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

/*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 */

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

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

```
void addEmployees(struct employee *ptr, int n);
void displayEmployees(struct employee *ptr, int n);
void searchByID(struct employee *ptr, int n);
int isDuplicateID(struct employee *ptr, int currentCount, int id) ;

int main() {
   struct employee *ptr = NULL;
   int choice, numEmployees = 0;
   int a=0;

   while(a!=1)
   {
      printf("1. Add Employees\n");
      printf("2. Display All Employees\n");
      printf("3. Search Employee by ID\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
         case 1:
            printf("Enter the number of employees to add: ");
            scanf("%d", &numEmployees);

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

            addEmployees(ptr, numEmployees);
```

```
            break;

        case 2:
            if (ptr == NULL || numEmployees == 0) {
                printf("No employees to display.\n");
            }
            else {
                displayEmployees(ptr, numEmployees);
            }
            break;

        case 3:
            if (ptr == NULL || numEmployees == 0) {
                printf("No employees to search.\n");
            }
            else {
                searchByID(ptr, numEmployees);
            }
            break;

        case 4:
            printf("Exiting the program. Freeing memory.\n");
            free(ptr);
            a=1;
            break;

    }
}
```

```
    return 0;
}

void addEmployees(struct employee *ptr, int n)
{
    for (int i = 0; i < n; i++)
    {
        int id;
        printf("Enter details of employee %d:\n", i + 1);

        while (1)
        {
            printf("ID: ");
            scanf("%d", &id);

            if (isDuplicateID(ptr, i, id))
            {
                printf("Error: ID %d already exists. Please enter a unique ID.\n", id);
            }
            else
            {
                (ptr + i)->id = id;
                break;
            }
        }

        printf("Name: ");
```

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

void displayEmployees(struct employee *ptr, int n)
{
    printf("\n Employee Details \n");
    for (int i = 0; i < n; i++)
    {
        printf("ID: %d, Name: %s, Salary: %.2f\n", (ptr + i)->id, (ptr + i)->name, (ptr + i)->salary);
    }
}

void searchByID(struct employee *ptr, int n)
{
    int searchID, found = 0;

    printf("Enter the ID of the employee to search: ");
    scanf("%d", &searchID);

    for (int i = 0; i < n; i++)
    {
        if ((ptr + i)->id == searchID)
        {
            printf("Employee Found: ID: %d, Name: %s, Salary: %.2f\n", (ptr + i)->id, (ptr + i)->name, (ptr + i)->salary);
```

```
            found = 1;
            break;
        }
    }

    if (found==0)
    {
        printf("Employee with ID %d not found.\n", searchID);
    }
}

int isDuplicateID(struct employee *ptr, int currentCount, int id)
{
    for (int i = 0; i < currentCount; i++)
    {
        if ((ptr + i)->id == id)
        {
            return 1;
        }
    }
    return 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;
};

void addBooks(struct book *ptr, int n);
void displayBooks(struct book *ptr, int n);
void findCheapestBook(struct book *ptr, int n);
void updateBookPrice(struct book *ptr, int n);
int isDuplicateID(struct book *ptr, int currentCount, int id);
```

```
int main()
{
   struct book *ptr = NULL;
   int choice, numBooks = 0;
   int a = 0;

   while (a!=1)
   {

      printf("1. Add Books\n");
      printf("2. Display All Books\n");
      printf("3. Find Cheapest Book\n");
      printf("4. Update Book Price\n");
      printf("5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
         case 1:
            printf("Enter the number of books to add: ");
            scanf("%d", &numBooks);

            ptr = (struct book *)malloc(numBooks * sizeof(struct book));



            addBooks(ptr, numBooks);
            break;
```

```
case 2:
   if (ptr == NULL || numBooks == 0)
   {
      printf("No books to display.\n");
   }
   else
   {
      displayBooks(ptr, numBooks);
   }
   break;

case 3:
   if (ptr == NULL || numBooks == 0)
   {
      printf("No books available.\n");
   }
   else
   {
      findCheapestBook(ptr, numBooks);
   }
   break;

case 4:

   if (ptr == NULL || numBooks == 0)
   {
      printf("No books to update.\n");
```

