```
//Accessing structure member through pointer using dynamic memory allocation
 #include <stdio.h>
 #include <stdlib.h>


struct course{
        int marks;
        char subject[30];
 };


int main(){
        struct course *ptr;
        int noOfRecords;
        printf("Enter the number of records: ");
        scanf("%d",&noOfRecords);


        //Dynamic Memory allocation for noOfRecords
        ptr = (struct course *)malloc(noOfRecords * sizeof(struct course));


        for(int i = 0; i < noOfRecords; i++){
        printf("Enter SubjectNames and Marks \n");
        scanf("%s %d",(ptr + i)->subject, &(ptr + i)->marks);
        }


        //Display the information


        printf("Displaying Information:\n");
        for(int i = 0; i < noOfRecords; i++){
        printf("%s\t%d\n",(ptr+i)->subject,(ptr+i)->marks);
        }


        free(ptr);
        return 0;
 }
```

Problem Statement: Employee Records Management

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

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.

Dynamically allocate memory for storing information about n employees (where n is input by the user).

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
```

Answer:

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

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

void inputDetails(struct Employee *emp, int n);
void displayDetails(struct Employee *emp, int n);
void searchByID(struct Employee *emp, int n, int search_id);

int main()
{
    int n, search_id;

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



    struct Employee *employees = (struct Employee *)malloc(n * sizeof(struct Employee));

    inputDetails(employees, n);

    printf("\nEmployee Details:\n");
    displayDetails(employees, n);

    printf("\nEnter ID to search for: ");
    scanf("%d", &search_id);
    searchByID(employees, n, search_id);
    free(employees);

    return 0;
}

    void inputDetails(struct Employee *emp, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("\nEnter details of employee %d:\n", i + 1);

        int flag = 0;

        while (0==flag)
        {
            flag= 1;
            printf("ID: ");
            scanf("%d", &emp[i].id);

            for (int j = 0; j < i; j++)
            {
                if (emp[i].id == emp[j].id)
                {
                    flag = 0;
                    printf("ID already exists. Please enter a unique ID.\n");
                    break;
                }
            }
        }

        printf("Name: ");
        scanf(" %[^\n]", emp[i].name);
        getchar();
        printf("Salary: ");
        scanf("%f", &emp[i].salary);
    }
}

void displayDetails(struct Employee *emp, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("ID: %d\nName: %s\nSalary: %.2f\n", emp[i].id, emp[i].name, emp[i].salary);
    }
```

```
}

void searchByID(struct Employee *emp, int n, int search_id)
{
    for (int i = 0; i < n; i++)
    {
        if (emp[i].id == search_id)
        {
            printf("ID: %d\nName: %s\nSalary: %.2f\n", emp[i].id, emp[i].name, emp[i].salary);
            return;
        }
    }
    printf("Employee with ID %d not found.\n", search_id);
}
```

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_inventory
```

```
{
    int id;
    char title[50];
    float price;
};

void inputDetails(struct Book_inventory *, int n);
void displayDetails(struct Book_inventory *, int n);
void find_cheapest(struct Book_inventory *, int n);
void update_price(struct Book_inventory *, int up_id, int n);

int main()
{
    int n, choice, up_id;
    struct Book_inventory *books = NULL;

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

    books = (struct Book_inventory *)malloc(n * sizeof(struct Book_inventory));
    inputDetails(books, n);

    while (1)
    {
```

```
printf("\nMenu:\n");
printf("1. Display All Books\n");
printf("2. Find Cheapest Book\n");
printf("3. Update Book Price\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice)
{
case 1:
   printf("\nBook Details:\n");
   displayDetails(books, n);
   break;

case 2:
   find_cheapest(books, n);
   break;

case 3:
   printf("\nEnter book ID to update price: ");
   scanf("%d", &up_id);
   update_price(books, up_id, n);
   break;
```

```
        case 4:
            printf("Exiting program.\n");
            free(books);
            return 0;

