Problem Statement: Employee Records Management

Write a C program to manage a list of employees using **dynamic memory allocation**. The program should:

1. Define a structure named Employee with the following fields:
    - id (integer): A unique identifier for the employee.
    - name (character array of size 50): The employee's name.
    - salary (float): The employee's salary.
2. Dynamically allocate memory for storing information about n employees (where n is input by the user).
3. Implement the following features:
    - **Input Details**: Allow the user to input the details of each employee (ID, name, and salary).
    - **Display Details**: Display the details of all employees.
    - **Search by ID**: Allow the user to search for an employee by their ID and display their details.
    - **Free Memory**: Ensure that all dynamically allocated memory is freed at the end of the program.

Constraints

- n (number of employees) must be a positive integer.
- Employee IDs are unique.

Sample Input/Output

Input:

Enter the number of employees: 3

Enter details of employee 1:

ID: 101

Name: Alice

Salary: 50000

Enter details of employee 2:

ID: 102

Name: Bob

Salary: 60000

Enter details of employee 3:

ID: 103

Name: Charlie

Salary: 55000

Enter ID to search for: 102

Output:

Employee Details:

ID: 101, Name: Alice, Salary: 50000.00

ID: 102, Name: Bob, Salary: 60000.00

ID: 103, Name: Charlie, Salary: 55000.00

Search Result:

ID: 102, Name: Bob, Salary: 60000.00

```c
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

struct employee{

    int id;

    char name[50];

    float salary;

};

int main(){

    printf("\nEmployee Record");

    int op,count=0;

    struct employee *ptr=NULL;

    int n;

    int id;
```

```
    while(1){

        printf("\n");

        printf("\n1.Add Employee");

        printf("\n2.Display all data");

        printf("\n3.Search by ID");

        printf("\n4.Exit by freeing memory");

        printf("\nEnter the option::");

        scanf("%d",&op);

        switch(op){

            case 1:



                printf("\nEnter no:of Employees to add::");

                scanf("%d",&n);

                ptr=(struct employee *)malloc(n*sizeof(struct
employee));



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

                    int unique = 0;

                    while (!unique) {

                        unique = 1;

                        printf("\nEnter ID:: ");

                        scanf("%d", &(ptr + count + i)->id);



                        for (int j = 0; j < count + i; j++) {

                            if ((ptr + count + i)->id == (ptr + j)->id)
{
```

```
                        printf("\nEnter a unique ID");
                        unique = 0;
                        break;
                    }
                }
            }

            printf("\nEnter name:: ");
            scanf("%s", (ptr + count + i)->name);
            printf("\nEnter salary:: ");
            scanf("%f", &(ptr + count + i)->salary);
        }

        count += n;
        break;
    case 2:
        printf("\nDisplaying all Employee::::");
        for(int i=0;i<count;i++){
            printf("\nEmployee ID::%d",(ptr+i)->id);
            printf("\nEmployee name::%d",(ptr+i)->name);
            printf("\nEmployee ID::%d",(ptr+i)->salary);

        }
        break;
    case 3:
```

```
            printf("\nEnter ID of employee to search::");
            scanf("%d",&id);
            for(int i=0;i<n;i++){
                if(id==(ptr+i)->id){
                    printf("\nFound!!");
                    printf("\nEmployee ID::%d",(ptr+i)->id);
                    printf("\nEmployee name::%s",(ptr+i)->name);
                    printf("\nEmployee ID::%f",(ptr+i)->salary);
                    break;
                }
                else{
                    printf("\nNot Found!!");
                    break;
                }
            }
            break;
        case 4:
            printf("\nExiting...");
            free(ptr);
            exit(0);
    }


}
```

```
}
```

Problem 1: Book Inventory System

Problem Statement:

Write a C program to manage a book inventory system using dynamic memory allocation. The program should:

1. Define a structure named Book with the following fields:
    - id (integer): The book's unique identifier.
    - title (character array of size 100): The book's title.
    - price (float): The price of the book.
2. Dynamically allocate memory for n books (where n is input by the user).
3. Implement the following features:
    - **Input Details**: Input details for each book (ID, title, and price).
    - **Display Details**: Display the details of all books.
    - **Find Cheapest Book**: Identify and display the details of the cheapest book.
    - **Update Price**: Allow the user to update the price of a specific book by entering its ID.