```
            }
            else
            {
               updateBookPrice(ptr, numBooks);
            }
            break;

         case 5:
            printf("Exiting the program. Freeing memory.\n");
            free(ptr);
            a = 1;
            break;

      }
   }

   return 0;
}

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

      while (1) {
         printf("ID: ");
```

```
        scanf("%d", &id);

        if (isDuplicateID(ptr, i, id))
        {
            printf("Error: ID %d already exists. Please enter a unique ID.\n", id);
        }
        else
        {
            (ptr + i)->id = id;
            break;
        }
      }

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

void displayBooks(struct book *ptr, int n)
{
   printf("\n Book Details \n");
   for (int i = 0; i < n; i++)
   {
      printf("ID: %d, Title: %s, Price: %.2f\n", (ptr + i)->id, (ptr + i)->title, (ptr + i)->price);
   }
}
```

```
void findCheapestBook(struct book *ptr, int n)
{
   int cheapestIndex = 0;

   for (int i = 1; i < n; i++) {
      if ((ptr + i)->price < (ptr + cheapestIndex)->price) {
         cheapestIndex = i;
      }
   }

   printf("\nCheapest Book:\n");
   printf("ID: %d, Title: %s, Price: %.2f\n", (ptr + cheapestIndex)->id, (ptr +
cheapestIndex)->title, (ptr + cheapestIndex)->price);
}

void updateBookPrice(struct book *ptr, int n)
{
   int id, found = 0;

   printf("Enter the ID of the book to update: ");
   scanf("%d", &id);

   for (int i = 0; i < n; i++)
   {
      if ((ptr + i)->id == id)
      {
```

```
            printf("Enter new price: ");
            scanf("%f", &(ptr + i)->price);
            printf("Price updated successfully.\n");
            found = 1;
            break;
        }
    }

    if (found==0)
    {
        printf("Book with ID %d not found.\n", id);
    }
}

int isDuplicateID(struct book *ptr, int currentCount, int id)
{
    for (int i = 0; i < currentCount; i++)
    {
        if ((ptr + i)->id == id)
        {
            return 1;
        }
    }
    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.

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

struct Point {
  float x;
  float y;
};