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

    return 0;
}

void inputDetails(struct Book_inventory *books, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("\nEnter details of book %d:\n", i + 1);

        int flag = 0;

        while (flag == 0)
        {
            flag = 1;
            printf("ID: ");
            scanf("%d", &books[i].id);

            for (int j = 0; j < i; j++)
            {
                if (books[i].id == books[j].id)
                {
                    flag = 0;
                    printf("ID already exists. Please enter a unique ID.\n");
                    break;
                }
            }
        }

        printf("Title: ");
        scanf(" %[^\n]", books[i].title);
        printf("Price: ");
        scanf("%f", &books[i].price);
    }
}

void displayDetails(struct Book_inventory *books, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("ID: %d\nTitle: %s\nPrice: %.2f\n", books[i].id, books[i].title, books[i].price);
    }
}

void find_cheapest(struct Book_inventory *books, int n)
{
    int cheapest_index = 0;

    for (int i = 1; i < n; i++)
    {
        if (books[i].price < books[cheapest_index].price)
        {
            cheapest_index = i;
        }
    }

    printf("\nCheapest book in the inventory:\n");
    printf("ID: %d\nTitle: %s\nPrice: %.2f\n", books[cheapest_index].id,
books[cheapest_index].title, books[cheapest_index].price);
}

void update_price(struct Book_inventory *books, int up_id, int n)
{
    float new_price;
    int found = 0;

    for (int i = 0; i < n; i++)
    {
        if (books[i].id == up_id)
        {
            printf("\nEnter new price: ");
            scanf("%f", &new_price);

            books[i].price = new_price;
            printf("Price updated successfully for book ID %d.\n", up_id);
            found = 1;
            break;
        }
    }

    if (!found)
    {
        printf("Book with ID %d not found.\n", up_id);
    }
}
```

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.

   Answer:

```
#include <stdio.h>

#include <stdlib.h>



struct Point

{

   float x;

   float y;

};



float calculateDistance(struct Point p1, struct Point p2);

void inputPoints(struct Point *points, int n);

void displayPoints(struct Point *points, int n);
void findClosestPair(struct Point *points, int n);


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



    struct Point *points = (struct Point *)malloc(n * sizeof(struct Point));



    inputPoints(points, n);
    displayPoints(points, n);


    int p1, p2;
    printf("\nEnter indices of two points to calculate the distance (1 to %d): ", n);
    scanf("%d %d", &p1, &p2);


        float distance = calculateDistance(points[p1 - 1], points[p2 - 1]);
        printf("Distance squared between Point %d and Point %d: %.2f\n", p1, p2,
distance);



    findClosestPair(points, n);
    free(points);

    return 0;
}

float calculateDistance(struct Point p1, struct Point p2)
{
    float dx = p1.x - p2.x;
    float dy = p1.y - p2.y;
    return (dx * dx + dy * dy);
}

void inputPoints(struct Point *points, int n)
{
    for (int i = 0; i < n; i++) {
        printf("Enter coordinates for point %d (x y): ", i + 1);
        scanf("%f %f", &points[i].x, &points[i].y);
    }
}

void displayPoints(struct Point *points, int n)
{
    printf("\nCoordinates of the points:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Point %d: (%.2f, %.2f)\n", i + 1, points[i].x, points[i].y);
    }
}

void findClosestPair(struct Point *points, int n)
{
    float minDistance = 500;
    int p1Index = -1, p2Index = -1;

    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            float distance = calculateDistance(points[i], points[j]);
            if (distance < minDistance)
            {
                minDistance = distance;
                p1Index = i;
                p2Index = j;
            }
        }
    }
    if (p1Index != -1 && p2Index != -1)

    {

        printf("\nClosest pair of points: Point %d and Point %d\n", p1Index + 1,
p2Index + 1);

        printf("Coordinates: (%.2f, %.2f) and (%.2f, %.2f)\n",

               points[p1Index].x, points[p1Index].y, points[p2Index].x,
points[p2Index].y);

        printf("Distance squared: %.2f\n", minDistance);

