(emp, n);

    // Display employee details
    printf("\nEmployee Details:\n");
    for (int i = 0; i < n; i++) {
        printf("\nEmployee %d:\n", i + 1);
```

```
        printf("Name: %s\n", emp[i].name);
        printf("Employee ID: %d\n", emp[i].emp_id);
        printf("Salary: $%.2f\n", emp[i].salary);
        printf("Tax: $%.2f\n", emp[i].tax);
    }

    // Free allocated memory
    free(emp);

    return 0;

}
void calcTax(struct Employee *e,int num)
{
    for(int i=0;i<num;i++)
    {
        if(e[i].salary<50000)
        {
            e[i].tax = (e[i].salary)*(15.0/100);
        }
        else{
            e[i].tax = (e[i].salary)*(20.0/100);
        }
    }
}
```

12.

```
/*
Problem 1: Complex Number Operations

Objective: Perform addition and multiplication of two complex numbers using
structures passed to functions.

Description:
1.  Define a structure Complex with fields:
    o   float real: Real part of the complex number
    o   float imag: Imaginary part of the complex number
2.  Write functions to:
    o   Add two complex numbers and return the result.
    o   Multiply two complex numbers and return the result.
3.  Pass the structures as arguments to these functions and display the
results.

*/

#include <stdio.h>

struct Complex
```

```c
{
    //Real part of the complex number
    float real;

    //Imaginary part of the complex number
    float imag;
};

int addComplex(struct Complex,struct Complex);
int mulComplex(struct Complex,struct Complex);

int main()
{
    struct Complex num1 = {3,4};
    struct Complex num2 = {5,6};

    addComplex(num1,num2);


    mulComplex(num1,num2);

    return 0;

}

int addComplex(struct Complex num1,struct Complex num2)
{
    printf("\nThe Result obtained after adding the two given complex numbers :
");
    float r1 = num1.real + num2.real;//Adds the real parts
    float r2 = num1.imag + num2.imag;//Adds the imaginary part
    printf(" %.1f + i(%.1f)",r1,r2);
    return 0;
}


//z1 z2=(a1+ib1)(a2+ib2)
//z1 z2 = (a1 a2 – b1 b2 ) + i(a1 b2 + a2 b1 )
int mulComplex(struct Complex num1,struct Complex num2){
    float r1 = ((num1.real)*(num2.real))-((num1.imag)*(num2.imag));
    float r2 = ((num1.real)*(num2.imag))+((num2.real)*(num1.imag));
    printf("\nThe Result obtained after multiplying the two given complex
numbers : ");
    printf(" %.1f + i(%.1f)",r1,r2);
    return 0;
}
```

13.

```
/*
Problem 2: Rectangle Area and Perimeter Calculator

Objective: Calculate the area and perimeter of a rectangle by passing a
structure to functions.

Description:
1.  Define a structure Rectangle with fields:
    o   float length: Length of the rectangle
    o   float width: Width of the rectangle
2.  Write functions to:
    o   Calculate and return the area of the rectangle.
    o   Calculate and return the perimeter of the rectangle.
3.  Pass the structure to these functions by value and display the results in
main.



*/

#include <stdio.h>

struct Rectangle {
    float length;
    float width;
};

int areaRec(struct Rectangle);
int periRec(struct Rectangle);

int main()
{
    struct Rectangle lb = {4,8};


    areaRec(lb);

    periRec(lb);

    return 0;
}
int areaRec(struct Rectangle lb)
{
    float r = ((lb.length)*(lb.width));
    printf("\nArea of the Rectangle = %.1f",r);
    return 0;
}
```

```c
int periRec(struct Rectangle lb)
{
    float r = (2*(lb.length)+(lb.width));
    printf("\nPerimeter of Rectangle = %.1f",r);
    return 0;
}
```

14.

```c
/*
Problem 3: Student Grade Calculation

Objective: Calculate and assign grades to students based on their marks by
passing a structure to a function.

Description:
1.  Define a structure Student with fields:
    o   char name[50]: Name of the student
    o   int roll_no: Roll number
    o   float marks[5]: Marks in 5 subjects
    o   char grade: Grade assigned to the student
2.  Write a function to:
    o   Calculate the average marks and assign a grade (A, B, etc.) based on
predefined criteria.
3.  Pass the structure by reference to the function and modify the grade
field.


