Problem Statement:1 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:
    1. id (integer): A unique identifier for the employee.
    2. name (character array of size 50): The employee's name.
    3. 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:

    1. **Input Details**: Allow the user to input the details of each employee (ID, name, and salary).
    2. **Display Details**: Display the details of all employees.
    3. **Search by ID**: Allow the user to search for an employee by their ID and display their details.
    4. **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 inputDetails(struct Employee *employees, int n);
void displayDetails(struct Employee *employees, int n);
void searchById(struct Employee *employees, int n, int searchId);
int isUniqueID(struct Employee *employees, int count, int id);

int main() {
```

```
struct Employee *employees = NULL;
int n = 0, choice, searchId;

do {
   printf("\nMenu:\n");
   printf("1. Input Employee Details\n");
   printf("2. Display Employee Details\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: ");
         scanf("%d", &n);

         if (n <= 0) {
            printf("Number of employees must be a positive integer.\n");
            n = 0;
            break;
         }

         employees = (struct Employee *)malloc(n * sizeof(struct Employee));
         if (employees == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
         }

         inputDetails(employees, n);
         break;

      case 2:
         if (employees == NULL) {
            printf("No employee data available. Please input employee details first.\n");
         } else {
            displayDetails(employees, n);
         }
         break;

      case 3:
         if (employees == NULL) {
            printf("No employee data available. Please input employee details first.\n");
         } else {
            printf("Enter ID to search for: ");
            scanf("%d", &searchId);
            searchById(employees, n, searchId);
         }
         break;

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

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

```
    } while (choice != 4);

    if (employees != NULL) {
        free(employees);
    }

    return 0;
}

void inputDetails(struct Employee *employees, int n) {
    for (int i = 0; i < n; i++) {
        int id;
        printf("Enter details of employee %d:\n", i + 1);
        do {
            printf("ID: ");
            scanf("%d", &id);
            if (!isUniqueID(employees, i, id)) {
                printf("Error: ID must be unique. Please enter a different ID.\n");
            }
        } while (!isUniqueID(employees, i, id));

        employees[i].id = id;

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

int isUniqueID(struct Employee *employees, int count, int id) {
    for (int i = 0; i < count; i++) {
        if (employees[i].id == id) {
            return 0; // ID is not unique
        }
    }
    return 1; // ID is unique
}

void displayDetails(struct Employee *employees, int n) {
    printf("\nEmployee Details:\n");
    for (int i = 0; i < n; i++) {
        printf("ID: %d, Name: %s, Salary: %.2f\n", employees[i].id, employees[i].name,
employees[i].salary);
    }
}

void searchById(struct Employee *employees, int n, int searchId) {
    for (int i = 0; i < n; i++) {
        if (employees[i].id == searchId) {
            printf("\nSearch Result:\n");
            printf("ID: %d, Name: %s, Salary: %.2f\n", employees[i].id, employees[i].name,
employees[i].salary);
            return;
        }
    }
    printf("\nSearch Result: Employee with ID %d not found.\n", searchId);
```

}

Problem 2: 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:
    1. id (integer): The book's unique identifier.
    2. title (character array of size 100): The book's title.
    3. 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:

    1. **Input Details**: Input details for each book (ID, title, and price).
    2. **Display Details**: Display the details of all books.
    3. **Find Cheapest Book**: Identify and display the details of the cheapest book.
    4. **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>



typedef struct {

    int id;

    char title[100];

    float price;

} Book;



void input_details(Book *books, int n);
```

```
void display_details(Book *books, int n);
void find_cheapest_book(Book *books, int n);
void update_price(Book *books, int n);
int is_unique_id(Book *books, int count, int id);

int main() {
    Book *books = NULL;
    int n = 0, choice;

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

    if (n <= 0) {
        printf("Invalid number of books. Exiting program.\n");
        return 1;
    }

    books = (Book *)malloc(n * sizeof(Book));
    if (books == NULL) {
        printf("Memory allocation failed. Exiting program.\n");
        return 1;
    }
```

```
do {
    printf("\nBook Inventory System:\n");
    printf("1. Input Book Details\n");
    printf("2. Display Book Details\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:
            input_details(books, n);
            break;
        case 2:
            display_details(books, n);
            break;
        case 3:
            find_cheapest_book(books, n);
            break;
        case 4:
            update_price(books, n);
            break;
        case 5:
            free(books);
```

