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>
#include <string.h>


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


void input_details(Employee *employees, int n);
int is_unique_id(Employee *employees, int count, int id);
```

```
void display_details(Employee *employees, int n);

void search_by_id(Employee *employees, int n);

void free_memory(Employee *employees);


int main()

{

   int n;

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

   scanf("%d", &n);


   if (n <= 0)

   {

      printf("Invalid number of employees. Exiting program.\n");

      return 1;

   }




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

   if (employees == NULL)

   {

      printf("Memory allocation failed. Exiting program.\n");

      return 1;

   }


   int choice;

   do {

      printf("\nEmployee Management System:\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:
                input_details(employees, n);
                break;
            case 2:
                display_details(employees, n);
                break;
            case 3:
                search_by_id(employees, n);
                break;
            case 4:
                free_memory(employees);
                printf("Exiting program. Memory freed successfully.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}


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


            if (is_unique_id(employees, i, id))
            {
                employees[i].id = id;
                unique = 1;
            }
            else
            {
                printf("ID %d is already in use. Please enter a unique ID.\n", id);
            }
        } while (!unique);

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


void display_details(Employee *employees, int n)
{
```

```c
    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 search_by_id(Employee *employees, int n)
{
    int search_id;
    printf("\nEnter the Employee ID to search: ");
    scanf("%d", &search_id);

    for (int i = 0; i < n; i++)
    {
        if (employees[i].id == search_id)
        {
            printf("Employee Found:\n ID: %d, Name: %s, Salary: %.2f\n", employees[i].id, employees[i].name, employees[i].salary);
            return;
        }
    }
    printf("Employee with ID %d not found.\n", search_id);
}


void free_memory(Employee *employees)
{
    free(employees);
}
```

```
int is_unique_id(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
}
```

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>


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 search_cheapest(Book *books, int n);

void update_price(Book *books, int n);


int main() {

    int n;

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

    scanf(" %d", &n);


    Book *books = (Book *)malloc(n * sizeof(Book));

    if (books == NULL) {

        printf("Memory allocation failed. Exiting program.\n");

        return 1;

    }


    int choice;

    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 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:
                search_cheapest(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)

{

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

   {

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

       printf("ID: ");

       scanf("%d", &books[i].id);

       printf("Title: ");

       scanf(" %[^\n]",books[i].title );

       printf("Price: ");

       scanf("%f", &books[i].price);

   }

}


void display_details(Book *books, int n)

{

   printf("\nBook Details:\n");

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

   {

       printf("Book %d:\n", i + 1);

       printf("ID: %d\n", books[i].id);

       printf("Title: %s\n", books[i].title);

       printf("Price: %.2f\n", books[i].price);

   }

}


void search_cheapest(Book *books, int n)

{

   if (n == 0)

   {

       printf("No books available.\n");
```

```
        return;
    }

    int cheapest_index = 0;
    for (int i = 1; i < n; i++)
    {
        if (books[i].price < books[cheapest_index].price)
        {
            cheapest_index = i;
        }
    }

    printf("\nCheapest Book:\n");
    printf("ID: %d\n", books[cheapest_index].id);
    printf("Title: %s\n", books[cheapest_index].title);
    printf("Price: %.2f\n", books[cheapest_index].price);
}

void update_price(Book *books, int n)
{
    int id, found = 0;
    float new_price;

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

    for (int i = 0; i < n; i++) {
        if (books[i].id == id) {
            printf("Current Price: %.2f\n", books[i].price);
            printf("Enter New Price: ");
            scanf("%f", &new_price);
            books[i].price = new_price;

            printf("Price updated successfully.\n");

            found = 1;

            break;

        }

    }


    if (!found) {

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

    }

}
```

Problem 2: Dynamic Point Array

Problem Statement:

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

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

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


typedef struct {

    float x;
    float y;
} Point;


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

int main() {
    int n;

