1. **Library System with Dynamic Allocation Objective: Manage a library system where book details are dynamically stored using pointers inside a structure. Description: Define a structure Book with fields: char *title: Pointer to dynamically allocated memory for the book's title char *author: Pointer to dynamically allocated memory for the author's name int *copies: Pointer to the number of available copies (stored dynamically) Write a program to: Dynamically allocate memory for n books. Accept and display book details. Update the number of copies of a specific book. Free all allocated memory before exiting.**

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

typedef struct {
  char *title;
  char *author;
  int *copies;
} Book;

void acceptBooks(Book *books, int n) {
  for (int i = 0; i < n; i++) {
    char temp[100];
    printf("\nEnter details for book %d:\n", i + 1);

    printf("Enter title: ");
    scanf(" %[^\n]s", temp);
    books[i].title = (char *)malloc(strlen(temp) + 1);
    strcpy(books[i].title, temp);

    printf("Enter author: ");
    scanf(" %[^\n]s", temp);
    books[i].author = (char *)malloc(strlen(temp) + 1);
    strcpy(books[i].author, temp);

    books[i].copies = (int *)malloc(sizeof(int));
    printf("Enter number of copies: ");
    scanf("%d", books[i].copies);
  }
}
```

```
void displayBooks(Book *books, int n) {
   printf("\nBook Details:\n");
   for (int i = 0; i < n; i++) {
      printf("\nBook %d:\n", i + 1);
      printf("Title: %s\n", books[i].title);
      printf("Author: %s\n", books[i].author);
      printf("Copies: %d\n", *books[i].copies);
   }
}

void updateCopies(Book *books, int n) {
   char title[100];
   int newCopies;
   printf("\nEnter the title of the book to update copies: ");
   scanf(" %[^\n]s", title);

   for (int i = 0; i < n; i++) {
      if (strcmp(books[i].title, title) == 0) {
         printf("Enter new number of copies: ");
         scanf("%d", &newCopies);
         *books[i].copies = newCopies;
         printf("Updated successfully.\n");
         return;
      }
   }
   printf("Book not found.\n");
}

void freeBooks(Book *books, int n) {
   for (int i = 0; i < n; i++) {
      free(books[i].title);
      free(books[i].author);
      free(books[i].copies);
   }
}

int main() {
   int n;
   printf("Enter the number of books: ");
   scanf("%d", &n);
```

```
    Book *books = (Book *)malloc(n * sizeof(Book));

    acceptBooks(books, n);
    displayBooks(books, n);
    updateCopies(books, n);
    displayBooks(books, n);

    freeBooks(books);
    free(books);

    return 0;
}
```

Problem 2: Dynamic Student Record Management
Objective: Manage student records using pointers to structures and dynamically allocate memory for student names.
Description:
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
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>

typedef struct {
  int roll_no;
  char *name;
  float marks;
} Student;

void acceptStudents(Student *students, int n) {
  for (int i = 0; i < n; i++) {
    char temp[100];
    printf("\nEnter details for student %d:\n", i + 1);

    printf("Enter roll number: ");
    scanf("%d", &students[i].roll_no);
```

```
        printf("Enter name: ");
        scanf(" %[^\n]s", temp);
        students[i].name = (char *)malloc(strlen(temp) + 1);
        strcpy(students[i].name, temp);

        printf("Enter marks: ");
        scanf("%f", &students[i].marks);
    }
}

void displayStudents(Student *students, int n) {
    printf("\nStudent Details:\n");
    for (int i = 0; i < n; i++) {
        printf("\nStudent %d:\n", i + 1);
        printf("Roll Number: %d\n", students[i].roll_no);
        printf("Name: %s\n", students[i].name);
        printf("Marks: %.2f\n", students[i].marks);
    }
}

void freeStudents(Student *students, int n) {
    for (int i = 0; i < n; i++) {
        free(students[i].name);
    }
}

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

    Student *students = (Student *)malloc(n * sizeof(Student));

    acceptStudents(students, n);
    displayStudents(students, n);

    freeStudents(students);
    free(students);

    return 0;
}
```

