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>
typedef struct Student
{
int roll_no;
char *name;
float marks[5];
} Student;
void inputStudentDetails(Student *sptr, int n);
void displayStudentDetails(const Student *sptr, int n);
int main()
{
int n;
printf("Enter the number of students: ");
scanf("%d", &n);
Student students[n];
Student *sptr = students;
inputStudentDetails(sptr, n);
displayStudentDetails(sptr, n);
for (int i = 0; i < n; i++)
{
free(sptr[i].name);
}
return 0;
}
void inputStudentDetails(Student *sptr, int n)
{
for (int i = 0; i < n; i++)
{
printf("\nEnter details for Student %d:\n", i + 1);
printf("Roll Number: ");
scanf("%d", &(sptr[i].roll_no));
sptr[i].name = (char *)malloc(50 * sizeof(char));
if (sptr[i].name == NULL)
{
printf("Memory allocation failed for name.\n");
exit(1);
}
printf("Name: ");
scanf(" %[^\n]", sptr[i].name);
printf("Enter marks for 5 subjects: ");
for (int j = 0; j < 5; j++)
{
scanf("%f", &(sptr[i].marks[j]));
}
}
}
void displayStudentDetails(const Student *sptr, int n)
{
printf("\nStudent Details:\n");
printf("Roll No\t\tName\t\tAverage\n");
for (int i = 0; i < n; i++) {
float total = 0.0;
for (int j = 0; j < 5; j++) {
total += sptr[i].marks[j];
}
float average = total / 5;
printf("%d\t\t%s\t\t%.2f\n", sptr[i].roll_no, sptr[i].name, average);
}
}
/*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>
typedef struct Book {
char *title;
char *author;
int *copies;
} Book;
void inputBookDetails(Book *books, int count);
void displayBookDetails(const Book *books, int count);
void updateBookCopies(Book *books, int count);
void issueBook(Book *books, int count);
void freeBookMemory(Book *books, int count);
int main() {
int n = 0;
Book *books = (Book *)malloc(100 * sizeof(Book));
if (books == NULL) {
printf("Memory allocation failed.\n");
return 1;
}
int choice;
do {
printf("\nLibrary System Menu:\n");
printf("1. Add Book Details\n");
printf("2. Update Book Copies\n");
printf("3. Display All Books\n");
printf("4. Issue Book\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice)
{
case 1:
inputBookDetails(books, n);
n++;
printf("The book details are added successfully!!\n");
break;
case 2:
updateBookCopies(books, n);
break;
case 3:
displayBookDetails(books, n);
break;
case 4:
issueBook(books, n);
break;
case 5:
printf("Exiting the program. Freeing allocated memory...\n");
break;
default:
printf("Invalid choice. Please try again.\n");
}
} while (choice != 5);
freeBookMemory(books, n);
free(books);
return 0;
}
void inputBookDetails(Book *books, int count)
{
printf("\nEnter details for Book %d:\n", count + 1);
books[count].title = (char *)malloc(100 * sizeof(char));
if (books[count].title == NULL)
{
printf("Memory allocation failed for title.\n");
exit(1);
}
printf("Enter book title: ");
scanf(" %[^\n]", books[count].title);
books[count].author = (char *)malloc(100 * sizeof(char));
if (books[count].author == NULL)
{
printf("Memory allocation failed for author.\n");
exit(1);
}
printf("Enter author name: ");
scanf(" %[^\n]", books[count].author);
books[count].copies = (int *)malloc(sizeof(int));
if (books[count].copies == NULL)
{
printf("Memory allocation failed for copies.\n");
exit(1);
}
printf("Enter number of copies: ");
scanf("%d", books[count].copies);
}
void displayBookDetails(const Book *books, int count)
{
```