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


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

    int choice;
    do {
        printf("\nPoint Array Menu:\n");
        printf("1. Input Points\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_points(points, n);
                break;
            case 2:
                display_points(points, n);
                break;
            case 3:
                find_distance(points, 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_points(Point *points, int n)
```

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


void display_points(Point *points, int n)
{
    printf("\nCoordinates of Points:\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 sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}

void find_distance(Point *points, int n)
{
    int index1, index2;

    printf("Enter the indices of two points (1 to %d): ", n);
    scanf("%d %d", &index1, &index2);
    if (index1 < 1 || index1 > n || index2 < 1 || index2 > n)
    {
        printf("Invalid indices. Please try again.\n");
        return;
    }

    float distance = calculate_distance(points[index1 - 1], points[index2 - 1]);
    printf("Distance between Point %d and Point %d: %.2f\n", index1, index2, distance);
}



void find_closest_pair(Point *points, int n)
{
    if (n < 2)
    {
        printf("Not enough points to find a pair.\n");
        return;
    }

    int closest_i = 0, closest_j = 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_i = i;
                closest_j = j;
            }
        }
    }

    printf("Closest Pair of Points: Point %d (%.2f, %.2f) and Point %d (%.2f, %.2f)\n",
           closest_i + 1, points[closest_i].x, points[closest_i].y,
           closest_j + 1, points[closest_j].x, points[closest_j].y);
    printf("Minimum 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:
    - 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 <string.h>


union Vehicle {

  char car_model[50];

  int bike_cc;

  int bus_seats;

};



struct Vehicleinfo {

  char type;

  union Vehicle vehicle;

};
```

```
int main() {
    struct Vehicleinfo var;

    do{
    printf("Enter the vehicle type (C for Car, B for Bike, S for Bus): ");
    scanf(" %c", &var.type);

    switch (var.type) {
        case 'C':
            printf("Enter car model: ");
            scanf(" %[^\n]", var.vehicle.car_model);
            printf("Vehicle Type: Car\n");
            printf("Car Model: %s\n", var.vehicle.car_model);
            break;

        case 'B':
            printf("Enter bike engine capacity (CC): ");
            scanf("%d", &var.vehicle.bike_cc);
            printf("Vehicle Type: Bike\n");
            printf("Engine Capacity: %d CC\n", var.vehicle.bike_cc);
            break;

        case 'S':
            printf("Enter number of seats in the bus: ");
            scanf("%d", &var.vehicle.bus_seats);
            printf("Vehicle Type: Bus\n");
            printf("Number of Seats: %d\n", var.vehicle.bus_seats);
            break;

        case 'E':
```

```
      printf("Exiting !!\n");
      break;

    default:
      printf("Invalid option! Please try again.\n");
      break;
  }
  }while( var.type != 'E');

  return 0;
}
```

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()
{

    enum TrafficLight sig;

    int input;
    printf("Enter the current colour (0 for RED, 1 for YELLOW, 2 for GREEN): ");
    scanf("%d", &input);

    sig = input;


    switch (sig)
    {
        case RED:
            printf("Stop\n");
            break;
        case YELLOW:
            printf("Ready to move\n");
            break;
        case GREEN:
            printf("Go\n");
            break;
        default:
            printf("Invalid input!!\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

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 Shapes
{
    CIRCLE,
    RECTANGLE,
    TRIANGLE,
};

int main()
{
    enum Shapes shape;
    int input;
    printf("Select a shape (0 for CIRCLE, 1 for RECTANGLE, 2 for TRIANGLE):");
    scanf("%d", &input);


    shape = input;

    switch (shape)
    {
        case CIRCLE:
        {
            int r;
            printf("Enter the radius: ");
            scanf("%d", &r);
            printf("Area of the circle is %.2f", (3.14*r*r));
        }
        break;
        case RECTANGLE:
        {
            int l,b;
            printf("Enter the length and breadth: ");
            scanf("%d%d", &l, &b);
             printf("Area of the rectangle is %d", l*b);
        }
        break;
        case TRIANGLE:
```

```
    {
      int l, b;

      printf("Enter the base and height: ");

      scanf("%d%d", &l, &b);

      printf("Area of the triangle is %g", 0.5*l*b);

    }

    break;

    default:

      printf("Invalid input! Please enter a number between 0 and 2.\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>

#include <stdlib.h>



typedef enum {
    SUCCESS,
    FILE_NOT_FOUND,
    ACCESS_DENIED,
    OUT_OF_MEMORY,
    UNKNOWN_ERROR
} ErrorCode;


void printErrorMessage(ErrorCode error)
{
    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: An unknown error occurred.\n");
            break;
        default:
            printf("Error: Invalid error code.\n");
    }
}
```

```
int main()
{
   int scenario;
   ErrorCode error;

   printf("Enter a scenario number (1: File not found, 2: Access denied, 3: Out of memory, others: Unknown error): ");
   scanf("%d", &scenario);

   if(scenario == 0)
   {
      error = SUCCESS;
   }else if (scenario == 1) {
      error = FILE_NOT_FOUND;
   } else if (scenario == 2) {
      error = ACCESS_DENIED;
   } else if (scenario == 3) {
      error = OUT_OF_MEMORY;
   } else {
      error = UNKNOWN_ERROR;
   }


   printErrorMessage(error);