```
            printf("Exiting program. Memory freed successfully.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 5);

return 0;
}



void input_details(Book *books, int n) {
    printf("\nEnter details for %d books:\n", n);
    for (int i = 0; i < n; i++) {
        int unique = 0;
        do {
            printf("\nBook %d:\n", i + 1);
            printf("ID: ");
            int id;
            scanf("%d", &id);



            if (is_unique_id(books, i, id)) {
                books[i].id = id;
                unique = 1;
```

```
            } else {
                printf("ID %d is already in use. Please enter a unique ID.\n", id);
            }
        } while (!unique);

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


void display_details(Book *books, int n) {
    printf("\nBook 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);
    }
}


void find_cheapest_book(Book *books, int n) {
    int min_index = 0;
    for (int i = 1; i < n; i++) {
```

```
        if (books[i].price < books[min_index].price) {
            min_index = i;
        }
    }
    printf("\nCheapest Book:\n");
    printf("ID: %d, Title: %s, Price: %.2f\n", books[min_index].id,
books[min_index].title, books[min_index].price);
}

void update_price(Book *books, int n) {
    int search_id, found = 0;
    printf("\nEnter the Book ID to update the price: ");
    scanf("%d", &search_id);

    for (int i = 0; i < n; i++) {
        if (books[i].id == search_id) {
            found = 1;
            printf("Enter the new price for '%s': ", books[i].title);
            scanf("%f", &books[i].price);
            printf("Price updated successfully.\n");
            break;
        }
    }

    if (!found) {
```

```
        printf("Book with ID %d not found.\n", search_id);
    }
}


int is_unique_id(Book *books, int count, int id) {
    for (int i = 0; i < count; i++) {
        if (books[i].id == id) {
            return 0;
        }
    }
    return 1;
}
```

Problem 3: 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:
    1. x (float): The x-coordinate of the point.
    2. 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:
    1. **Input Details**: Input the coordinates of each point.
    2. **Display Points**: Display the coordinates of all points.
    3. **Find Distance**: Calculate the Euclidean distance between two points chosen by the user (by their indices in the array).
    4. **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>


typedef struct {
    float x;
    float y;
} Point;


void input_details(Point *points, int n);
void display_points(Point *points, int n);
float calculate_distance(Point p1, Point p2);
void find_closest_pair(Point *points, int n);

int main() {
    int n, choice;

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

    if (n <= 0) {
        printf("Invalid number of points. Exiting program.\n");
        return 1;
```

```
}



Point *points = (Point *)malloc(n * sizeof(Point));
if (points == NULL) {
    printf("Memory allocation failed. Exiting program.\n");
    return 1;
}

do {
    printf("\n2D Point Array Management System:\n");
    printf("1. Input Point Coordinates\n");
    printf("2. Display Points\n");
    printf("3. Find 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:
            input_details(points, n);
            break;
        case 2:
            display_points(points, n);
```

```
            break;
        case 3: {
            int idx1, idx2;
            printf("Enter the indices (0-based) of the two points: ");
            scanf("%d %d", &idx1, &idx2);
            if (idx1 >= 0 && idx1 < n && idx2 >= 0 && idx2 < n) {
                float distance = calculate_distance(points[idx1], points[idx2]);
                printf("Distance between point %d and point %d: %.2f\n", idx1, idx2,
distance);
            } else {
                printf("Invalid indices.\n");
            }
            break;
        }
        case 4:
            find_closest_pair(points, n);
            break;
        case 5:
            free(points);
            printf("Exiting program. Memory freed successfully.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 5);
```

```
    return 0;
}



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



void display_points(Point *points, int n) {
    printf("\nPoints in the 2D space:\n");
    for (int i = 0; i < n; i++) {
        printf("Point %d: (%.2f, %.2f)\n", i + 1, points[i].x, points[i].y);
    }
}



float calculate_distance(Point p1, Point p2) {
```

```
    return sqrtf((p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.y) * (p2.y - p1.y));
}



void find_closest_pair(Point *points, int n) {
    if (n < 2) {
        printf("At least two points are required to find the closest pair.\n");
        return;
    }

    int closest_idx1 = 0, closest_idx2 = 1;
    float min_distance = calculate_distance(points[0], points[1]);

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            float distance = calculate_distance(points[i], points[j]);
            if (distance < min_distance) {
                min_distance = distance;
                closest_idx1 = i;
                closest_idx2 = j;
            }
        }
    }

    printf("\nClosest pair of points:\n");
```

```
    printf("Point %d: (%.2f, %.2f)\n", closest_idx1 + 1, points[closest_idx1].x,
points[closest_idx1].y);

    printf("Point %d: (%.2f, %.2f)\n", closest_idx2 + 1, points[closest_idx2].x,
points[closest_idx2].y);

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

}
```