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

```
int main()
{
   struct Point *points = NULL;
   int numPoints = 0;
   int choice, a = 0;

   while (a!=1) {
      printf("\nMenu:\n");
      printf("1. Add Points\n");
      printf("2. Display Points\n");
      printf("3. Calculate Distance Between Two Points\n");
      printf("4. Find Closest Pair of Points\n");
      printf("5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the number of points to add: ");
            scanf("%d", &numPoints);

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

            addPoints(points, numPoints);
            break;

         case 2:
```

```
    if (points == NULL || numPoints == 0)
    {
        printf("No points to display.\n");
    }
    else
    {
        displayPoints(points, numPoints);
    }
    break;

case 3:
    if (points == NULL || numPoints == 0)
    {
        printf("No points available.\n");
    }
    else
    {
        calculateDistance(points, numPoints);
    }
    break;

case 4:
    if (points == NULL || numPoints == 0)
    {
        printf("No points available.\n");
    }
    else
    {
```

```
            findClosestPair(points, numPoints);
        }
        break;

      case 5:
        printf("Exiting the program. Freeing memory.\n");
        free(points);
        a=1;
        break;

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

  return 0;
}

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

void displayPoints(struct Point *points, int n)
```

```
{
    printf("\nPoints in 2D Space:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Point %d: (%.2f, %.2f)\n", i + 1, (points + i)->x, (points + i)->y);
    }
}

void calculateDistance(struct Point *points, int n)
{
    int index1, index2;
    printf("Enter the indices of the two points (1 to %d): ", n);
    scanf("%d %d", &index1, &index2);

    if (index1 < 1 || index1 > n || index2 < 1 || index2 > n)
    {
        printf("Invalid indices. Please enter values between 1 and %d.\n", n);
        return;
    }

    float distance = sqrt(pow((points + index1 - 1)->x - (points + index2 - 1)->x, 2) +
                          pow((points + index1 - 1)->y - (points + index2 - 1)->y, 2));
    printf("Distance between Point %d and Point %d: %.2f\n", index1, index2, distance);
}

void findClosestPair(struct Point *points, int n)
{
    if (n < 2) {
```

```
    printf("Not enough points to find the closest pair.\n");
    return;
  }

  int index1 = 0, index2 = 1;
  float minDistance = sqrt(pow((points + index1)->x - (points + index2)->x, 2) +
                           pow((points + index1)->y - (points + index2)->y, 2));

  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      float distance = sqrt(pow((points + i)->x - (points + j)->x, 2) +
                            pow((points + i)->y - (points + j)->y, 2));
      if (distance < minDistance) {
        minDistance = distance;
        index1 = i;
        index2 = j;
      }
    }
  }

  printf("\nClosest Pair of Points:\n");
  printf("Point %d: (%.2f, %.2f)\n", index1 + 1, (points + index1)->x, (points + index1)->y);
  printf("Point %d: (%.2f, %.2f)\n", index2 + 1, (points + index2)->x, (points + index2)->y);
  printf("Distance: %.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:

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>
#include <string.h>

union Vehicle
{
   char car_model[50];
   int bike_cc;
   int bus_seats;
};

struct VehicleInfo
{
   char type;
   union Vehicle details;
};

void inputVehicleDetails(struct VehicleInfo *v);
void displayVehicleDetails(const struct VehicleInfo *v);

int main()
{
   struct VehicleInfo vehicle;

   inputVehicleDetails(&vehicle);
   displayVehicleDetails(&vehicle);

   return 0;
```

```
}

void inputVehicleDetails(struct VehicleInfo *v)
{
   while (1)
   {
      printf("Enter vehicle type (C for Car, B for Bike, S for Bus): ");
      scanf(" %c", &v->type);

      if (v->type == 'C' || v->type == 'B' || v->type == 'S')
      {
         break;
      }
      else
      {
         printf("Invalid input. Please enter C, B, or S.\n");
      }
   }

   switch (v->type) {
      case 'C':
         printf("Enter car model: ");
         scanf("%s", v->details.car_model);
         break;

      case 'B':
         printf("Enter bike engine capacity (CC): ");
         scanf("%d", &v->details.bike_cc);
```

```
            break;

        case 'S':
            printf("Enter number of seats in the bus: ");
            scanf("%d", &v->details.bus_seats);
            break;
    }
}

void displayVehicleDetails(const struct VehicleInfo *v) {
    printf("\nVehicle Details:\n");

    switch (v->type) {
        case 'C':
            printf("Vehicle Type: Car\n");
            printf("Car Model: %s\n", v->details.car_model);
            break;

        case 'B':
            printf("Vehicle Type: Bike\n");
            printf("Engine Capacity: %d CC\n", v->details.bike_cc);
            break;

        case 'S':
            printf("Vehicle Type: Bus\n");
            printf("Number of Seats: %d\n", v->details.bus_seats);
            break;
```

```
    default:
        printf("Unknown vehicle type.\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"

```
#include <stdio.h>

enum traffic {
  red,
  yellow,
  green
};

int main() {
  int color;
  enum traffic light;
```

```
    printf("0: red  1: yellow  2: green\nEnter: ");
    scanf("%d", &color);


    light = (enum traffic)color;


    switch (light) {
        case red:
            printf("Traffic Light: RED - Stop\n");
            break;
        case yellow:
            printf("Traffic Light: YELLOW - Ready to move\n");
            break;
        case green:
            printf("Traffic Light: GREEN - Go\n");
            break;
        default:
            printf("Invalid traffic light state!\n");
    }