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

```
#include <stdlib.h>

#include <string.h>



struct Book {

    int id;

    char title[100];

    float price;

};



int main() {

    struct Book *books = NULL;
```

```c
    int n = 0, count = 0, op, id;
    float new_price;


    while (1) {
        printf("\n\nBook Inventory System");
        printf("\n1. Add Books");
        printf("\n2. Display All Books");
        printf("\n3. Find Cheapest Book");
        printf("\n4. Update Price");
        printf("\n5. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &op);


        switch (op) {
            case 1:
                printf("\nEnter number of books to add: ");
                scanf("%d", &n);
                books =(struct Book *)malloc(n*sizeof(struct Book));
                for (int i = 0; i < n; i++) {
                    int unique = 0;
                    while (!unique) {
                        unique = 1;
                        printf("\nEnter Book ID: ");
                        scanf("%d", &(books + count + i)->id);
```

```
                    for (int j = 0; j < count + i; j++) {
                        if ((books + count + i)->id == (books +
j)->id) {
                            printf("\nEnter a unique ID");
                            unique = 0;
                            break;
                        }
                    }
                }

                printf("Enter Book Title: ");
                scanf(" %s", (books + count + i)->title);
                printf("Enter Book Price: ");
                scanf("%f", &(books + count + i)->price);
            }

            count += n;
            break;

        case 2:
            if (count == 0) {
                printf("\nNo books in the inventory.");
            } else {
                printf("\nDisplaying all books:");
```

```
                    for (int i = 0; i < count; i++) {
                        printf("\nBook ID: %d", (books + i)->id);
                        printf("\nBook Title: %s", (books + i)->title);
                        printf("\nBook Price: %.2f", (books +
i)->price);

                    }
                }
                break;

            case 3:
                if (count == 0) {
                    printf("\nNo books in the inventory.");
                } else {
                    float min_price = (books + 0)->price;
                    int min_index = 0;

                    for (int i = 1; i < count; i++) {
                        if ((books + i)->price < min_price) {
                            min_price = (books + i)->price;
                            min_index = i;
                        }
                    }

                    printf("\nThe cheapest book is:");
```

```
                    printf("\nBook ID: %d", (books + min_index)->id);
                    printf("\nBook Title: %s", (books +
min_index)->title);
                    printf("\nBook Price: %.2f", (books +
min_index)->price);
                }
                break;

            case 4:
                if (count == 0) {
                    printf("\nNo books");
                } else {
                    printf("\nEnter the ID of the book to update its
price: ");
                    scanf("%d", &id);

                    int found = 0;
                    for (int i = 0; i < count; i++) {
                        if ((books + i)->id == id) {
                            printf("Enter new price: ");
                            scanf("%f", &new_price);
                            (books + i)->price = new_price;
                            printf("\nPrice updated");
                            found = 1;
                            break;
                        }
```

```
                    }
                    if (!found) {
                        printf("\nNo Book Found!");
                    }
                }
                break;

            case 5:
                printf("\nExiting...");
                free(books);
                exit(0);

            default:
                printf("\nWrong Choice");
        }
    }

    return 0;
}
```

Problem 2: Dynamic Point Array

Problem Statement:

Write a C program to handle a dynamic array of points in a 2D space using dynamic memory allocation. The program should:

1. Define a structure named Point with the following fields:
    - x (float): The x-coordinate of the point.
    - y (float): The y-coordinate of the point.
2. Dynamically allocate memory for n points (where n is input by the user).
3. Implement the following features:
    - **Input Details**: Input the coordinates of each point.
    - **Display Points**: Display the coordinates of all points.
    - **Find Distance**: Calculate the Euclidean distance between two points chosen by the user (by their indices in the array).
    - **Find Closest Pair**: Identify and display the pair of points that are closest to each other.