    }

}
```

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:

Define a union named Vehicle with the following members:

1.
    - 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.

Create a structure VehicleInfo that contains:

2.
    - 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.

Implement the following features:

3.

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

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


union Vehicle_union
{
    char car_model[50];
    int bike_cc;
    int bus_seats;
};
struct Vehicle_structure
{
  char type;
  union Vehicle_union data;
};
void inputdetails(struct Vehicle_structure *);
void display(struct Vehicle_structure);

int main()
{
    struct Vehicle_structure vehicle;

    while(1)
    {
    inputdetails(&vehicle);
    display(vehicle);
    }
    return 0;
}
void inputdetails(struct Vehicle_structure *v)
{

        printf("Enter vehicle type (C for Car, B for Bike, S for Bus): ");
        scanf(" %c", &v->type);

        if (v->type == 'C') {
            printf("Enter car model: ");
            scanf(" %[^\n]", v->data.car_model);

        } else if (v->type == 'B') {
            printf("Enter bike engine capacity (CC): ");
            scanf("%d", &v->data.bike_cc);

        } else if (v->type == 'S') {
            printf("Enter number of seats in the bus: ");
            scanf("%d", &v->data.bus_seats);

        } else {
            printf("Invalid input. Please try again.\n");
        }

}
void display(struct Vehicle_structure v)
{

if (v.type == 'C')
{
        printf("\nVehicle Type: Car\n");
        printf("Car Model: %s\n", v.data.car_model);
    }
    else if (v.type == 'B')
    {
        printf("\nVehicle Type: Bike\n");
        printf("Engine Capacity: %d CC\n", v.data.bike_cc);
    } else if (v.type == 'S')
    {
        printf("\nVehicle Type: Bus\n");
        printf("Number of Seats: %d\n", v.data.bus_seats);
    }
}
```

ENUMS

```
#include<stdio.h>
enum math
{
    add=2,
    sub,
    divi
};
int main()
{
    enum math var=divi;
    printf("size: %dbytes\n",sizeof(var));
    switch(var)
    {
        case 1:
        printf("Add");
        break;
        case 2:
        printf("sub");
        break;
        case 3:
        printf("div");
        break;
        default:
        printf("default");
    }
}
```

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"

Answer:

```
#include<stdio.h>
enum  TrafficLight
{
   RED=0,
   YELLOW,
   GREEN
};

int main()
{
   enum TrafficLight var;
   printf("Enter the current light (0-RED 1-YELLOW 2-GREEN) :");
   scanf("%d",&var);

   switch(var)
   {
      case 0:
      printf("\nSTOP");
      break;
      case 1:
      printf("READY TO MOVE");
```

```
        break;
        case 2:
        printf("GO");
        break;
        default:
        printf("default");
    }
}
```

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

```
 #include <stdio.h>
enum Weekday
 {
    MONDAY = 1,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
};

int main()
 {
    enum Weekday day;
    int userInput;

    printf("Enter a number (1 for MONDAY to 7 for SUNDAY): ");



    day = (enum Weekday)userInput;

    switch (day)
 {
        case 1:
            printf("MONDAY - Weekday\n");
            break;
        case 2:
            printf("TUESDAY - Weekday\n");
            break;
        case:3:
            printf("WEDNESDAY - Weekday\n");
            break;
        case 4:
            printf("THURSDAY - Weekday\n");
            break;
        case 5:
            printf("FRIDAY - Weekday\n");
            break;
        case 6:
            printf("SATURDAY - Weekend\n");
            break;
        case 7:
            printf("SUNDAY - Weekend\n");
            break;
        default:
            printf("Error: Invalid day.\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.

Answer:

```
#include <stdio.h>

enum Shape {
    CIRCLE = 0,
    RECTANGLE,
    TRIANGLE
};

int main() {
    enum Shape shape;
    int choice;
    float area;

    printf("Select a shape to calculate the area:\n");
    printf("0 - CIRCLE\n1 - RECTANGLE\n2 - TRIANGLE\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
```