    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

```
#include <stdio.h>

enum Weekday { MONDAY = 1,
        TUESDAY,
        WEDNESDAY,
        THURSDAY,
        FRIDAY,
        SATURDAY,
        SUNDAY };

void displayDayInfo(enum Weekday day);

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

   printf("Enter a number (1 for MONDAY, 2 for TUESDAY, ..., 7 for SUNDAY): ");
   scanf("%d", &input);


   day = (enum Weekday)input;
   switch (day)
```

```
{
    case MONDAY:
        printf("Day: MONDAY - Weekday\n");
        break;
    case TUESDAY:
        printf("Day: TUESDAY - Weekday\n");
        break;
    case WEDNESDAY:
        printf("Day: WEDNESDAY - Weekday\n");
        break;
    case THURSDAY:
        printf("Day: THURSDAY - Weekday\n");
        break;
    case FRIDAY:
        printf("Day: FRIDAY - Weekday\n");
        break;
    case SATURDAY:
        printf("Day: SATURDAY - Weekend\n");
        break;
    case SUNDAY:
        printf("Day: SUNDAY - Weekend\n");
        break;
    default:
        printf("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.

```
#include <stdio.h>
#define pi 3.14

enum Shape
{
    CIRCLE,
    RECTANGLE,
    TRIANGLE
};

int main()
{
```

```
int choice;
enum Shape shape;

printf("Select a shape:\n");
printf("0: Circle\n");
printf("1: Rectangle\n");
printf("2: Triangle\n");
printf("Enter your choice: ");
scanf("%d", &choice);

shape = (enum Shape)choice;

switch (shape)
{
  case CIRCLE:
  {
    float radius, area;
    printf("Enter the radius of the circle: ");
    scanf("%f", &radius);

    area = pi * radius * radius;
    printf("The area of the circle is: %.2f\n", area);
    break;
  }

  case RECTANGLE:
  {
    float length, breadth, area;
```

```
        printf("Enter the length of the rectangle: ");
        scanf("%f", &length);
        printf("Enter the breadth of the rectangle: ");
        scanf("%f", &breadth);

        area = length * breadth;
        printf("The area of the rectangle is: %.2f\n", area);
        break;
    }

    case TRIANGLE:
    {
        float base, height, area;
        printf("Enter the base of the triangle: ");
        scanf("%f", &base);
        printf("Enter the height of the triangle: ");
        scanf("%f", &height);

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

    default:
        printf("Invalid Shape.\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. 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() {
    int scenario;
    enum ErrorCode error;



    printf("Select a scenario to simulate:\n");
    printf("1: File not found\n");
    printf("2: Access denied\n");
    printf("3: Out of memory\n");
    printf("4: Unknown error\n");
    printf("Enter your choice: ");
    scanf("%d", &scenario);



    error = (enum ErrorCode)scenario;



    switch (error) {
        case SUCCESS:
            printf("Operation completed successfully.\n");
            break;
        case FILE_NOT_FOUND:
            printf("Error: File not found. Please check the file path.\n");
            break;
        case ACCESS_DENIED:
            printf("Error: Access denied. You do not have permission.\n");
            break;
        case OUT_OF_MEMORY:
```

```
            printf("Error: Out of memory. Try freeing up some memory and retry.\n");
            break;
        case UNKNOWN_ERROR:
            printf("Error: Unknown error occurred. Please try again.\n");
            break;
        default:
            printf("Error: Unrecognized error code.\n");
    }

    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>

enum UserRole {
```

```
    ADMIN ,
    EDITOR ,
    VIEWER ,
    GUEST
};

int main() {
    int role;
    enum UserRole userRole;