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:
    1. car_model (character array of size 50): To store the model name of a car.
    2. bike_cc (integer): To store the engine capacity (in CC) of a bike.
    3. bus_seats (integer): To store the number of seats in a bus.
2. Create a structure VehicleInfo that contains:
    1. type (character): To indicate the type of vehicle (C for car, B for bike, S for bus).
    2. Vehicle (the union defined above): To store the specific details of the vehicle based on its type.
3. Implement the following features:
    1. **Input Details**: Prompt the user to input the type of vehicle and its corresponding details:
        1. For a car: Input the model name.
        2. For a bike: Input the engine capacity.
        3. For a bus: Input the number of seats.
    2. **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 <string.h>


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


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

void inputAndDisplayDetails();
int main() {

   inputAndDisplayDetails();
   return 0;
}

void inputAndDisplayDetails()
{
   struct VehicleInfo v_info;


   while(1){
   printf("Enter vehicle type (C for Car, B for Bike, S for Bus,E for exit): ");
   scanf(" %c", &v_info.type);


   if (v_info.type == 'C' || v_info.type == 'c')
   {
      printf("Enter car model: ");
      getchar();
      scanf("%[^\n]",v_info.vehicle.car_model);
      printf("Vehicle Type: Car\nCar Model: %s\n", v_info.vehicle.car_model);
      return;

   }
   else if (v_info.type == 'B' || v_info.type == 'b')
   {
      printf("Enter bike engine capacity (CC): ");
      scanf("%d", &v_info.vehicle.bike_cc);
      printf("Vehicle Type: Bike\nEngine Capacity: %d CC\n", v_info.vehicle.bike_cc);
      return;
   }
   else if (v_info.type == 'S' || v_info.type == 's')
   {
```

```
    printf("Enter number of seats in the bus: ");
    scanf("%d", &v_info.vehicle.bus_seats);
    printf("Vehicle Type: Bus\nNumber of Seats: %d\n", v_info.vehicle.bus_seats);
    return;
  } else
  {
    printf("Invalid vehicle type! Please enter 'C', 'B' or 'S'.\n");
    //return;
  }
  }


}
```

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:
    1. RED: "Stop"
    2. YELLOW: "Ready to move"
    3. GREEN: "Go"

```
#include <stdio.h>



enum TrafficLight{

  RED,

  YELLOW,

  GREEN

};




int main(){

  int input;

  enum TrafficLight currentlight;



  printf("Enter the current traffic light color (0 for RED, 1 for YELLOW, 2 for GREEN): ");
```

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



    if (input < 0 || input > 2) {
        printf("Invalid input. Please enter 0, 1, or 2.\n");
        return 1;
    }


    currentlight = input;
  switch(currentlight){
      case RED:
      printf("Stop\n");
      break;
      case YELLOW:
      printf("Ready to move\n");
      break;
      case GREEN:
      printf("Go\n");
      break;
      default:
      printf("Unknown light\n");
      break;
   }
    return 0;
}
```

Problem 2: Days of the Week

Problem Statement:

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

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

```
#include <stdio.h>



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


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



    printf("Enter a number (1 to 7) representing the day of the week: ");
    scanf("%d", &input);



    if (input < 1 || input > 7) {
```

```
    printf("Invalid input. Please enter a number between 1 and 7.\n");
    return 1;
}



day = input;



switch (day) {
    case MONDAY:
        printf("Monday - Weekday\n");
        break;
    case TUESDAY:
        printf("Tuesday - Weekday\n");
        break;
    case WEDNESDAY:
        printf("Wednesday - Weekday\n");
        break;
    case THURSDAY:
        printf("Thursday - Weekday\n");
        break;
    case FRIDAY:
        printf("Friday - Weekday\n");
        break;
    case SATURDAY:
        printf("Saturday - Weekend\n");
        break;
    case SUNDAY:
```

```
            printf("Sunday - Weekend\n");
            break;
        default:
            printf("Unknown day\n");
            break;
    }

    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:
    1. For CIRCLE: Radius
    2. For RECTANGLE: Length and breadth
    3. For TRIANGLE: Base and height
4. Calculate and display the area of the selected shape.