```
shape = (enum Shape)choice;

switch (shape)
{
    case 0:
    {
        float radius;
        printf("Enter the radius of the circle: ");
        scanf("%f", &radius);
        area = 3.14*radius*radius;
        printf("The area of the circle is: %.2f\n", area);
        break;
    }
    case 1:
    {
    float length, breadth;
        printf("Enter the length and breadth of the rectangle: ");
        scanf("%f %f", &length, &breadth);
        area = length * breadth;
        printf("The area of the rectangle is: %.2f\n", area);
        break;
    }
    case 3:
    {
            float base, height;
            printf("Enter the base and height of the triangle: ");
            scanf("%f %f", &base, &height);

            area = 0.5 * base * height;
            printf("The area of the triangle is: %.2f\n", area);
            break;
        }
        default:
            printf("Error: Invalid shape selection.\n");
    }

    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. Print an appropriate message to the user based on the returned error code.

ANSWER:

```
#include <stdio.h>

enum ErrorCode
{
   SUCCESS= 0,
   FILE_NOT_FOUND,
   ACCESS_DENIED,
   OUT_OF_MEMORY,
   UNKNOWN_ERROR
};

int main()
{
   int choice;


   printf("\nSUCCESS (0) FILE_NOT_FOUND (1) ACCESS_DENIED (2)OUT_OF_MEMORY (3)UNKNOWN_ERROR (4)\n");
   printf("Enter your choice: ");
   scanf("%d",&choice);

   int error = (enum ErrorCode)choice;

   switch (error)
```

```
{
    case 0:
    {
        printf("SUCCESS");
        break;
    }
    case 1:
    {
       printf("FILE NOT FOUND");

        break;
    }
   case 2:
    {
    printf("ACCESS DENIED");

        break;
    }
    case 3:
    {
        printf("OUT OF MEMORY");

        break;
    }
```

```
        case 4:
        {
            printf("UNKNOWN ERROR");

            break;
        }
        default:
            printf("Error: Invalid shape selection.\n");
    }

    return 0;
}
```

```
GUEST
```

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."

Answer:

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

```
enum UserRole
{
    ADMIN=0,
    EDITOR,
    VEIWER,
    GUEST
};

int main()
{
    int choice ;


    printf("\nADMIN (0) EDITOR (1) VIEWER (2) GUEST (3)\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);

    int CH = (enum UserRole)choice;

    switch (CH)
    {
        case 0:
        {
            printf("ADMIN:Full access to the system.");
            break;
        }
        case 1:
        {
           printf("EDITOR:Can edit content but not manage users.");

            break;
        }
       case 2:
        {
        printf("VIEWER: Can view content only.");

            break;
        }
        case 3:
        {
            printf("GUEST: Limited access, view public content only.");
            break;
        }
        default:
            printf("Error: Invalid\n");
    }

    return 0;
}
```

Problem 1: Compact Date Storage

Problem Statement:

Write a C program to store and display dates using bit-fields. The program should:

Define a structure named Date with bit-fields:

day (5 bits): Stores the day of the month (1-31).

month (4 bits): Stores the month (1-12).

year (12 bits): Stores the year (e.g., 2024).

Create an array of dates to store 5 different dates.

Allow the user to input 5 dates in the format DD MM YYYY and store them in the array.

Display the stored dates in the format DD-MM-YYYY.

Answer:

```
#include <stdio.h>

struct Date
{
    unsigned int day : 5;
    unsigned int month : 4;
    unsigned int year : 12;
};

int main()
{
    struct Date dates[5];
    struct Date *date;

    for (int i = 0; i < 5; i++)
    {
        date = &dates[i];
        unsigned int day, month, year;

        printf("Enter date %d (DD MM YYYY): ", i + 1);
        scanf("%u %u %u", &day, &month, &year);
        date->day = day;
        date->month = month;
        date->year = year;
    }

    printf("\nStored Dates:\n");
    for (int i = 0; i < 5; i++)
    {
        date = &dates[i];
        printf("%u-%u-%u\n", date->day, date->month, date->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:

Define a structure named DeviceStatus with the following bit-fields:

power (1 bit): 1 if the device is ON, 0 if OFF.

connection (1 bit): 1 if the device is connected, 0 if disconnected.

error (1 bit): 1 if there's an error, 0 otherwise.

Simulate the device status by updating the bit-fields based on user input:

Allow the user to set or reset each status.

Display the current status of the device in a readable format (e.g., Power: ON, Connection: DISCONNECTED, Error: NO).

Answer:

```
#include <stdio.h>

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

int main()
 {
    struct DeviceStatus device = {0, 0, 0};
    unsigned int power, connection, error;

    printf("Enter device status:\n");

    printf("Power (0: OFF, 1: ON): ");
    scanf("%u", &power);
    device.power = power;

    printf("Connection (0: DISCONNECTED, 1: CONNECTED): ");
    scanf("%u", &connection);
    device.connection = connection;

    printf("Error (0: NO ERROR, 1: ERROR): ");
    scanf("%u", &error);
    device.error = error;

    printf("\nDevice Status:\n");
    printf("Power: %s\n", device.power ? "ON" : "OFF");
    printf("Connection: %s\n", device.connection ? "CONNECTED" : "DISCONNECTED");
    printf("Error: %s\n", device.error ? "ERROR" : "NO ERROR");

    return 0;
}
```

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

Answer:

```
#include <stdio.h>

struct FilePermissions
{
    unsigned int read : 1;
    unsigned int write : 1;
    unsigned int execute : 1;
};

int main()
{
    struct FilePermissions permissions = {0, 0, 0};

    unsigned int read, write, execute;

    printf("Set or clear permissions (1: Grant, 0: Deny):\n");
    printf("Read permission: ");
    scanf("%u", &read);
    permissions.read = read;

    printf("Write permission: ");
    scanf("%u", &write);
    permissions.write = write;

    printf("Execute permission: ");
    scanf("%u", &execute);
    permissions.execute = execute;

    printf("\nCurrent Permissions:\n");
    printf("R:%u W:%u X:%u\n", permissions.read, permissions.write, permissions.execute);

    return 0;
}
```

Problem 4: Network Packet Header
Problem Statement:
Write a C program to represent a network packet header using bit-fields. The program should:
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.
Allow the user to input values for each field and store them in the structure.
Display the packet header details in a structured format.
Answer:

```
#include <stdio.h>

struct PacketHeader
{
    unsigned int version : 4;
    unsigned int IHL : 4;
    unsigned int type_of_service : 8;
    unsigned int total_length : 16;
};

int main()
{
    struct PacketHeader packet;

    unsigned int version, IHL, type_of_service, total_length;

    printf("Enter the network packet details:\n");
    printf("Version (0-15): ");
    scanf("%u", &version);
    packet.version = version;

    printf("IHL (0-15): ");
    scanf("%u", &IHL);
    packet.IHL = IHL;

    printf("Type of Service (0-255): ");
    scanf("%u", &type_of_service);
    packet.type_of_service = type_of_service;

    printf("Total Length (0-65535): ");
    scanf("%u", &total_length);
    packet.total_length = total_length;

    printf("\nPacket Header:\n");
    printf("Version: %u\n", packet.version);
    printf("IHL: %u\n", packet.IHL);
    printf("Type of Service: %u\n", packet.type_of_service);
    printf("Total Length: %u\n", packet.total_length);

    return 0;
}
```

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

```
#include <stdio.h>

struct WorkHours
{
   unsigned int days_worked : 7;
   unsigned int hours_per_day : 4;
};

int main()
{
   struct WorkHours employee;

   unsigned int days_worked, hours_per_day;

   printf("Enter the number of days worked: ");
   scanf("%u", &days_worked);
   employee.days_worked = days_worked;

   printf("Enter the average hours worked per day: ");
   scanf("%u", &hours_per_day);
   employee.hours_per_day = hours_per_day;

   unsigned int total_hours = employee.days_worked * employee.hours_per_day;

   printf("\nEmployee's Total Hours Worked: %u hours\n", total_hours);

   return 0;
}
```