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

struct Point {
    float x;
    float y;
};

int main() {
    struct Point *points = NULL;
    int n = 0, count = 0, option;
    int index1, index2;

    while (1) {
        printf("\n\nDynamic Point Array");
        printf("\n1. Add Points");
        printf("\n2. Display Points");
        printf("\n3. Find Distance Between Two Points");
        printf("\n4. Find Closest Pair");
        printf("\n5. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &option);

        switch (option) {
            case 1:
                printf("\nEnter number of points to add: ");
                scanf("%d", &n);
                points = realloc(points, (count + n) * sizeof(struct
Point));
                if (points == NULL) {
                    printf("\nMemory allocation failed!");
```

```
                    exit(1);
                }

                for (int i = count; i < count + n; i++) {
                    printf("\nEnter coordinates for Point %d:", i + 1);
                    printf("\nEnter x: ");
                    scanf("%f", &(points + i)->x);
                    printf("Enter y: ");
                    scanf("%f", &(points + i)->y);
                }

                count += n;
                break;

            case 2:
                if (count == 0) {
                    printf("\nNo points available.");
                } else {
                    printf("\nDisplaying all points:");
                    for (int i = 0; i < count; i++) {
                        printf("\nPoint %d: (%.2f, %.2f)", i + 1,
(points + i)->x, (points + i)->y);
                    }
                }
                break;

            case 3:
                if (count < 2) {
                    printf("\nAt least two points are required to
calculate the distance.");
                } else {
                    printf("\nEnter the indices of the two points (1 to
%d): ", count);
                    scanf("%d %d", &index1, &index2);

                    if (index1 < 1 || index2 < 1 || index1 > count ||
index2 > count) {
                        printf("\nInvalid indices! Please try again.");
                    } else {
                        float dx = (points + index1 - 1)->x - (points +
index2 - 1)->x;
                        float dy = (points + index1 - 1)->y - (points +
index2 - 1)->y;
```

```
                        float distance = sqrt(dx * dx + dy * dy);
                        printf("\nDistance between Point %d and Point
%d: %.2f", index1, index2, distance);
                    }
                }
                break;

            case 4:
                if (count < 2) {
                    printf("\nAt least two points are required to find
the closest pair.");
                } else {
                    float min_distance = -1;
                    int p1 = 0, p2 = 0;

                    for (int i = 0; i < count; i++) {
                        for (int j = i + 1; j < count; j++) {
                            float dx = (points + i)->x - (points +
j)->x;
                            float dy = (points + i)->y - (points +
j)->y;
                            float distance = sqrt(dx * dx + dy * dy);

                            if (min_distance == -1 || distance <
min_distance) {
                                min_distance = distance;
                                p1 = i;
                                p2 = j;
                            }
                        }
                    }

                    printf("\nThe closest pair of points is:");
                    printf("\nPoint %d: (%.2f, %.2f)", p1 + 1, (points
+ p1)->x, (points + p1)->y);
                    printf("\nPoint %d: (%.2f, %.2f)", p2 + 1, (points
+ p2)->x, (points + p2)->y);
                    printf("\nDistance: %.2f", min_distance);
                }
                break;

            case 5:
                printf("\nExiting...");
```

```
                free(points);
                exit(0);

            default:
                printf("\nInvalid choice! Please try again.");
        }
    }

    return 0;
}
```

Problem Statement: Vehicle Registration System

Write a C program to simulate a vehicle registration system using **unions** to handle different types of vehicles. The program should:

1. Define a union named Vehicle with the following members:
    - car_model (character array of size 50): To store the model name of a car.
    - bike_cc (integer): To store the engine capacity (in CC) of a bike.
    - bus_seats (integer): To store the number of seats in a bus.
2. Create a structure VehicleInfo that contains:
    - type (character): To indicate the type of vehicle (C for car, B for bike, S for bus).
    - Vehicle (the union defined above): To store the specific details of the vehicle based on its type.
3. Implement the following features:
    - **Input Details**: Prompt the user to input the type of vehicle and its corresponding details:
        - For a car: Input the model name.
        - For a bike: Input the engine capacity.
        - For a bus: Input the number of seats.
    - **Display Details**: Display the details of the vehicle based on its type.
4. Use the union effectively to save memory and ensure only relevant information is stored.

Constraints

- The type of vehicle should be one of C, B, or S.
- For invalid input, prompt the user again.

Sample Input/Output

Input:

Enter vehicle type (C for Car, B for Bike, S for Bus): C

Enter car model: Toyota Corolla

Output:

Vehicle Type: Car

Car Model: Toyota Corolla

Input:

Enter vehicle type (C for Car, B for Bike, S for Bus): B

Enter bike engine capacity (CC): 150

Output:

Vehicle Type: Bike

Engine Capacity: 150 CC

Input:

Enter vehicle type (C for Car, B for Bike, S for Bus): S

Enter number of seats in the bus: 50

Output:

Vehicle Type: Bus

Number of Seats: 50