```
#include <stdio.h>



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

```
int main() {
    int input;
    enum Shape selectedShape;



    printf("Select a shape to calculate its area:\n");
    printf("0 for CIRCLE, 1 for RECTANGLE, 2 for TRIANGLE: ");
    scanf("%d", &input);



    if (input < 0 || input > 2) {
        printf("Invalid input. Please enter 0, 1, or 2.\n");
        return 1;
    }



    selectedShape = input;



    float area = 0.0;
    switch (selectedShape)
    {
        case CIRCLE:
        {
            float radius;
            printf("Enter the radius of the circle: ");
            scanf("%f", &radius);
```

```
    if (radius < 0)
    {
        printf("Invalid radius. It must be positive.\n");
        return 1;
    }
    area = 3.14 * radius * radius;
    printf("The area of the circle is: %.2f\n", area);
    break;
}
case RECTANGLE:
{
    float length, breadth;
    printf("Enter the length of the rectangle: ");
    scanf("%f", &length);
    printf("Enter the breadth of the rectangle: ");
    scanf("%f", &breadth);
    if (length < 0 || breadth < 0) {
        printf("Invalid dimensions. Length and breadth must be positive.\n");
        return 1;
    }
    area = length * breadth;
    printf("The area of the rectangle is: %.2f\n", area);
    break;
}
case TRIANGLE:
{
    float base, height;
    printf("Enter the base of the triangle: ");
```

```
            scanf("%f", &base);
            printf("Enter the height of the triangle: ");
            scanf("%f", &height);
            if (base < 0 || height < 0) {
                printf("Invalid dimensions. Base and height must be positive.\n");
                return 1;
            }
            area = 0.5 * base * height;
            printf("The area of the triangle is: %.2f\n", area);
            break;
        }
        default:
            printf("Unknown shape\n");
            break;
    }

    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:
    1. SUCCESS (0)
    2. FILE_NOT_FOUND (1)
    3. ACCESS_DENIED (2)
    4. OUT_OF_MEMORY (3)
    5. 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 = 0,
    FILE_NOT_FOUND,
    ACCESS_DENIED,
    OUT_OF_MEMORY,
    UNKNOWN_ERROR
};



enum ErrorCode simulateProcess(int scenario) {
    switch (scenario) {
        case 1:
            return FILE_NOT_FOUND;
        case 2:
            return ACCESS_DENIED;
        case 3:
            return OUT_OF_MEMORY;
        case 4:
            return UNKNOWN_ERROR;
        default:
            return SUCCESS;
    }
}


int main()
```

```
{
    int scenario;
    enum ErrorCode error;


    printf("Enter a scenario (0 for SUCCESS, 1 for FILE_NOT_FOUND, 2 for ACCESS_DENIED, 3 for OUT_OF_MEMORY, 4 for UNKNOWN_ERROR): ");
    scanf("%d", &scenario);


    error = simulateProcess(scenario);


    switch (error) {
        case SUCCESS:
            printf("Operation completed 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 occurred.\n");
            break;
```

```
        default:
            printf("Error: Unrecognized error code.\n");
            break;
    }

    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:
    1. ADMIN: "Full access to the system."
    2. EDITOR: "Can edit content but not manage users."
    3. VIEWER: "Can view content only."
    4. GUEST: "Limited access, view public content only."

```
#include <stdio.h>

enum UserRole {
    ADMIN ,
    EDITOR,
    VIEWER,
    GUEST
};

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

    printf("Enter the user role (0 for ADMIN, 1 for EDITOR, 2 for VIEWER, 3 for GUEST): ");
    scanf("%d", &input);

    if (input < 0 || input > 3) {
        printf("Invalid role. Please enter a number between 0 and 3.\n");
        return 1;
    }
```

```
  role = input;


  switch (role)
  {
    case ADMIN:
      printf("Role: ADMIN\nPermissions: Full access to the system.\n");
      break;
    case EDITOR:
      printf("Role: EDITOR\nPermissions: Can edit content but not manage users.\n");
      break;
    case VIEWER:
      printf("Role: VIEWER\nPermissions: Can view content only.\n");
      break;
    case GUEST:
      printf("Role: GUEST\nPermissions: Limited access, view public content only.\n");
      break;
    default:
      printf("Unknown role.\n");
      break;
  }