```
if (count == 0)
{
printf("\nNo books in the system.\n");
return;
}
printf("\nBook Details:\n");
for (int i = 0; i < count; i++) {
printf("Book %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 updateBookCopies(Book *books, int count)
{
if (count == 0)
{
printf("\nNo books in the system to update.\n");
return;
}
char title[100];
printf("\nEnter the title of the book to update copies: ");
scanf(" %[^\n]", title);
for (int i = 0; i < count; i++)
{
if (strcmp(books[i].title, title) == 0) {
printf("Current number of copies: %d\n", *(books[i].copies));
printf("Enter new number of copies: ");
scanf("%d", books[i].copies);
printf("Copies updated successfully.\n");
return;
}
}
printf("Book with the title '%s' not found.\n", title);
}
void issueBook(Book *books, int count)
{
if (count == 0)
{
printf("\nNo books in the system to issue.\n");
return;
}
char title[100];
printf("\nEnter the title of the book to issue: ");
```

```
scanf(" %[^\n]", title);
for (int i = 0; i < count; i++)
{
if (strcmp(books[i].title, title) == 0)
{
if (*(books[i].copies) > 0)
{
(*(books[i].copies))--;
printf("Book '%s' issued successfully. Remaining copies: %d\n", books[i].title,
*(books[i].copies));
}
else
{
printf("Book '%s' is out of stock.\n", books[i].title);
}
return;
}
}
printf("Book with the title '%s' not found.\n", title);
}
void freeBookMemory(Book *books, int count)
{
for (int i = 0; i < count; i++)
{
free(books[i].title);
free(books[i].author);
free(books[i].copies);
}
}
```

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:
○ float real: Real part of the complex number
○ float imag: Imaginary part of the complex number
2. Write functions to:
○ Add two complex numbers and return the result.
○ 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 num1, struct Complex num2) {
struct Complex result;
result.real = num1.real + num2.real;
result.imag = num1.imag + num2.imag;
return result;
}
struct Complex multiplyComplex(struct Complex num1, struct Complex num2) {
struct Complex result;
result.real = (num1.real * num2.real) - (num1.imag * num2.imag);
result.imag = (num1.real * num2.imag) + (num1.imag * num2.real);
return result;
}
void displayComplex(struct Complex num) {
if(num.imag < 0)
printf("%.2f - %.2fi\n", num.real, -num.imag);
else
printf("%.2f + %.2fi\n", num.real, num.imag);
}
int main() {
struct Complex num1, num2, sum, product;
printf("Enter the real and imaginary parts of the first complex number: ");
scanf("%f %f", &num1.real, &num1.imag);
printf("Enter the real and imaginary parts of the second complex number: ");
scanf("%f %f", &num2.real, &num2.imag);
sum = addComplex(num1, num2);
product = multiplyComplex(num1, num2);
printf("\nSum: ");
displayComplex(sum);
printf("Product: ");
displayComplex(product);
return 0;
}
```

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:
   - ○ float length: Length of the rectangle
   - ○ float width: Width of the rectangle
2. Write functions to:
   - ○ Calculate and return the area of the rectangle.
   - ○ 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 len;
float width;
};
float calculateArea(struct Rectangle rect) {
return rect.len * rect.width;
}
float calculatePerimeter(struct Rectangle rect) {
return 2 * (rect.len + rect.width);
}
int main() {
struct Rectangle rect;
printf("Enter the length and width of the rectangle: ");
scanf("%f %f", &rect.len, &rect.width);
float area = calculateArea(rect);
float perimeter = calculatePerimeter(rect);
printf("Area of the rectangle: %.2f\n", area);
printf("Perimeter of the rectangle: %.2f\n", perimeter);
return 0;
}
```

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:
   - ○ 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
2. Write a function to:
   - ○ 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 {
char name[50];
int roll_no;
float marks[5];
char grade;
};
void calculateGrade(struct Student *student) {
float total = 0.0;
for (int i = 0; i < 5; i++) {
total += student->marks[i];
}
float average = total / 5;
if (average >= 90) {
student->grade = 'A';
} else if (average >= 75) {
student->grade = 'B';
} else if (average >= 50) {
student->grade = 'C';
} else {
student->grade = 'F';
}
printf("Average Marks: %.2f\n", average);
}
int main() {
struct Student student;
printf("Enter student name: ");
fgets(student.name, sizeof(student.name), stdin);
printf("Enter roll number: ");
scanf("%d", &student.roll_no);
printf("Enter marks for 5 subjects: ");
for (int i = 0; i < 5; i++) {
scanf("%f", &student.marks[i]);
}
calculateGrade(&student);
printf("\nStudent Name: %s", student.name);
printf("Roll Number: %d\n", student.roll_no);
printf("Marks: ");
for (int i = 0; i < 5; i++) {
printf("%.2f ", student.marks[i]);
}
printf("\nGrade: %c\n", student.grade);
return 0;
}
```