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

    userRole = (enum UserRole)role;



    switch (userRole)
    {
        case ADMIN:
            printf("User Role: ADMIN\n");
            printf("Permissions: Full access to the system.\n");
            break;
```

```
        case EDITOR:
            printf("User Role: EDITOR\n");
            printf("Permissions: Can edit content but not manage users.\n");
            break;
        case VIEWER:
            printf("User Role: VIEWER\n");
            printf("Permissions: Can view content only.\n");
            break;
        case GUEST:
            printf("User Role: GUEST\n");
            printf("Permissions: Limited access, view public content only.\n");
            break;
        default:
            printf("Invalid user role! Please enter a valid choice (0-3).\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:

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 in the format DD MM YYYY:\n");
   int day, month, year;
   int i=0;
   while(i!=5)
   {
      printf("Enter date %d: ", i + 1);
      scanf("%d %d %d", &day, &month, &year) ;
      if(day>31 || day<1 || month>12 || month<1 || year<1)
      {
         printf("Enter valid date");
         printf("\n");
      }
```

```
    else
    {
        dates[i].day = day;
        dates[i].month = month;
        dates[i].year = year;
        i=i+1;
    }

  }

  printf("\nStored Dates:\n");
  for (int 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:
    - 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;
};
void displayStatus(struct DeviceStatus device);

void displayStatus(struct DeviceStatus device)
{
   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");
}

int main() {
   int a=0;
   int choice;
   struct DeviceStatus device;

   while (a!=1)
```

```
{
    printf("\nMenu:\n");
    printf("1. Set Power ON\n");
    printf("2. Set Power OFF\n");
    printf("3. Set Connection CONNECTED\n");
    printf("4. Set Connection DISCONNECTED\n");
    printf("5. Set Error YES\n");
    printf("6. Set Error NO\n");
    printf("7. Display Current Status\n");
    printf("8. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice)
    {
        case 1:
            device.power = 1;
            printf("Power is now ON.\n");
            break;
        case 2:
            device.power = 0;
            printf("Power is now OFF.\n");
            break;
        case 3:
            device.connection = 1;
            printf("Connection is now CONNECTED.\n");
            break;
        case 4:
```

```
            device.connection = 0;
            printf("Connection is now DISCONNECTED.\n");
            break;
        case 5:
            device.error = 1;
            printf("Error is now YES.\n");
            break;
        case 6:
            device.error = 0;
            printf("Error is now NO.\n");
            break;
        case 7:
            displayStatus(device);
            break;
        case 8:
            printf("Exiting program. Goodbye!\n");
            a=1;
            break;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}
```

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).

```
#include <stdio.h>

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

void displayPermissions(struct FilePermissions permissions) {
   printf("\nCurrent File Permissions:\n");
   printf("R:%d W:%d X:%d\n", permissions.read, permissions.write,
permissions.execute);
}

int main() {
   struct FilePermissions permissions;
   int choice;
   int a = 0;
```

```
while (a!=1) {
    printf("\nMenu:\n");
    printf("1. Grant Read Permission\n");
    printf("2. Revoke Read Permission\n");
    printf("3. Grant Write Permission\n");
    printf("4. Revoke Write Permission\n");
    printf("5. Grant Execute Permission\n");
    printf("6. Revoke Execute Permission\n");
    printf("7. Display Current Permissions\n");
    printf("8. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            permissions.read = 1;
            printf("Read permission granted.\n");
            break;
        case 2:
            permissions.read = 0;
            printf("Read permission revoked.\n");
            break;
        case 3:
            permissions.write = 1;
            printf("Write permission granted.\n");
```

```
            break;
        case 4:
            permissions.write = 0;
            printf("Write permission revoked.\n");
            break;
        case 5:
            permissions.execute = 1;
            printf("Execute permission granted.\n");
            break;
        case 6:
            permissions.execute = 0;
            printf("Execute permission revoked.\n");
            break;
        case 7:
            displayPermissions(permissions);
            break;
        case 8:
            printf("Exiting program. Goodbye!\n");
            a = 1;
            break;