   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>

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

int main()
{
    int roleInput;

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

    UserRole role = roleInput;
    switch (role)
    {
        case ADMIN:
            printf("ADMIN: Full access to the system.\n");
            break;
        case EDITOR:
            printf("EDITOR: Can edit content but not manage 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 role! Please enter a valid user role (0 for ADMIN, 1 for EDITOR, 2 for VIEWER, 3 for GUEST).\n");
    }
}
```

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];
    int day, month, year;

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


        if (day < 1 || day > 31 || month < 1 || month > 12)
        {
            printf("Invalid 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("\nThe stored dates are:\n");
    for (int i = 0; i < 5; i++)
    {
        printf("%d -> %d-%d-%d\n", i + 1, 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);

int main()
{
```

```
struct DeviceStatus device = {0, 0, 0};

unsigned int choice, power, connection, error;
do
{
    printf("\nDevice Status Management Menu:\n");
    printf("1. Turn Power ON/OFF\n");
    printf("2. Set Connection (CONNECTED/DISCONNECTED)\n");
    printf("3. Set Error Status (YES/NO)\n");
    printf("4. Display Current Status\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice)
    {
    case 1:
        printf("Enter Power Status (1 for ON, 0 for OFF): ");
        scanf("%u", &power);
        if (power > 1)
            device.power = 0;
        break;

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

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

        case 4:
            displayStatus(device);
            break;

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

        default:
            printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 5);

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

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:%u W:%u X:%u\n", permissions.read, permissions.write, permissions.execute);
   printf("--------------------------------\n");
}

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

```
unsigned int choice;

do

{

   printf("\nFile Permissions Management Menu:\n");

   printf("1. Set Read Permission\n");

   printf("2. Set Write Permission\n");

   printf("3. Set Execute Permission\n");

   printf("4. Clear Read Permission\n");

   printf("5. Clear Write Permission\n");

   printf("6. Clear Execute Permission\n");

   printf("7. Display Current Permissions\n");

   printf("8. Exit\n");

   printf("Enter your choice: ");

   scanf("%u", &choice);


   switch (choice)

   {

   case 1:

      permissions.read = 1;

      printf("Read permission granted.\n");

      break;

   case 2:

      permissions.write = 1;

      printf("Write permission granted.\n");

      break;

   case 3:

      permissions.execute = 1;

      printf("Execute permission granted.\n");

      break;

   case 4:
```

```
            permissions.read = 0;
            printf("Read permission cleared.\n");
            break;
        case 5:
            permissions.write = 0;
            printf("Write permission cleared.\n");
            break;
        case 6:
            permissions.execute = 0;
            printf("Execute permission cleared.\n");
            break;
        case 7:
            displayPermissions(permissions);
            break;
        case 8:
            printf("Exiting the program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 8);

    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 displayHeader(struct PacketHeader header)
{
   printf("\nPacket Header Details:\n");
   printf("Version          : %u\n", header.version);
   printf("Internet Header Length (IHL): %u\n", header.IHL);
   printf("Type of Service  : %u\n", header.type_of_service);
   printf("Total Length     : %u\n", header.total_length);
   printf("--------------------------------\n");
}

int main()
{
   unsigned int version, IHL, type_of_service, total_length;
   struct PacketHeader header;

   printf("Enter the Packet Header details:\n");
   printf("Enter Version (0-15): ");
   scanf("%u", &version);
   if (version > 15)
       header.version = 0;
   else
        header.version = version;

   printf("Enter Internet Header Length (IHL) (0-15): ");
   scanf("%u", &IHL);
   if (IHL > 15)
       header.IHL = 0;
   else
       header.IHL = IHL;

   printf("Enter Type of Service (0-255): ");
   scanf("%u", &type_of_service);
   if (type_of_service > 255)
       header.type_of_service = 0;
   else
       header.type_of_service = type_of_service;

   printf("Enter Total Length (0-65535): ");
   scanf("%u", &total_length);
   if (total_length > 65535)
       header.total_length = 0;
   else
       header.total_length = total_length;

   displayHeader(header);

   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;
};

unsigned int get_input(const char *prompt, unsigned int max_value)
{
  unsigned int value;
  printf("%s (0-%u): ", prompt, max_value);
  scanf("%u", &value);
  if (value > max_value)
  {
    printf("Invalid input! Setting to 0.\n");
    return 0;
  }
```

```
    return value;
}

int main()
{
    struct WorkHours employee;


    employee.days_worked = get_input("Enter the number of days worked in a week", 7);
    employee.hours_per_day = get_input("Enter the average number of hours worked per day", 15);


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

    return 0;
}
```