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
```

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

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

int main() {
   int n, search_id;
   struct Employee *employees;
```

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



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

for (int i = 0; i < n; i++) {
    int id_unique = 1;

    printf("Enter details of employee %d:\n", i + 1);

    do {
        printf("ID: ");
        scanf("%d", &employees[i].id);



        for (int j = 0; j < i; j++) {
            if (employees[j].id == employees[i].id) {
                printf("Error: ID %d is already taken. Please enter a unique ID.\n", employees[i].id);
                id_unique = 0;
                break;
            }
        }
    } while (id_unique == 0);

    getchar();

    printf("Name: ");
    scanf("%[^\n]", employees[i].name);
```

```
        printf("Salary: ");
        scanf("%f", &employees[i].salary);
    }

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


    printf("Enter ID to search for: ");
    scanf("%d", &search_id);

    int id_found = 0;
    for (int i = 0; i < n; i++) {
        if (employees[i].id == search_id) {
            printf("Search Result:\n");
            printf("ID: %d \nName: %s \nSalary: %.2f\n", employees[i].id, employees[i].name,
employees[i].salary);
            id_found = 1;
            break;
        }
    }

    if (!id_found) {
        printf("ID not found\n");
    }


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

Output:

```
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
Employee details:
ID: 101
Name: Alice
Salary: 50000.00
ID: 102
Name: Bob
Salary: 60000.00
ID: 103
Name: Charlie
Salary: 55000.00
Enter ID to search for: 102
Search Result:
ID: 102
Name: Bob
```

Salary: 60000.00

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

```
books=(struct Book*)malloc(n*sizeof(struct Book));
for(int i=0;i<n;i++)
{
   int id_unique = 0;
   do {
      printf("ID: ");
      scanf("%d", &books[i].id);
      id_unique = 1;
      for (int j = 0; j < i; j++) {
         if (books[j].id == books[i].id) {
            printf("ID %d already taken. Please enter a unique ID.\n", books[i].id);
            id_unique = 0;
            break;
         }
      }
   } while (!id_unique);
   getchar();
   printf("Title:");
   scanf("%[^\n]",books[i].title);
   printf("Price:");
   scanf("%f",&books[i].price);
}
printf("Book Inventory Details:\n");
for(int i=0;i<n;i++)
{
   printf("ID:%d, Title:%s, Price:%.2f\n",books[i].id,books[i].title,books[i].price);
}
int cheapest_book=0;
for(int i=0;i<n;i++)
{
   if(books[i].price<books[cheapest_book].price)
```

```
        {
            cheapest_book=i;
        }
    }
    printf("Cheapest book:\n");
    printf("ID:%d, Title:%s,
Price:%.2f\n",books[cheapest_book].id,books[cheapest_book].title,books[cheapest_book].price);
    printf("Enter the ID of the book to update price: ");
    scanf("%d",&update_id);
    int book_found=0;
    for(int i=0;i<n;i++)
    {
        if(books[i].id==update_id)
        {
            printf("Enter new price for book %d:",update_id);
            scanf("%f",&books[i].price);
            book_found=1;
            break;
        }
    }
    if(!book_found)
    {
        printf("Book with ID %d is not found \n",update_id);
    }
    else
    {
        printf("Price updated successfully \n");
    }
    printf("Updated book inventory: \n");
    for(int i=0;i<n;i++)
    {
        printf("ID:%d, Title:%s, Price:%.2f\n",books[i].id,books[i].title,books[i].price);
    }
    free(books);
    return 0;
}
```

Output:

```
Enter the number of books:3
ID: 1001
Title:The great gatsby
Price:500
ID: 1002
Title:Pride and prejudice
Price:300
ID: 1003
Title:Harry potter
Price:700
Book Inventory Details:
ID:1001, Title:The great gatsby, Price:500.00
ID:1002, Title:Pride and prejudice, Price:300.00
ID:1003, Title:Harry potter, Price:700.00
Cheapest book:
ID:1002, Title:Pride and prejudice, Price:300.00
Enter the ID of the book to update price: 1002
Enter new price for book 1002:450
Price updated successfully
Updated book inventory:
ID:1001, Title:The great gatsby, Price:500.00
ID:1002, Title:Pride and prejudice, Price:450.00
ID:1003, Title:Harry potter, Price:700.00
```