  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:
    1. day (5 bits): Stores the day of the month (1-31).
    2. month (4 bits): Stores the month (1-12).
    3. year (12 bits): Stores the year (e.g., 2024).
2. Create an array of dates to store 5 different dates.
3. Allow the user to input 5 dates in the format DD MM YYYY and store them in the array.
4. Display the stored dates in the format DD-MM-YYYY.

```
#include<stdio.h>


struct Date{

   unsigned int day :5;

   unsigned int month :4;

   unsigned int year: 12;
```

```
};
int main(){
    struct Date dates[5];
    int day,month,year;
    printf("Enter the 5 dates to be stored:\n");

    for(int i=0;i<5;i++)
    {
        printf("Enter the date %d: ",i+1);
        scanf("%d %d %d",&day,&month,&year);
        if (day < 1 || day > 31 || month < 1 || month > 12)
        {
            printf("\nInvalid date! Please enter a valid date in DD MM YYYY format.\n");
            i--;
        }
        else
        {
            dates[i].day = day;
            dates[i].month = month;
            dates[i].year = year;
        }

    }
```

```
    printf("The dates are:\n");
    for(int i=0;i<5;i++)
    {
        printf("%u-%u-%u\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:
    1. power (1 bit): 1 if the device is ON, 0 if OFF.
    2. connection (1 bit): 1 if the device is connected, 0 if disconnected.
    3. 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:
    1. 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);

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

    while(1)
    {

        displayStatus(device);



        printf("\nChoose an option to update the device status:\n");
        printf("1. Toggle Power\n");
        printf("2. Toggle Connection\n");
        printf("3. Toggle Error\n");
        printf("4. Exit\n");

        printf("Enter your choice (1-4):");
        scanf("%d", &choice);

        switch(choice)
        {
```

```
            case 1:
                device.power = !device.power;
                break;
            case 2:
                device.connection = !device.connection;
                break;
            case 3:
                device.error = !device.error;
                break;
            case 4:
                printf("Exiting the program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}

void displayStatus(struct DeviceStatus device)
{

    printf("Device Status:\n");
    printf("Power: %s\n", device.power ? "ON" : "OFF");
```

```
    printf("Connection: %s\n", device.connection ? "CONNECTED" :
"DISCONNECTED");

    printf("Error: %s\n", device.error ? "ERROR" : "NO ERROR");

}
```

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:
    1. read (1 bit): Permission to read the file.
    2. write (1 bit): Permission to write to the file.
    3. execute (1 bit): Permission to execute the file.
2. Simulate managing file permissions:
    1. Allow the user to set or clear each permission for a file.
    2. 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 fp);


int main() {

    struct FilePermissions file = {0, 0, 0};
```

```
int choice, permission, value;

printf("File Permissions Management:\n");

while (1) {
    printf("\nMenu:\n");
    printf("1. Set Permission\n");
    printf("2. Clear Permission\n");
    printf("3. Display Permissions\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter permission to set (1: Read, 2: Write, 3: Execute): ");
            scanf("%d", &permission);
            if (permission == 1) file.read = 1;
            else if (permission == 2) file.write = 1;
            else if (permission == 3) file.execute = 1;
            else printf("Invalid permission!\n");
            break;
```

```
            case 2:
                printf("Enter permission to clear (1: Read, 2: Write, 3: Execute): ");
                scanf("%d", &permission);
                if (permission == 1) file.read = 0;
                else if (permission == 2) file.write = 0;
                else if (permission == 3) file.execute = 0;
                else printf("Invalid permission!\n");
                break;

            case 3:
                printf("Current Permissions: ");
                displayPermissions(file);
                break;

            case 4:
                printf("Exiting...\n");
                return 0;

            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
}
void displayPermissions(struct FilePermissions fp)
{
```

```
    printf("R:%d W:%d X:%d\n", fp.read, fp.write, fp.execute);
}
```

Problem 4: Network Packet Header

Problem Statement:

Write a C program to represent a network packet header using bit-fields. The program should:

1. Define a structure named PacketHeader with the following bit-fields:
    1. version (4 bits): Protocol version (0-15).
    2. IHL (4 bits): Internet Header Length (0-15).
    3. type_of_service (8 bits): Type of service.
    4. 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;
    int ver_sion,I_H_L,type_service,totl_len;
```

```
printf("Enter the packet header details:\n");


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


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

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

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



printf("\nPacket Header Details:\n");
printf("Version: %u\n", packet.version);
```

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

    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:
    1. days_worked (7 bits): Number of days worked in a week (0-7).
    2. 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;

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

```
        employee.days_worked = days_worked;
    }


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


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


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

    return 0;
}
```