Problem 1: 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>

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

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

   students = (struct Student *)malloc(n * sizeof(struct Student));
   if (students == NULL)
   {
      printf("Memory allocation failed.\n");
      return 1;
   }
   for(int i=0;i<n;i++)
   {
      printf("\nEnter student details :\n");

      printf("\nRoll no:");
      scanf("%d",&students[i].roll_no);

      char temp[100];
      printf("\nName:");
      getchar();
      scanf("%[^\n]",temp);

      students[i].name = (char *)malloc(strlen(temp)+1);
      if (students[i].name == NULL)
      {
         printf("Memory allocation failed.\n");
         return 1;
```

```
        }
        strcpy(students[i].name, temp);
        printf("Marks: ");
        scanf("%f", &students[i].marks);
    }

    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);
        printf("*------------------------------------------------------*\n");
    }


    for (int i = 0; i < n; i++)
    {
        free(students[i].name);
    }
    free(students);
}
```

Problem 2: 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>

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

};
void add_book_details(struct Book *books,int n);
void disp_details(struct Book *books,int n);
void update_detail(struct Book *books,int n);
void freeMemory(struct Book *books, int n);
```

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

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

    add_book_details(books,n);
    disp_details(books,n);
    update_detail(books,n);
    freeMemory(books, n);
}


void add_book_details(struct Book *books,int n)
{
     for (int i = 0; i < n; i++)
     {
        char temp[100];

        printf("\nEnter details for book %d:\n", i + 1);

        printf("Title: ");
        scanf(" %[^\n]", temp);
        books[i].title = (char *)malloc(strlen(temp) + 1);
        if (books[i].title == NULL)
        {
            printf("Memory allocation failed.\n");
            freeMemory(books, i);
            return;
        }
        strcpy(books[i].title, temp);

        printf("Author: ");
        scanf(" %[^\n]", temp);
        books[i].author = (char *)malloc(strlen(temp) + 1);
        if (books[i].author == NULL)
        {
            printf("Memory allocation failed.\n");
            freeMemory(books, i);
            return;
        }
        strcpy(books[i].author, temp);

        books[i].copies = (int *)malloc(sizeof(int));
        if (books[i].copies == NULL)
        {
            printf("Memory allocation failed.\n");
            freeMemory(books, i);
            return;
        }
        printf("Number of copies: ");
        scanf("%d", books[i].copies);
    }
}


void disp_details(struct 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("Available Copies: %d\n", *(books[i].copies));
    }
}

void update_detail(struct Book *books,int n)
{
    char temp[100];
    int found =0;
    printf("Enter the name of book to be updated:");
    getchar();
    scanf("%[^\n]",temp);
    for (int i = 0; i < n; i++)
    {
        if(strcmp(books[i].title,temp) == 0)
        {
            int newCopies;
            found = 1;
            printf("Enter the new number of copies: ");
            scanf("%d", &newCopies);
            *(books[i].copies) = newCopies;
            printf("\nUpdated details for book :%s\n", temp);
            printf("Title: %s\n", books[i].title);
            printf("Author: %s\n", books[i].author);
            printf("Available Copies: %d\n", *(books[i].copies));
        }
    }
    if(!found)
    {
        printf("\n No such book found.\n");
    }
}

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

Problem 3: 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:
    1. float real: Real part of the complex number
    2. float imag: Imaginary part of the complex number
2. Write functions to:

    1. Add two complex numbers and return the result.
    2. 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 {

        float real;

        float imag;



};



struct Complex addComplex(struct Complex c1, struct Complex c2);

struct Complex multiplyComplex(struct Complex c1, struct Complex c2);

int op;



int main()

{

  do{
        struct Complex com1,com2,sum,prod;


        printf("Enter the real and imaginary parts of first complex number:");

        scanf("%f %f",&com1.real,&com1.imag);


        printf("\nEnter the real and imaginary parts of second complex number:");

        scanf("%f %f",&com2.real,&com2.imag);




        printf("Select option:\n1->To add complex number\n2->To multiply complex
number\n");

        scanf("%d",&op);


        switch(op)

        {

        case 1:

                sum = addComplex(com1, com2);

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

                break;

        case 2:

                prod = multiplyComplex(com1, com2);

                printf("Product: %.2f * %.2fi\n", prod.real, prod.imag);

                break;

        case 3:

                printf("\nExiting!\n");

                break;
```

```
                        default:

                        printf("Invalid Option!\n");



            }



  }while(op != 3);

            return 0;

}


struct Complex addComplex(struct Complex c1, struct Complex c2)

{

            struct Complex result;

            result.real = c1.real + c2.real;

            result.imag = c1.imag + c2.imag;

            return result;

}





struct Complex multiplyComplex(struct Complex c1, struct Complex c2)