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.

```
#include <stdio.h>

#include <stdlib.h>


struct Point {

   float x;

   float y;

};


int main() {

   int n;

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

   scanf("%d", &n);


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


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

      printf("Enter coordinates of point %d:\n", i + 1);
```

```
    printf("x: ");
    scanf("%f", &points[i].x);
    printf("y: ");
    scanf("%f", &points[i].y);
}

printf("Points entered:\n");
for (int i = 0; i < n; i++) {
    printf("Point %d: (%.2f, %.2f)\n", i + 1, points[i].x, points[i].y);
}

int index1, index2;
printf("Enter indices (1 to %d) of two points to find the distance between them: ", n);
scanf("%d %d", &index1, &index2);

if (index1 > 0 && index1 <= n && index2 > 0 && index2 <= n) {
    float dx = points[index1 - 1].x - points[index2 - 1].x;
    float dy = points[index1 - 1].y - points[index2 - 1].y;
    float distance = dx * dx + dy * dy;

    printf("Squared distance between Point %d and Point %d: %.2f\n", index1, index2, distance);
} else {
    printf("Invalid indices!\n");
}

float minDistance = -1;
int closestPair1 = -1, closestPair2 = -1;
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        float dx = points[i].x - points[j].x;
        float dy = points[i].y - points[j].y;
```

```
            float distance = dx * dx + dy * dy;

            if (minDistance == -1 || distance < minDistance) {
                minDistance = distance;
                closestPair1 = i;
                closestPair2 = j;
            }
        }
    }

    if (closestPair1 != -1 && closestPair2 != -1) {
        printf("The closest pair of points is Point %d and Point %d with a squared distance of %.2f\n",
               closestPair1 + 1, closestPair2 + 1, minDistance);
    }

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

Output:

```
Enter the number of points: 3
Enter coordinates of point 1:
x: 2
y: 3
Enter coordinates of point 2:
x: 4
y: 5
Enter coordinates of point 3:
x: 6
y: 7
Points entered:
Point 1: (2.00, 3.00)
Point 2: (4.00, 5.00)

Point 3: (6.00, 7.00)

Enter indices (1 to 3) of two points to find the distance between them: 2

3

Squared distance between Point 2 and Point 3: 8.00

The closest pair of points is Point 1 and Point 2 with a squared distance of 8.00
```

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

```
#include<stdio.h>

#include<stdlib.h>


union Vehicle

{

   char car_model[50];

   int bike_cc;

   int bus_seats;

};


struct VehicleInfo

{

   char type;

   union Vehicle vehicle;

};


int main()

{

   struct VehicleInfo vehicle_info;

   char vehicle_type;

   printf("Enter vehicle type(C for car, B for bike, S for bus):");

   scanf("%c", &vehicle_type);

   getchar();


   while(vehicle_type != 'C' && vehicle_type != 'B' && vehicle_type != 'S')

   {

      printf("Invalid input\n");

      printf("Enter vehicle type(C for car, B for bike, S for bus):");

      scanf("%c", &vehicle_type);

      getchar();

   }
```

```
    vehicle_info.type = vehicle_type;

    if(vehicle_info.type == 'C')
    {
        printf("Enter car model:");
        scanf("%[^\n]", vehicle_info.vehicle.car_model);
        printf("Vehicle type: Car\n");
        printf("Car Model: %s\n", vehicle_info.vehicle.car_model);
    }
    else if(vehicle_info.type == 'B')
    {
        printf("Enter bike engine capacity (CC):");
        scanf("%d", &vehicle_info.vehicle.bike_cc);
        printf("Vehicle type: Bike\n");
        printf("Engine capacity: %d CC\n", vehicle_info.vehicle.bike_cc);
    }
    else if(vehicle_info.type == 'S')
    {
        printf("Enter number of seats in the bus:");
        scanf("%d", &vehicle_info.vehicle.bus_seats);
        printf("Vehicle type: Bus\n");
        printf("Number of seats: %d\n", vehicle_info.vehicle.bus_seats);
    }

    return 0;
}
```

Output:

```
Enter vehicle type(C for car, B for bike, S for bus):C
Enter car model:Toyota Corolla
Vehicle type: Car
```

Car Model: Toyota Corolla

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"