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

   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:

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 displayPacketHeader(struct PacketHeader packet)
{
   printf("\nPacket 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);
}

int main()
{
    struct PacketHeader packet;
    int choice;
    int a = 0;

    while (a != 1)
    {
        printf("\nMenu:\n");
        printf("1. Set Protocol Version\n");
        printf("2. Set Internet Header Length (IHL)\n");
        printf("3. Set Type of Service\n");
        printf("4. Set Total Packet Length\n");
        printf("5. Display Packet Header Details\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1: {
                unsigned int temp;
                printf("Enter Protocol Version (0-15): ");
                scanf("%u", &temp);
```

```
    if (temp > 15)
    {
        printf("Error: Protocol Version must be between 0 and 15. Value not set.\n");
    }
    else
    {
        packet.version = temp;
        printf("Protocol Version set successfully.\n");
    }
    break;
}
case 2: {
    unsigned int temp;
    printf("Enter Internet Header Length (IHL) (0-15): ");
    scanf("%u", &temp);
    if (temp > 15)
    {
        printf("Error: IHL must be between 0 and 15. Value not set.\n");
    }
    else
    {
        packet.IHL = temp;
        printf("IHL set successfully.\n");
    }
    break;
}
case 3: {
    unsigned int temp;
```

```
            printf("Enter Type of Service (0-255): ");
            scanf("%u", &temp);
            if (temp > 255)
            {
                printf("Error: Type of Service must be between 0 and 255. Value not set.\n");
            }
            else
            {
                packet.type_of_service = temp;
                printf("Type of Service set successfully.\n");
            }
            break;
        }
        case 4: {
            unsigned int temp;
            printf("Enter Total Packet Length (0-65535): ");
            scanf("%u", &temp);
            if (temp > 65535)
            {
                printf("Error: Total Packet Length must be between 0 and 65535. Value not
set.\n");
            }
            else
            {
                packet.total_length = temp;
                printf("Total Packet Length set successfully.\n");
            }
            break;
```

```
        }
        case 5:
          displayPacketHeader(packet);
          break;

        case 6:
          printf("Exiting program.\n");
          a = 1;
          break;

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

    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:

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 : 3;
   unsigned int hours_per_day : 4;
};

void displayTotalHours(struct WorkHours work)
{
   int total_hours = work.days_worked * work.hours_per_day;
   printf("\nEmployee Work Hours Details:\n");
   printf("Days Worked: %u\n", work.days_worked);
   printf("Hours Per Day: %u\n", work.hours_per_day);
   printf("Total Hours Worked in the Week: %d\n", total_hours);
}

int main()
{
   struct WorkHours work ;
   int choice;
   int a = 0;

   while (a != 1)
   {
      printf("\nMenu:\n");
      printf("1. Set Days Worked (0-7)\n");
      printf("2. Set Hours Per Day (0-15)\n");
```

```
printf("3. Display Total Work Hours for the Week\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice)
{
  case 1: {
    unsigned int temp;
    printf("Enter the number of days worked (0-7): ");
    scanf("%u", &temp);
    if (temp > 7)
    {
      printf("Error: Days worked must be between 0 and 7. Value not set.\n");
    }
    else
    {
      work.days_worked = temp;
      printf("Days worked set successfully.\n");
    }
    break;
  }
  case 2: {
    unsigned int temp;
    printf("Enter average hours worked per day (0-15): ");
    scanf("%u", &temp);
    if (temp > 15)
    {
```

```
            printf("Error: Hours per day must be between 0 and 15. Value not set.\n");
        }
        else
        {
            work.hours_per_day = temp;
            printf("Hours per day set successfully.\n");
        }
        break;
      }
      case 3:
        displayTotalHours(work);
        break;

      case 4:
        printf("Exiting program.\n");
        a = 1;
        break;

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

  return 0;
}
```