*/
#include <stdio.h>
#include <stdlib.h>

struct Student
{
    char name[50];
    int roll_no;
    float marks[5];
    char grade;
};

int avg(struct Student *,int);

int main()
{

    int n;
    printf("\nEnter the number of Students :  ");
```

```c
    scanf("%d",&n);
    //Dynamically allocating memory for n students
    struct Student *stud = (struct Student *)malloc(n * sizeof(struct
Student));
    if (stud == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    printf("\nEnter the details of the students : ");

    for(int i=0;i<n;i++)
    {
        printf("\nStudent %d :\n",i+1);


        printf("Student Name: ");
        scanf(" %[^\n]s", stud[i].name); // Read string with spaces

        printf("\nRoll number : ");
        scanf("%d",&stud[i].roll_no);

        printf("\nEnter the marks of all 5 Subjects : ");
        for(int j=0;j<5;j++)
        {
            scanf("%f",&stud[i].marks[j]);
        }

    }
    avg(stud,n);


    free(stud);
    return 0;


}
int avg(struct Student *stud,int n)
{
    for (int i = 0; i < n; i++) {
        float sum = 0;
        for (int j = 0; j < 5; j++) {
            sum += stud[i].marks[j];
        }
        float average = sum / 5;

        // Assign grades based on average marks
        if (average > 70 && average <= 80) {
            stud[i].grade = 'A';
```

```
        } else if (average > 60 && average <= 70) {
            stud[i].grade = 'B';
        } else if (average > 50 && average <= 60) {
            stud[i].grade = 'C';
        } else if (average > 40 && average <= 50) {
            stud[i].grade = 'D';
        } else {
            stud[i].grade = 'F';
        }
    }
    for(int i=0;i<n;i++)
    {
        printf("\nName : %s\nRoll_No : %d\nMarks
:",stud[i].name,stud[i].roll_no);
        for(int j=0;j<5;j++)
        {
            printf(" %.2f ",stud[i].marks[j]);
        }
        printf("\nGrade : %c",stud[i].grade);
    }
    return 0;
}
```

15.

```
/*
Problem 4: Point Operations in 2D Space

Objective: Calculate the distance between two points and check if a point lies
within a circle using structures.

Description:
1.  Define a structure Point with fields:
    o   float x: X-coordinate of the point
    o   float y: Y-coordinate of the point
2.  Write functions to:
    o   Calculate the distance between two points.
    o   Check if a given point lies inside a circle of a specified radius
(center at origin).
3.  Pass the Point structure to these functions and display the results.



*/

#include <stdio.h>
```

```c
#include <math.h>

// Define the Point structure
struct Point {
    float x; // X-coordinate
    float y; // Y-coordinate
};

// Function to calculate the distance between two points
float calculateDistance(struct Point p1, struct Point p2) {
    return sqrt(pow((p2.x - p1.x), 2) + pow((p2.y - p1.y), 2));
}

// Function to check if a point lies within a circle
int isPointInCircle(struct Point p, float radius) {
    float distanceFromOrigin = sqrt(pow(p.x, 2) + pow(p.y, 2));
    return distanceFromOrigin <= radius;
}

int main() {
    struct Point p1, p2;
    float radius;

    // Input for two points
    printf("Enter the coordinates of Point 1 (x y): ");
    scanf("%f %f", &p1.x, &p1.y);

    printf("Enter the coordinates of Point 2 (x y): ");
    scanf("%f %f", &p2.x, &p2.y);

    // Input for radius of the circle
    printf("Enter the radius of the circle: ");
    scanf("%f", &radius);

    // Calculate the distance between the two points
    float distance = calculateDistance(p1, p2);
    printf("\nDistance between Point 1 and Point 2: %.2f\n", distance);

    // Check if Point 1 is inside the circle
    if (isPointInCircle(p1, radius)) {
        printf("Point 1 (%.2f, %.2f) lies within the circle of radius %.2f
centered at the origin.\n", p1.x, p1.y, radius);
    } else {
        printf("Point 1 (%.2f, %.2f) does not lie within the circle of radius
%.2f centered at the origin.\n", p1.x, p1.y, radius);
    }

    // Check if Point 2 is inside the circle
```

```
    if (isPointInCircle(p2, radius)) {
        printf("Point 2 (%.2f, %.2f) lies within the circle of radius %.2f
centered at the origin.\n", p2.x, p2.y, radius);
    } else {
        printf("Point 2 (%.2f, %.2f) does not lie within the circle of radius
%.2f centered at the origin.\n", p2.x, p2.y, radius);
    }

    return 0;
}
```

16.

```
/*
Problem Statement: Vehicle Service Center Management

Objective: Build a system to manage vehicle servicing records using nested
structures.

Description:
1.  Define a structure Vehicle with fields:
    o   char license_plate[15]: Vehicle's license plate number
    o   char owner_name[50]: Owner's name
    o   char vehicle_type[20]: Type of vehicle (e.g., car, bike)
2.  Define a nested structure Service inside Vehicle with fields:
    o   char service_type[30]: Type of service performed
    o   float cost: Cost of the service
    o   char service_date[12]: Date of service
3.  Implement the following features:
    o   Add a vehicle to the service center record.
    o   Update the service history for a vehicle.
    o   Display the service details of a specific vehicle.
    o   Generate and display a summary report of all vehicles serviced,
including total revenue.