```
#include<stdio.h>
enum TrafficLight{
  RED,
  YELLOW,
  GREEN
};
int main()
{
  int light;
  printf("Enter the current light colour(0 for RED,1 for YELLOW,2 for GREEN):");
  scanf("%d",&light);
  if(light==RED)
  {
    printf("Stop \n");
  }
  else if(light==YELLOW)
  {
    printf("Ready to move \n");
  }
  else if(light==GREEN)
  {
    printf("Go \n");
  }
  else
  {
    printf("Invalid input \n");
  }

  return 0;
}
```

Output:

```
Enter the current light colour(0 for RED,1 for YELLOW,2 for GREEN):1
Ready to move
```

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()
{
    int day;
    printf("Enter the number:");
    scanf("%d",&day);
    if(day>=MONDAY && day<=SUNDAY)
    {
        printf("Day:");
        switch(day)
        {
            case MONDAY:
            printf("MONDAY \n");
            break;
            case TUESDAY:
            printf("TUESDAY \n");
            break;
            case WEDNESDAY:
            printf("WEDNESDAY \n");
            break;
            case THURSDAY:
            printf("THURSDAY \n");
            break;
            case FRIDAY:
            printf("FRIDAY \n");
            break;
            case SATURDAY:
            printf("SATURDAY \n");
            break;
            case SUNDAY:
            printf("SUNDAY \n");
            break;
        }
        if(day==SATURDAY || day ==SUNDAY)
        {
            printf("It is weekend \n");
        }
        else
        {
            printf("It is weekday \n");
        }
    }else
    {
        printf("Invalid input \n");
    }
    return 0;
}
```

Output:

```
Enter the number:5
Day:FRIDAY
It is weekday
```

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.

```
 #include<stdio.h>
enum Shape{
   CIRCLE,
   RECTANGLE,
   TRIANGLE
};
int main()
{
   int choice;
   printf("Select a shape:");
   scanf("%d",&choice);
   if(choice==CIRCLE)
   {
      float radius;
      printf("Enter the radius:");
      scanf("%f",&radius);
      printf("Area of circle:%.2f\n",3.14*radius*radius);
   }
   else if(choice==RECTANGLE)
   {
      float length,breadth;
      printf("Enter the length:");
      scanf("%f",&length);
      printf("Enter the breadth:");
```

```
        scanf("%f",&breadth);
        printf("Area of rectangle:%.2f\n",length*breadth);
    }
    else if(choice==TRIANGLE)
    {
        float base,height;
        printf("Enter the base:");
        scanf("%f",&base);
        printf("Enter the height:");
        scanf("%f",&height);
        printf("Area of triangle:%.2f\n",0.5*base*height);
    }
    else
    {
        printf("Invalid input \n");
    }
    return 0;
}
```

Output:

```
Select a shape:1
Enter the length:34
Enter the breadth:43
Area of rectangle:1462.00
```

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>
enum ErrorCode
{
   SUCCESS,
   FILE_NOT_FOUND,
   ACCESS_DENIED,
   OUT_OF_MEMORY,
   UNKNOWN_ERROR
};
int main()
{
   enum ErrorCode error;
   printf("Enter the error code:");
   scanf("%u",&error);
   switch(error)
   {
      case SUCCESS:
      printf("Code executed successfully\n");
      break;
      case FILE_NOT_FOUND:
      printf("Error:File not found \n");
      break;
      case ACCESS_DENIED:
      printf("Error:Access denied \n");
      break;
```

```
        case OUT_OF_MEMORY:
        printf("Error:Out of memory \n");
        break;
        case UNKNOWN_ERROR:
        printf("Error:Unknown error \n");
        break;
        default:
        printf("Invalid error");
    }
    return 0;
}
```

Output:

```
Enter the error code:2
Error:Access denied
```

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>
enum UserRole{
    ADMIN,
    EDITOR,
    VIEWER,
    GUEST
};
int main()
{
    int user_role;
    printf("Enter the user role:");
    scanf("%d",&user_role);
    switch(user_role)
    {
       case ADMIN:
       printf("Admin:Full access to the system \n");
       break;
       case EDITOR:
       printf("Editor:Can edit the content but not manage the users \n");
       break;
       case VIEWER:
       printf("Viewer:Can view content only \n");
       break;
       case GUEST:
       printf("Guest:Limited access,view public content only \n");
       break;
       default:
       printf("Invalid user \n");
    }
    return 0;
}
```

Output:

```
Enter the user role:3
Guest:Limited access,view public content only
```