Problem 3: Complex Number Operations Objective: Perform addition and multiplication of two complex numbers using structures passed to functions. Description: Define a structure Complex with fields: float real: Real part of the

complex number float imag: Imaginary part of the complex number Write functions to: Add two complex numbers and return the result. Multiply two complex numbers and return the result. Pass the structures as arguments to these functions and display the results.

```
#include <stdio.h>

typedef struct {
  float real;
  float imag;
} Complex;

Complex addComplex(Complex c1, Complex c2) {
  Complex result;
  result.real = c1.real + c2.real;
  result.imag = c1.imag + c2.imag;
  return result;
}

Complex multiplyComplex(Complex c1, Complex c2) {
  Complex result;
  result.real = c1.real * c2.real - c1.imag * c2.imag;
  result.imag = c1.real * c2.imag + c1.imag * c2.real;
  return result;
}

int main() {
  Complex c1 = {3.0, 4.0}, c2 = {1.0, 2.0};

  Complex sum = addComplex(c1, c2);
  Complex product = multiplyComplex(c1, c2);

  printf("Sum: %.2f + %.2fi\n", sum.real, sum.imag);
  printf("Product: %.2f + %.2fi\n", product.real, product.imag);

  return 0;
}
```

Problem 4: Rectangle Area and Perimeter Calculator Objective: Calculate the area and perimeter of a rectangle by passing a structure to functions. Description: Define a structure Rectangle with fields: float length: Length of the rectangle float width: Width of the rectangle Write functions to: Calculate and return the area of the rectangle. Calculate and return the perimeter of the rectangle. Pass the structure to these functions by value and display the results in main.

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

```
typedef struct {
   float length;
   float width;
} Rectangle;

float calculateArea(Rectangle rect) {
   return rect.length * rect.width;
}

float calculatePerimeter(Rectangle rect) {
   return 2 * (rect.length + rect.width);
}

int main() {
   Rectangle rect = {5.0, 3.0};

   float area = calculateArea(rect);
   float perimeter = calculatePerimeter(rect);

   printf("Area: %.2f\n", area);
   printf("Perimeter: %.2f\n", perimeter);

   return 0;
}
```

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

```
#include <stdio.h>

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

void calculateGrade(Student *student) {
   float sum = 0.0;
   for (int i = 0; i < 5; i++) {
```

```
        sum += student->marks[i];
    }
    float average = sum / 5;

    if (average >= 90) student->grade = 'A';
    else if (average >= 75) student->grade = 'B';
    else if (average >= 50) student->grade = 'C';
    else student->grade = 'D';
}

int main() {
    Student student = {"Alice", 101, {85, 92, 78, 88, 90}, ' '};

    calculateGrade(&student);

    printf("Student: %s\n", student.name);
    printf("Grade: %c\n", student.grade);

    return 0;
}
```

Problem 6: Point Operations in 2D Space Objective: Calculate the distance between two points and check if a point lies within a circle using structures. Description: Define a structure Point with fields: float x: X-coordinate of the point float y: Y-coordinate of the point Write functions to: Calculate the distance between two points. Check if a given point lies inside a circle of a specified radius (center at origin). Pass the Point structure to these functions and display the results.

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

typedef struct {
    float x;
    float y;
} Point;

float calculateDistance(Point p1, Point p2) {
    return sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
}

int isInsideCircle(Point p, float radius) {
    return sqrt(p.x * p.x + p.y * p.y) <= radius;
}
```

```
int main() {
   Point p1 = {1.0, 2.0}, p2 = {4.0, 6.0};
   float radius = 5.0;

   float distance = calculateDistance(p1, p2);
   printf("Distance: %.2f\n", distance);

   Point p3 = {3.0, 4.0};
   if (isInsideCircle(p3, radius)) {
      printf("Point lies inside the circle.\n");
   } else {
      printf("Point lies outside the circle.\n");
   }

   return 0;
}
```

Problem 7: Employee Tax Calculation Objective: Calculate income tax for an employee based on their salary by passing a structure to a function. Description: Define a structure Employee with fields: char name[50]: Employee name int emp_id: Employee ID float salary: Employee salary float tax: Tax to be calculated (initialized to 0) Write a function to: Calculate tax based on salary slabs (e.g., 10% for salaries below $50,000, 20% otherwise). Modify the tax field of the structure. Pass the structure by reference to the function and display the updated tax in main. has context menu