*/
#include <stdio.h>
#include <string.h>



// Define the structures
struct Service {
    char service_type[30];
    float cost;
```

```
    char service_date[12];
};

struct Vehicle {
    char license_plate[15];
    char owner_name[50];
    char vehicle_type[20];
    int service_count;
    struct Service services[10];//nested structure
};

// Function prototypes
int addVehicle(struct Vehicle vehicles[], int vehicle_count);
void updateService(struct Vehicle vehicles[], int vehicle_count);
void displayVehicleService(struct Vehicle vehicles[], int vehicle_count);
void generateSummaryReport(struct Vehicle vehicles[], int vehicle_count);

int main() {
    struct Vehicle vehicles[100];
    int vehicle_count = 0;
    int choice;

    do {
        printf("\nVehicle Service Center Management\n");
        printf("1. Add a Vehicle\n");
        printf("2. Update Service History\n");
        printf("3. Display Service Details of a Vehicle\n");
        printf("4. Generate Summary Report\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                vehicle_count = addVehicle(vehicles, vehicle_count);
                break;
            case 2:
                updateService(vehicles, vehicle_count);
                break;
            case 3:
                displayVehicleService(vehicles, vehicle_count);
                break;
            case 4:
                generateSummaryReport(vehicles, vehicle_count);
                break;
            case 5:
                printf("Exiting the program.\n");
                break;
```

```
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}

// Function to add a vehicle to the service center record
int addVehicle(struct Vehicle vehicles[], int vehicle_count) {
    if (vehicle_count >= 100) {
        printf("Maximum vehicle limit reached. Cannot add more vehicles.\n");
        return vehicle_count;
    }

    printf("\nEnter Vehicle Details:\n");
    printf("License Plate: ");
    scanf(" %s", vehicles[vehicle_count].license_plate);
    printf("Owner Name: ");
    scanf(" %[^\n]s", vehicles[vehicle_count].owner_name);
    printf("Vehicle Type (e.g., car, bike): ");
    scanf(" %s", vehicles[vehicle_count].vehicle_type);

    vehicles[vehicle_count].service_count = 0;
    printf("Vehicle added successfully!\n");
    return vehicle_count + 1;
}

// Function to update the service history for a vehicle
void updateService(struct Vehicle vehicles[], int vehicle_count) {
    char license_plate[15];
    int found = 0;

    printf("\nEnter License Plate of the Vehicle: ");
    scanf(" %s", license_plate);

    for (int i = 0; i < vehicle_count; i++) {
        if (strcmp(vehicles[i].license_plate, license_plate) == 0) {
            if (vehicles[i].service_count >= 10) {
                printf("Maximum service history reached for this vehicle.\n");
                return;
            }

            int sc = vehicles[i].service_count;
            printf("Enter Service Details:\n");
            printf("Service Type: ");
            scanf(" %[^\n]s", vehicles[i].services[sc].service_type);
            printf("Service Cost: ");
```

```
            scanf("%f", &vehicles[i].services[sc].cost);
            printf("Service Date (DD/MM/YYYY): ");
            scanf(" %s", vehicles[i].services[sc].service_date);

            vehicles[i].service_count++;
            printf("Service details updated successfully!\n");
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Vehicle with license plate %s not found.\n", license_plate);
    }
}

// Function to display the service details of a specific vehicle
void displayVehicleService(struct Vehicle vehicles[], int vehicle_count) {
    char license_plate[15];
    int found = 0;

    printf("\nEnter License Plate of the Vehicle: ");
    scanf(" %s", license_plate);

    for (int i = 0; i < vehicle_count; i++) {
        if (strcmp(vehicles[i].license_plate, license_plate) == 0) {
            printf("\nService Details for Vehicle - %s:\n", license_plate);
            printf("Owner: %s\n", vehicles[i].owner_name);
            printf("Vehicle Type: %s\n", vehicles[i].vehicle_type);
            printf("Service History:\n");

            for (int j = 0; j < vehicles[i].service_count; j++) {
                printf("Service %d:\n", j + 1);
                printf("  Service Type: %s\n",
vehicles[i].services[j].service_type);
                printf("  Cost: $%.2f\n", vehicles[i].services[j].cost);
                printf("  Date: %s\n", vehicles[i].services[j].service_date);
            }

            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Vehicle with license plate %s not found.\n", license_plate);
    }
}
```

```c
// Function to generate and display a summary report
void generateSummaryReport(struct Vehicle vehicles[], int vehicle_count) {
    float total_revenue = 0;

    printf("\nSummary Report:\n");
    for (int i = 0; i < vehicle_count; i++) {
        printf("Vehicle - %s (Owner: %s, Type: %s)\n",
vehicles[i].license_plate, vehicles[i].owner_name, vehicles[i].vehicle_type);
        for (int j = 0; j < vehicles[i].service_count; j++) {
            printf("  Service Type: %s, Cost: $%.2f, Date: %s\n",
                   vehicles[i].services[j].service_type,
                   vehicles[i].services[j].cost,
                   vehicles[i].services[j].service_date);
            total_revenue += vehicles[i].services[j].cost;
        }
    }

    printf("\nTotal Revenue: $%.2f\n", total_revenue);
}
```