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

```
    printf("Stored dates:\n");
    for(int i=0;i<5;i++)
    {
        printf("Date%d: %u-%u-%u\n",i+1,dates[i].day,dates[i].month,dates[i].year);
    }
    return 0;
}
```

Output:

```
Enter 5 dates(DD MM YYYY):
Date1:24 06 2001
Date2:27 08 2005
Date3:30 11 2009
Date4:17 07 2020
Date5:07 05 2024
Stored dates:
Date1: 24-6-2001
Date2: 27-8-2005
Date3: 30-11-2009
Date4: 17-7-2020
Date5: 7-5-2024
```

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:
   - 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.
2. Simulate the device status by updating the bit-fields based on user input:
   - 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;
};
int main()
{
   struct DeviceStatus device={0,0,0};
   int power,connection,error;
   printf("Enter device status:\n");
   printf("power(1 for ON,0 for OFF):");
   scanf("%d",&power);
   printf("Connection(1 for CONNECTED, 0 for DISCONNECTED):");
   scanf("%d",&connection);
   printf("Error(1 for YES,0 for NO):");
   scanf("%d",&error);
   device.power = power;
   device.connection = connection;
   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 ? "YES" : "NO");
   return 0;
}
```

Output:

```
Enter device status:
power(1 for ON,0 for OFF):1
```

```
Connection(1 for CONNECTED, 0 for DISCONNECTED):1
Error(1 for YES,0 for NO):0ERROR!


Device Status:
Power: ON
Connection: CONNECTED
Error: 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:
   - 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.
2. 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).

```c
 #include <stdio.h>
struct FilePermissions {
   unsigned int read:1;
   unsigned int write:1;
   unsigned int execute:1;
};

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

```
    int readPerm, writePerm, execPerm;

    printf("Enter file permissions:\n");

    printf("Read (1 for granted, 0 for denied): ");
    scanf("%d", &readPerm);

    printf("Write (1 for granted, 0 for denied): ");
    scanf("%d", &writePerm);

    printf("Execute (1 for granted, 0 for denied): ");
    scanf("%d", &execPerm);

    filePerm.read = readPerm;
    filePerm.write = writePerm;
    filePerm.execute = execPerm;

    printf("\nCurrent File Permissions:\n");
    printf("R: %d W: %d X: %d\n", filePerm.read, filePerm.write, filePerm.execute);

    return 0;
}
```

Output:

```
Enter file permissions:
Read (1 for granted, 0 for denied): 1
Write (1 for granted, 0 for denied): 1
Execute (1 for granted, 0 for denied): 1

Current File Permissions:
R: 1 W: 1 X: 1
```

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;
};

int main() {
   struct PacketHeader packet;

   unsigned int version, IHL, type_of_service, total_length;

   printf("Enter Packet Header Details:\n");

   printf("Version (4 bits, 0-15): ");
   scanf("%u", &version);

   printf("IHL (4 bits, 0-15): ");
   scanf("%u", &IHL);
```

```
    printf("Type of Service (8 bits, 0-255): ");
    scanf("%u", &type_of_service);

    printf("Total Length (16 bits, 0-65535): ");
    scanf("%u", &total_length);

    packet.version = version;
    packet.IHL = IHL;
    packet.type_of_service = type_of_service;
    packet.total_length = total_length;

    printf("\nNetwork Packet Header Details:\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;
}
```

Output:

```
Enter Packet Header Details:
Version (4 bits, 0-15): 4
IHL (4 bits, 0-15): 5
Type of Service (8 bits, 0-255): 32
Total Length (16 bits, 0-65535): 1024

Network Packet Header Details:
Version: 4
IHL: 5
Type of Service: 32
Total Length: 1024
```

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:
   - 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).
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.

```
#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 (0-7): ");
  scanf("%u", &days_worked);

  printf("Enter the average number of hours worked per day (0-15): ");
  scanf("%u", &hours_per_day);

  employee.days_worked = days_worked;
  employee.hours_per_day = hours_per_day;

  unsigned 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("Hours Worked Per Day: %u\n", employee.hours_per_day);
    printf("Total Hours Worked in the Week: %u\n", total_hours);

    return 0;
}
```

Output:

```
Enter the number of days worked (0-7): 5
Enter the average number of hours worked per day (0-15): 9

Employee Work Hours Details:
Days Worked: 5
Hours Worked Per Day: 9
Total Hours Worked in the Week: 45
```