```c
#include<stdio.h>
#include<string.h>
union Vehicle{
    char car_model[50];
    int bike_cc;
    int bus_seats;
};
struct VehicleInfo{
    char type;
    union Vehicle details;
}vehicle;
void display(struct VehicleInfo vehicle);
int main(){

    printf("\nEnter type of vehicle::");
```

```
    scanf("%c",&vehicle.type);
    if(vehicle.type == 'C'){
        printf("\nEnter car model::");
        scanf("%s",&vehicle.details.car_model);
    }
    else if(vehicle.type == 'B'){
        printf("\nEnter Bike CC::");
        scanf("%d",&vehicle.details.bike_cc);
    }
    else if(vehicle.type == 'S'){
        printf("\nEnter Bus seats::");
        scanf("%d",&vehicle.details.bus_seats);
    }
    else{
        printf("\nInvalid Input");
    }
    display(vehicle);
    return 0;

}
void display(struct VehicleInfo vehicle){
    printf("\nVehicle ::");
    if(vehicle.type == 'C'){
        printf("Car\n");
        printf("Car Model: %s\n", vehicle.details.car_model);
    } else if (vehicle.type == 'B') {
        printf("Bike\n");
        printf("Engine Capacity: %d CC\n", vehicle.details.bike_cc);
    } else if (vehicle.type == 'S') {
        printf("Bus\n");
        printf("Number of Seats: %d\n", vehicle.details.bus_seats);
    } else {
        printf("\nError\n");
    }
}
```

Problem 1: Traffic Light System

Problem Statement:

Write a C program to simulate a traffic light system using enum. The program should:

1. Define an enum named TrafficLight with the values RED, YELLOW, and GREEN.
2. Accept the current light color as input from the user (as an integer: 0 for RED, 1 for YELLOW, 2 for GREEN).
3. Display an appropriate message based on the current light:
    - RED: "Stop"
    - YELLOW: "Ready to move"
    - GREEN: "Go"

```c
#include <stdio.h>



enum Trafficlight{

    RED,

    YELLOW,

    GREEN



};




int main(){


  enum Trafficlight light = RED;

  switch(light){


      case 0:


        printf("STOP\n");


        break;
```

```
    case 1:

      printf("READY TO MOVE\n");

      break;

    case 2:

      printf("GO\n");

      break;

    default:

      printf("Wrong Option\n");

      break;

 }

   return 0;

}
```

Problem 2: Days of the Week

Problem Statement:

Write a C program that uses an enum to represent the days of the week. The program should:

1. Define an enum named Weekday with values MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, and SUNDAY.
2. Accept a number (1 to 7) from the user representing the day of the week.
3. Print the name of the day and whether it is a weekday or a weekend.
    - Weekends: SATURDAY and SUNDAY
    - Weekdays: The rest

```c
#include <stdio.h>



enum Weekday {

    MONDAY = 1, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY

};



int main() {

    int day;

    printf("Enter a number (1-7):: ");

    scanf("%d", &day);

    switch (day) {

        case MONDAY:

            printf("MONDAY - Weekday\n");

            break;

        case TUESDAY:

            printf("TUESDAY - Weekday\n");

            break;
```

```
        case WEDNESDAY:
            printf("WEDNESDAY - Weekday\n");
            break;
        case THURSDAY:
            printf("THURSDAY - Weekday\n");
            break;
        case FRIDAY:
            printf("FRIDAY - Weekday\n");
            break;
        case SATURDAY:
            printf("SATURDAY - Weekend\n");
            break;
        case SUNDAY:
            printf("SUNDAY - Weekend\n");
            break;
        default:
            printf("Invalid\n");
    }

    return 0;
}
```

Problem 3: Shapes and Their Areas

Problem Statement:

Write a C program to calculate the area of a shape based on user input using enum. The program should:

1. Define an enum named Shape with values CIRCLE, RECTANGLE, and TRIANGLE.
2. Prompt the user to select a shape (0 for CIRCLE, 1 for RECTANGLE, 2 for TRIANGLE).
3. Based on the selection, input the required dimensions:
    - For CIRCLE: Radius
    - For RECTANGLE: Length and breadth
    - For TRIANGLE: Base and height
4. Calculate and display the area of the selected shape.