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:
   - ○ float x: X-coordinate of the point
   - ○ float y: Y-coordinate of the point
2. 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).
3. Pass the Point structure to these functions and display the results.

```
#include <stdio.h>
#include <math.h>
typedef struct Point
{
float x;
float y;
} Point;
float calculateDistance(Point *p1, Point *p2);
int isPointInsideCircle(Point *p, float radius);
int main()
{
Point point1, point2;
float radius;
int choice;
do
{
printf("\nMenu:\n");
printf("1. Calculate distance between two points\n");
printf("2. Check if a point lies inside a circle\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice)
{
case 1:
printf("Enter coordinates of the first point (x, y): ");
scanf("%f %f", &point1.x, &point1.y);
printf("Enter coordinates of the second point (x, y): ");
scanf("%f %f", &point2.x, &point2.y);
float distance = calculateDistance(&point1, &point2);
printf("The distance between the two points is: %.2f\n", distance);
break;
case 2:
printf("Enter the coordinates of the point (x, y): ");
scanf("%f %f", &point1.x, &point1.y);
printf("Enter the radius of the circle: ");
scanf("%f", &radius);
if (isPointInsideCircle(&point1, radius))
{
printf("The point lies inside the circle.\n");
}
else
{
printf("The point lies outside the circle.\n");
}
break;
case 3:
printf("Exiting...\n");
break;
default:
printf("Invalid option! Please try again.\n");
}
} while (choice != 3);
return 0;
}
float calculateDistance(Point *p1, Point *p2)
{
return sqrt(pow(p2->x - p1->x, 2) + pow(p2->y - p1->y, 2));
}
int isPointInsideCircle(Point *p, float radius)
{
float distanceFromOrigin = sqrt(pow(p->x, 2) + pow(p->y, 2));
return distanceFromOrigin <= radius;
}
```

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:
   - char name[50]: Employee name
   - int emp_id: Employee ID
   - float salary: Employee salary
   - float tax: Tax to be calculated (initialized to 0)
2. 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.
3. Pass the structure by reference to the function and display the updated tax in main.

```
#include <stdio.h>
typedef struct Employee
{
char name[50];
int emp_id;
float salary;
float tax;
} Employee;
void calculateTax(Employee *);
int main()
{
Employee emp;
Employee *ptr = &emp;
printf("Enter employee name: ");
scanf("%[^\n]", emp.name);
printf("Enter employee ID: ");
scanf("%d", &emp.emp_id);
printf("Enter employee salary: ");
scanf("%f", &emp.salary);
calculateTax(ptr);
printf("\nEmployee Tax Details:\n");
printf("Name: %s\n", emp.name);
printf("Employee ID: %d\n", emp.emp_id);
printf("Salary: %.2f\n", emp.salary);
printf("Calculated Tax: %.2f\n", emp.tax);
return 0;
}
void calculateTax(Employee *ptr)
{
if (ptr->salary < 50000)
{
ptr->tax = ptr->salary * 0.10;
}
else
{
ptr->tax = ptr->salary * 0.20;
}
}
```
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:
○ 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)
2. 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
3. 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>
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;
void add_vehicle(Vehicle *);
void update_service(void);
void display_vehicle_details(void);
void generate_summary_report(void);
Vehicle service_records[100]; //max 100 vehicles
int vehicle_count = 0;
int max_service = 10; //max service per vehicle
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: ");
scanf(" %[^\n]", v->license_plate);
printf("Enter owner name: ");
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);
printf("Enter cost of the service: ");
scanf("%f", &service->cost);
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[50];
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()
{
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);
}
```