```
#include <stdio.h>

typedef struct {
   char name[50];
   int emp_id;
   float salary;
   float tax;
} Employee;

void calculateTax(Employee *emp) {
   if (emp->salary < 50000) {
      emp->tax = emp->salary * 0.10;
   } else {
      emp->tax = emp->salary * 0.20;
   }
}

int main() {
   Employee emp = {"John Doe", 1234, 60000, 0};
```

```
    calculateTax(&emp);

    printf("Employee: %s\n", emp.name);
    printf("Tax: %.2f\n", emp.tax);

    return 0;
}
```

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

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

#define MAX_VEHICLES 10
#define MAX_SERVICES 5

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];
    struct Service services[MAX_SERVICES];
    int service_count;
};

void addVehicle(struct Vehicle vehicles[], int *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: ");
    scanf(" %[^\n]s", vehicles[*vehicle_count].vehicle_type);
    vehicles[*vehicle_count].service_count = 0;
    (*vehicle_count)++;
}

void updateServiceHistory(struct Vehicle vehicles[], int vehicle_count) {
    char license_plate[15];
    printf("\nEnter the vehicle license plate to update service history: ");
    scanf("%s", license_plate);

    int found = 0;
    for (int i = 0; i < vehicle_count; i++) {
        if (strcmp(vehicles[i].license_plate, license_plate) == 0) {
            found = 1;
            if (vehicles[i].service_count < MAX_SERVICES) {
                printf("Enter service type: ");
                scanf(" %[^\n]s", vehicles[i].services[vehicles[i].service_count].service_type);
                printf("Enter service cost: ");
                scanf("%f", &vehicles[i].services[vehicles[i].service_count].cost);
                printf("Enter service date (DD/MM/YYYY): ");
                scanf("%s", vehicles[i].services[vehicles[i].service_count].service_date);
                vehicles[i].service_count++;
                printf("Service history updated successfully!\n");
            } else {
                printf("Service history is full for this vehicle.\n");
            }
            break;
        }
    }

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

void displayVehicleServiceDetails(struct Vehicle vehicles[], int vehicle_count) {
    char license_plate[15];
    printf("\nEnter the vehicle license plate to display service details: ");
    scanf("%s", license_plate);

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

```
        if (strcmp(vehicles[i].license_plate, license_plate) == 0) {
            found = 1;
            printf("\nService details for vehicle %s:\n", vehicles[i].license_plate);
            for (int j = 0; j < vehicles[i].service_count; j++) {
                printf("Service Type: %s\n", vehicles[i].services[j].service_type);
                printf("Service Cost: %.2f\n", vehicles[i].services[j].cost);
                printf("Service Date: %s\n", vehicles[i].services[j].service_date);
                printf("\n");
            }
            break;
        }
    }

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

void generateSummaryReport(struct Vehicle vehicles[], int vehicle_count) {
    float total_revenue = 0;
    printf("\nSummary Report of All Vehicles Serviced:\n");
    for (int i = 0; i < vehicle_count; i++) {
        printf("Vehicle License Plate: %s\n", vehicles[i].license_plate);
        printf("Owner Name: %s\n", vehicles[i].owner_name);
        printf("Vehicle Type: %s\n", vehicles[i].vehicle_type);

        for (int j = 0; j < vehicles[i].service_count; j++) {
            total_revenue += vehicles[i].services[j].cost;
        }

        printf("\n");
    }

    printf("Total Revenue Generated: %.2f\n", total_revenue);
}

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

    while (1) {
        printf("\nVehicle Service Center Management\n");
        printf("1. Add Vehicle\n");
        printf("2. Update Service History\n");
        printf("3. Display Vehicle Service Details\n");
        printf("4. Generate Summary Report\n");
```

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

        switch (choice) {
            case 1:
                addVehicle(vehicles, &vehicle_count);
                break;
            case 2:
                updateServiceHistory(vehicles, vehicle_count);
                break;
            case 3:
                displayVehicleServiceDetails(vehicles, vehicle_count);
                break;
            case 4:
                generateSummaryReport(vehicles, vehicle_count);
                break;
            case 5:
                printf("Exiting the program.\n");
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```