```c
#include <stdio.h>
#define pi 3.14159

enum Shape {

    CIRCLE = 0,

    RECTANGLE,

    TRIANGLE

};



int main() {

    enum Shape op;

    printf("Enter 0 for circle,1 for rectangle,2 for triangle:");

    scanf("%d",&op);



    int r, l, b, base, h, a;



    switch (op) {

        case 0:

            printf("You selected CIRCLE. Enter the radius: ");
```

```
            scanf("%d", &r);
            a= pi * r * r;
            printf("The area of the circle is: %d\n", a);
            break;

        case 2:
            printf("You selected RECTANGLE. Enter the length and
breadth: ");
            scanf("%d %d", &l, &b);
            a= l * b;
            printf("The area of the rectangle is: %d\n", a);
            break;

        case 3:
            printf("You selected TRIANGLE. Enter the base and height:
");
            scanf("%d %d", &base, &h);
            a = 0.5 * base * h;
            printf("The area of the triangle is: %d\n", a);
            break;

        default:
            printf("Invalid choice.\n");
            return 1;
    }
```

```
    return 0;

}
```

Problem 4: Error Codes in a Program

Problem Statement:

Write a C program to simulate error handling using enum. The program should:

1. Define an enum named ErrorCode with values:
    - SUCCESS (0)
    - FILE_NOT_FOUND (1)
    - ACCESS_DENIED (2)
    - OUT_OF_MEMORY (3)
    - UNKNOWN_ERROR (4)
2. Simulate a function that returns an error code based on a scenario.
3. Based on the returned error code, print an appropriate message to the user.

```
#include <stdio.h>

#define pi 3.14159

enum error {

    SUCCESS = 0,

    FILE_NOT_FOUND,

    ACCESS_DENIED,

    OUT_OF_MEMORY,

    UNKNOWN_ERROR

};



int main() {

    enum error op;

    printf("Enter error code (0-4)");
```

```c
    scanf("%d",&op);
    switch (op) {
        case SUCCESS:
            printf("SUCCESS");
            break;

        case FILE_NOT_FOUND:
            printf("FILE_NOT_FOUND");
            break;

        case ACCESS_DENIED:
            printf("ACCESS_DENIED");
            break;

        case OUT_OF_MEMORY:
            printf("OUT_OF_MEMORY");
            break;

         case UNKNOWN_ERROR:
            printf("UNKNOWN_ERROR");
            break;

        default:
            printf("Invalid choice.\n");
            return 1;
```

```
    }



    return 0;

}
```

Problem 5: User Roles in a System

Problem Statement:

Write a C program to define user roles in a system using enum. The program should:

1. Define an enum named UserRole with values ADMIN, EDITOR, VIEWER, and GUEST.
2. Accept the user role as input (0 for ADMIN, 1 for EDITOR, etc.).
3. Display the permissions associated with each role:
    - ADMIN: "Full access to the system."
    - EDITOR: "Can edit content but not manage users."
    - VIEWER: "Can view content only."
    - GUEST: "Limited access, view public content only.

```
#include <stdio.h>
#define pi 3.14159
enum access {
    ADMIN = 0,
    EDITOR,
    VIEWER,
    GUEST
};

int main() {
    enum access op;
    printf("Enter 0 for admin,1 for editor,2 for viewer,3 for
guest::");
    scanf("%d",&op);
    switch (op) {
        case ADMIN:
            printf("Full access to the system");
            break;
```

```
        case EDITOR:
            printf("Can edit content but not manage users");
            break;

        case VIEWER:
            printf("Can view content only");
            break;

        case GUEST:
            printf("Limited access, view public content only");
            break;

        default:
            printf("Invalid choice.\n");
            return 1;
    }

    return 0;
}
```

Problem 1: Compact Date Storage
Problem Statement:
Write a C program to store and display dates using bit-fields. The program should:
1. Define a structure named Date with bit-fields:
o day (5 bits): Stores the day of the month (1-31).
o month (4 bits): Stores the month (1-12).
o year (12 bits): Stores the year (e.g., 2024).
2. Create an array of dates to store 5 different dates.
3. Allow the user to input 5 dates in the format DD MM YYYY and store them in the array.
4. Display the stored dates in the format DD-MM-YYYY.