{

            struct Complex result;

            result.real = (c1.real * c2.real) - (c1.imag * c2.imag);

            result.imag = (c1.real * c2.imag) + (c1.imag * c2.real);

            return result;
```

```
}
```

Problem 4: 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:
    1. float length: Length of the rectangle
    2. float width: Width of the rectangle
2. Write functions to:
    1. Calculate and return the area of the rectangle.
    2. 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>

typedef struct{

    float length;

    float width;


}Rectangle;


float area_of_rect(Rectangle rect);

float peri_of_rect(Rectangle rect);


int main()

{

    Rectangle rectangle;

    float area,perimeter;
    printf("Enter the length of rectangle:");
    scanf("%f",&rectangle.length);

    printf("Enter the width of rectangle:");
    scanf("%f",&rectangle.width);

    area = area_of_rect(rectangle);

    perimeter = peri_of_rect(rectangle);

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

    return 0;
}

float area_of_rect(Rectangle rect)
{
    float area = rect.length * rect.width;
    return area;
}

float peri_of_rect(Rectangle rect)
{
    float perimeter = 2 *(rect.length + rect.width);

    return perimeter;

}
```

Problem 5: 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:
    1. char name[50]: Name of the student
    2. int roll_no: Roll number
    3. float marks[5]: Marks in 5 subjects
    4. char grade: Grade assigned to the student
2. Write a function to:
    1. 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>




struct Student {

    int roll_no;

    char name[50];

    float marks[5];

    char grade;

};
```

```
void input_details(struct Student *students, int n);

void calculate_grades(struct Student *students, int n);

void display_details(struct Student *students, int n);


int main()

{

    int n;

    printf("Enter the number of students: ");

    scanf("%d", &n);


    struct Student students[n];




    input_details(students, n);

    calculate_grades(students, n);

    display_details(students, n);


    return 0;

}




void input_details(struct Student *students, int n)

{

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

    {
        printf("\nEnter details for student %d:\n", i + 1);
        printf("Roll number: ");
        scanf("%d", &students[i].roll_no);
        printf("Name: ");
        getchar(); // Consume the newline character
        scanf("%[^\n]", students[i].name);
        printf("Enter marks in 5 subjects:\n");
        for (int j = 0; j < 5; j++)
        {
            printf("Subject %d: ", j + 1);
            scanf("%f", &students[i].marks[j]);
        }
    }
}


void calculate_grades(struct Student *students, int n)
{
    for (int i = 0; i < n; i++)
    {
        float total = 0.0;
        for (int j = 0; j < 5; j++)
        {
            total += students[i].marks[j];
        }
        float average = total / 5.0;
        if (average >= 90)
        {
            students[i].grade = 'A';
        }
        else if (average >= 80)
        {
            students[i].grade = 'B';
        }
        else if (average >= 70)
        {
            students[i].grade = 'C';
        }
        else if (average >= 60)
        {
            students[i].grade = 'D';
        }
        else
        {
            students[i].grade = 'F';
        }
    }
}
```

```
void display_details(struct Student *students, int n)

{

    printf("\nStudent Details:\n");

    printf("Roll No\tName\t\t\tMarks (5 subjects)\t\tGrade\n");

    printf("----------------------------------------------------------------------\n");

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

    {

        printf("%d\t%-20s\t", students[i].roll_no, students[i].name);

        for (int j = 0; j < 5; j++)

        {

            printf("%.2f ", students[i].marks[j]);

        }

        printf("\t\t%c\n", students[i].grade);

    }

}
```

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:

1. Define a structure Point with fields:
    1. float x: X-coordinate of the point
    2. float y: Y-coordinate of the point
2. Write functions to:
    1. Calculate the distance between two points.
    2. 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>


typedef struct {

            float x;

            float y;

} Point;


float distance_bw_points(Point p1,Point p2);

int isInsideCircle(Point p, float radius);

int main()

{

            Point p1,p2;

            float radius;




            printf("Enter the coordinates of p1(x1,y1):");

            scanf("%f %f",&p1.x,&p1.y);


    printf("Enter the coordinates of p2(x2,y2):");

            scanf("%f %f",&p2.x,&p2.y);


            printf("Enter the radius of the circle (centered at origin): ");

    scanf("%f", &radius);


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



}



float distance_bw_points(Point p1,Point p2)

{

    return sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));

}


int isInsideCircle(Point p, float radius)

{

    float distanceFromOrigin = sqrt(pow(p.x, 2) + pow(p.y, 2));

    return distanceFromOrigin <= radius;

}
```

Problem 7: 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:
    1. char name[50]: Employee name
    2. int emp_id: Employee ID
    3. float salary: Employee salary
    4. float tax: Tax to be calculated (initialized to 0)
2. Write a function to:

1. Calculate tax based on salary slabs (e.g., 10% for salaries below $50,000, 20% otherwise).
2. Modify the tax field of the structure.