```
#include <stdio.h>
struct Date {
    unsigned int day : 5;
    unsigned int month : 4;
    unsigned int year : 12;
};

int main() {
```

```
    struct Date dates[5];
    int i;
    printf("Enter 5 dates in the format DD MM YYYY:\n");
    for (i = 0; i < 5; i++) {
        int day, month, year;
        printf("Date %d: ", i + 1);
        scanf("%d %d %d", &day, &month, &year);
        (dates + i)->day = day;
        (dates + i)->month = month;
        (dates + i)->year = year;

    }


    printf("\nStored Dates:\n");
    for (i = 0; i < 5; i++) {
        printf("%02d-%02d-%04d\n", (dates + i)->day, (dates +
i)->month, (dates + i)->year);
    }

    return 0;
}
```

Problem 2: Status Flags for a Device
Problem Statement:
Write a C program to manage the status of a device using bit-fields. The program should:
1. Define a structure named DeviceStatus with the following bit-fields:
o power (1 bit): 1 if the device is ON, 0 if OFF.
o connection (1 bit): 1 if the device is connected, 0 if disconnected.
o error (1 bit): 1 if there's an error, 0 otherwise.
2. Simulate the device status by updating the bit-fields based on user input:
o Allow the user to set or reset each status.
3. Display the current status of the device in a readable format (e.g., Power: ON, Connection: DISCONNECTED, Error: NO).

```
#include <stdio.h>

struct DeviceStatus {
    unsigned int power : 1;
    unsigned int connection : 1;
    unsigned int error : 1;
```

```
};
void display(struct DeviceStatus device);
int main() {
    struct DeviceStatus device = {0, 0, 0};
    int op, value;

    printf("Device Status Management System\n");

    do {
        printf("\nMenu:\n");
        printf("1. Set Power Status (ON/OFF)\n");
        printf("2. Set Connection Status (CONNECTED/DISCONNECTED)\n");
        printf("3. Set Error Status (YES/NO)\n");
        printf("4. Display Status\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &op);

        switch (op) {
            case 1:
                printf("Enter Power Status (1 for ON, 0 for OFF): ");
                scanf("%d", &value);
                device.power = value;
                break;

            case 2:
                printf("Enter Connection Status (1 for CONNECTED, 0 for
DISCONNECTED): ");
                scanf("%d", &value);
                device.connection = value;
                break;

            case 3:
                printf("Enter Error Status (1 for YES, 0 for NO): ");
                scanf("%d", &value);
                device.error = value;
                break;

            case 4:
                display(device);
                break;

            case 5:
```

```
                printf("Exiting the program.\n");
                break;

            default:
                printf("Invalid\n");
        }
    } while (op != 5);

    return 0;
}
void displayStatus(struct DeviceStatus device) {
    printf("Device Status:\n");
    printf("Power: %s\n", device.power ? "ON" : "OFF");
    printf("Connection: %s\n", device.connection ? "CONNECTED" :
"DISCONNECTED");
    printf("Error: %s\n", device.error ? "YES" : "NO");
}
```

Problem 3: Storage Permissions
Problem Statement:
Write a C program to represent file permissions using bit-fields. The program should:
1. Define a structure named FilePermissions with the following bit-fields:
o read (1 bit): Permission to read the file.
o write (1 bit): Permission to write to the file.
o execute (1 bit): Permission to execute the file.
2. Simulate managing file permissions:
o Allow the user to set or clear each permission for a file.
o Display the current permissions in the format R:1 W:0 X:1 (1 for permission granted, 0 for denied).

```
#include <stdio.h>
struct FilePermissions {
    unsigned int read : 1;
    unsigned int write : 1;
    unsigned int execute : 1;
};
void display(struct FilePermissions file);
int main() {
    struct FilePermissions file = {0, 0, 0};
    int op, value;
```

```
    printf("File Permissions Management System\n");

    do {
        printf("\nMenu:\n");
        printf("1. Set Read Permission (1: Grant, 0: Deny)\n");
        printf("2. Set Write Permission (1: Grant, 0: Deny)\n");
        printf("3. Set Execute Permission (1: Grant, 0: Deny)\n");
        printf("4. Display Permissions\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &op);

        switch (op) {
            case 1:
                printf("Enter Read Permission (1: Grant, 0: Deny): ");
                scanf("%d", &value);
                file.read = value;
                break;

            case 2:
                printf("Enter Write Permission (1: Grant, 0: Deny): ");
                scanf("%d", &value);
                file.write = value;
                break;