3. Pass the structure by reference to the function and display the updated tax in main.

```c
#include <stdio.h>


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


void calculateTax(struct Employee *e)
{

   if (e->salary < 50000)
   {
      e->tax = e->salary * 0.10;  // 10% tax for salary below 50,000
   } else {
      e->tax = e->salary * 0.20;  // 20% tax for salary above or equal to 50,000
   }
}

int main() {
   struct Employee e;


   printf("Enter employee name: ");
   scanf("%[^\n]", e.name);
   printf("Enter employee ID: ");
   scanf("%d", &e.emp_id);
   printf("Enter employee salary: ");
   scanf("%f", &e.salary);


   calculateTax(&e);


   printf("\nEmployee Details:\n");
   printf("Name: %s\n", e.name);
   printf("Employee ID: %d\n", e.emp_id);
   printf("Salary: %.2f\n", e.salary);
   printf("Calculated Tax: %.2f\n", e.tax);

   return 0;
}
```

Problem Statement 8: Vehicle Service Center Management

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

Description:

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

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

typedef struct servicetype {
   char service_type[30];
   float cost;
   char service_date[12];
} Servicetype;

typedef struct vehicle {
   char license_plate[15];
   char owner_name[50];
   char vehicle_type[20];
   Servicetype services[10];
   int service_count;
} Vehicle;

Vehicle service_records[100]; // max 100 vehicles
int vehicle_count = 0;
const int max_service = 10;  // max services per vehicle

void add_vehicle(Vehicle *);
void update_service(void);
void display_vehicle_details(void);
void generate_summary_report(void);

int main()
{
    int choice;

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

        switch (choice)
        {
            case 1:
                if (vehicle_count >= 100)
                {
                    printf("Service center is full. Cannot add more vehicles.\n");
                } else
                {
                    add_vehicle(&service_records[vehicle_count]);
                    printf("Vehicle added successfully!\n");
                    vehicle_count++;
                }
                break;
            case 2:
                update_service();
                break;
            case 3:
                display_vehicle_details();
                break;
            case 4:
                generate_summary_report();
                break;
            case 5:
                printf("Exiting system. Goodbye!\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 5);
    return 0;
}

void add_vehicle(Vehicle *v)
{
    printf("Enter license plate (up to 14 characters): ");
    scanf(" %[^\n]", v->license_plate);

    printf("Enter owner name (up to 49 characters): ");
    scanf(" %[^\n]", v->owner_name);

    printf("Enter vehicle type (e.g., car, bike): ");
    scanf(" %[^\n]", v->vehicle_type);
    v->service_count = 0;
}

void update_service(void)
{
    char license_plate[15];
    printf("Enter the license plate of the vehicle to update service history: ");
    scanf(" %[^\n]", license_plate);

    for (int i = 0; i < vehicle_count; i++)
    {
        if (strcmp(service_records[i].license_plate, license_plate) == 0)
        {
            Vehicle *v = &service_records[i];

            if (v->service_count >= max_service)
            {
                printf("Service history for this vehicle is full.\n");
                return;
            }

            Servicetype *service = &v->services[v->service_count];

            printf("Enter service type (e.g., Oil Change, Tire Replacement): ");
            scanf(" %[^\n]", service->service_type);

            do {
                printf("Enter cost of the service (positive number): ");
                scanf("%f", &service->cost);
            } while (service->cost < 0);

            printf("Enter service date (DD-MM-YYYY): ");
            scanf(" %[^\n]", service->service_date);

            v->service_count++;
            printf("Service updated successfully for vehicle with license plate %s.\n", v-
>license_plate);
            return;
        }
    }

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

void display_vehicle_details(void)
{
    char license_plate[15];
    printf("Enter the license plate of the vehicle to display details: ");
    scanf(" %[^\n]", license_plate);

    for (int i = 0; i < vehicle_count; i++)
    {
        if (strcmp(service_records[i].license_plate, license_plate) == 0)
        {
            Vehicle *v = &service_records[i];
            printf("\n=== Vehicle Details ===\n");
            printf("License Plate: %s\n", v->license_plate);
            printf("Owner Name: %s\n", v->owner_name);
            printf("Vehicle Type: %s\n", v->vehicle_type);

            if (v->service_count == 0)
            {
                printf("No services recorded for this vehicle.\n");
            } else
            {
                printf("\n=== Service History ===\n");
                for (int j = 0; j < v->service_count; j++)
                {
                    printf("Service %d:\n", j + 1);
                    printf("  Service Type: %s\n", v->services[j].service_type);
                    printf("  Cost: %.2f\n", v->services[j].cost);
                    printf("  Service Date: %s\n", v->services[j].service_date);
                }
            }
            return;
        }
    }

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

void generate_summary_report(void)
{
    float total_revenue = 0.0;

    if (vehicle_count == 0)
    {
        printf("No vehicles in the service center records.\n");
        return;
    }

    printf("\n=== Summary Report ===\n");
    printf("Total Vehicles Serviced: %d\n", vehicle_count);

    for (int i = 0; i < vehicle_count; i++)
    {
        Vehicle *v = &service_records[i];
        printf("\nVehicle %d:\n", i + 1);
        printf("  License Plate: %s\n", v->license_plate);
        printf("  Owner Name: %s\n", v->owner_name);
        printf("  Vehicle Type: %s\n", v->vehicle_type);

        float vehicle_total_cost = 0.0;
        for (int j = 0; j < v->service_count; j++)
        {
            vehicle_total_cost += v->services[j].cost;
        }

        printf("  Total Service Cost for Vehicle: %.2f\n", vehicle_total_cost);
        total_revenue += vehicle_total_cost;
    }

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