            case 3:
                printf("Enter Execute Permission (1: Grant, 0: Deny):
");
                scanf("%d", &value);
                file.execute = value;
                break;

            case 4:
                display(file);
                break;

            case 5:
                printf("Exiting the program.\n");
                break;

            default:
                printf("Invalid\n");
        }
```

```
    } while (op != 5);

    return 0;
}
void displayPermissions(struct FilePermissions file) {
    printf("Current Permissions: R:%d W:%d X:%d\n",
           file.read, file.write, file.execute);
}
```

Problem 4: Network Packet Header
Problem Statement:
Write a C program to represent a network packet header using bit-fields. The program should:

1. Define a structure named PacketHeader with the following bit-fields:
   - version (4 bits): Protocol version (0-15).
   - IHL (4 bits): Internet Header Length (0-15).
   - type_of_service (8 bits): Type of service.
   - total_length (16 bits): Total packet length.
2. Allow the user to input values for each field and store them in the structure.
3. Display the packet header details in a structured format.

```
#include <stdio.h>
struct PacketHeader {
    unsigned int version : 4;
    unsigned int IHL : 4;
    unsigned int type_of_service : 8;
    unsigned int total_length : 16;
};
void display(struct PacketHeader packet);
int main() {
    struct PacketHeader packet;
    int value;

    printf("Network Packet Header Representation\n");
    printf("Enter Protocol Version (0-15): ");
    scanf("%d", &value);
    if (value >= 0 && value <= 15) {
        packet.version = value;
    } else {
        printf("Invalid input! Version must be between 0 and 15.\n");
        return 1;
```

```
    }

    printf("Enter Internet Header Length (0-15): ");
    scanf("%d", &value);
    if (value >= 0 && value <= 15) {
        packet.IHL = value;
    } else {
        printf("Invalid input! IHL must be between 0 and 15.\n");
        return 1;
    }

    printf("Enter Type of Service (0-255): ");
    scanf("%d", &value);
    if (value >= 0 && value <= 255) {
        packet.type_of_service = value;
    } else {
        printf("Invalid input! Type of Service must be between 0 and
255.\n");
        return 1;
    }

    printf("Enter Total Packet Length (0-65535): ");
    scanf("%d", &value);
    if (value >= 0 && value <= 65535) {
        packet.total_length = value;
    } else {
        printf("Invalid input! Total Length must be between 0 and
65535.\n");
        return 1;
    }

    display(packet);

    return 0;
}
void display(struct PacketHeader packet) {
    printf("\nPacket Header Details:\n");
    printf("Version: %u\n", packet.version);
    printf("IHL (Internet Header Length): %u\n", packet.IHL);
    printf("Type of Service: %u\n", packet.type_of_service);
    printf("Total Length: %u\n", packet.total_length);
}
```

Problem 5: Employee Work Hours Tracking
Problem Statement:
Write a C program to track employee work hours using bit-fields. The program should:
1. Define a structure named WorkHours with bit-fields:
o days_worked (7 bits): Number of days worked in a week (0-7).
o hours_per_day (4 bits): Average number of hours worked per day (0-15).
2. Allow the user to input the number of days worked and the average hours per day for an employee.
3. Calculate and display the total hours worked in the week.

```c
#include <stdio.h>
struct WorkHours {
    unsigned int days_worked : 7;
    unsigned int hours_per_day : 4;
};
int main() {
    struct WorkHours employee;
    int days, hours;

    printf("Employee Work Hours Tracking System\n");

    printf("Enter the number of days worked in a week (0-7): ");
    scanf("%d", &days);
    if (days >= 0 && days <= 7) {
        employee.days_worked = days;
    } else {
        printf("Invalid input! Days worked must be between 0 and
7.\n");
        return 1;
    }


    printf("Enter the average number of hours worked per day (0-15):
");
    scanf("%d", &hours);
    if (hours >= 0 && hours <= 15) {
        employee.hours_per_day = hours;
    } else {
        printf("Invalid input! Hours per day must be between 0 and
15.\n");
        return 1;
    }
```

```c
    int total_hours = employee.days_worked * employee.hours_per_day;


    printf("\nEmployee Work Hours Details:\n");
    printf("Days Worked: %u\n", employee.days_worked);
    printf("Average Hours per Day: %u\n", employee.hours_per_day);
    printf("Total Hours Worked in the Week: %d\n", total_hours);

    return 0;
}
```