Dynamically allocating memory for structures

```
#include<stdio.h>
#include<stdlib.h>
struct course{
   int marks;
   char subject[30];
};

int main(){
   struct course *ptr;
   int noOfRecords;
   printf("Enter the number of records: ");
   scanf("%d",&noOfRecords);
   //Dynamic memory allocation for noOfRecords
   ptr = (struct course *)malloc(noOfRecords * sizeof(struct course));
   for(int i=0;i<noOfRecords;i++){
      printf("Enter subject names and marks: \n");
      scanf("%s %d",(ptr + i)->subject,&(ptr + i)->marks);
   }
   //Display the information
   printf("Displaying information: \n");
   for(int i=0;i<noOfRecords;i++){
      printf("%s\t%d\n",(ptr+i)->subject,(ptr+1)->marks);
   }
   free(ptr);
   return 0;
}
```

Problem Statement: Employee Records Management

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

1. Define a structure named Employee with the following fields:
    - id (integer): A unique identifier for the employee.
    - name (character array of size 50): The employee's name.
    - salary (float): The employee's salary.
2. Dynamically allocate memory for storing information about n employees (where n is input by the user).
3. Implement the following features:
    - **Input Details**: Allow the user to input the details of each employee (ID, name, and salary).
    - **Display Details**: Display the details of all employees.
    - **Search by ID**: Allow the user to search for an employee by their ID and display their details.
    - **Free Memory**: Ensure that all dynamically allocated memory is freed at the end of the program.

Constraints

- n (number of employees) must be a positive integer.
- Employee IDs are unique.

Sample Input/Output

Input:

Enter the number of employees: 3

Enter details of employee 1:

ID: 101

Name: Alice

Salary: 50000

Enter details of employee 2:

ID: 102

Name: Bob

Salary: 60000

Enter details of employee 3:

ID: 103

Name: Charlie

Salary: 55000

Enter ID to search for: 102

Output:

Employee Details:

ID: 101, Name: Alice, Salary: 50000.00

ID: 102, Name: Bob, Salary: 60000.00

ID: 103, Name: Charlie, Salary: 55000.00

Search Result:

ID: 102, Name: Bob, Salary: 60000.00

```
#include<stdio.h>
#include<stdlib.h>
struct Employee{
   int id;
   char name[50];
   float salary;
};
void find(struct Employee *emp,int id1,int n);
int main(){
   struct Employee *emp;
   int n;
   printf("Enter the number of employees: \n");
   scanf("%d",&n);
```

```
    emp = (struct Employee *)malloc(n*sizeof(struct Employee));

    for(int i=0;i<n;i++){
        printf("Enter the details of employee %d: \n",i+1);
        printf("ID: ");
        scanf("%d",&(emp+i)->id);
        printf("Name: ");
        scanf("%s",(emp+i)->name);
        printf("Salary: ");
        scanf("%f",&(emp+i)->salary);
    }

    for(int i=0;i<n;i++){
        printf("ID: %d, Name: %s, Salary: %.2f",(emp+i)->id,(emp+i)->name,(emp+i)->salary);
        printf("\n");
    }

    int id1;
    printf("ID: ");
    scanf("%d",&id1);
    find(emp,id1,n);


    free(emp);

}
void find(struct Employee *emp,int id1,int n){
    int i,found=0;
    for(i=0;i<n;i++){
        if((emp+i)->id==id1){
            printf("Name: %s, salary: %.2f",(emp+i)->name,(emp+i)->salary);
```

```
        found=1;
        break;
      }
      if(!found){
        printf("Not found");
      }
   }

}
```

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>
struct Book{
   int id;
   char title[100];
   float price;
```

```
};
void cheapest(struct Book *b,int n);
void update(struct Book *b,int n,int id1);
int main(){
   struct Book *b;
   int n,id1;
   printf("Enter the number of books: ");
   scanf("%d",&n);
   b = (struct Book *)malloc(n*sizeof(struct Book));
   for(int i=0;i<n;i++){
      printf("Enter the book ID: ");
      scanf("%d",&(b+i)->id);
      printf("Enter the title of the book: ");
      scanf("%s",(b+i)->title);
      printf("Enter the price of book: ");
      scanf("%f",&(b+i)->price);
   }
   for(int i=0;i<n;i++){
      printf("ID: %d, Title: %s, Price: %.2f",(b+i)->id,(b+i)->title,(b+i)->price);
      printf("\n");
   }
   cheapest(b,n);
   printf("\nEnter the id of book for which the price is to be updated: \n");
   scanf("%d",&id1);
   update(b,n,id1);

}
void cheapest(struct Book *b,int n){
   if(n==0){
      printf("No books available");
   }
```

```
    else{
    int cheap=0,i;
    for(i=0;i<n;i++){
        if((b+i)->price<(b+cheap)->price){
            cheap=i;
        }
    }
    printf("Book with cheapest price: \n");
    printf("ID: %d,Title: %s,Price: %.2f",(b+cheap)->id,(b+cheap)->title,(b+cheap)->price);
    }
}
void update(struct Book *b,int n,int id1){
    int found=0;
    for(int i=0;i<n;i++){
        if((b+i)->id==id1){
            printf("\nEnter the updated price: ");
            scanf("%f",&(b+i)->price);
            printf("ID: %d,Title: %s,Price: %.2f",(b+i)->id,(b+i)->title,(b+i)->price);
            found=1;
            break;
        }
        if(!found){
            printf("Book not found");
        }
    }
}
```

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 Distance(struct Point *p,int j,int k);
int main(){
   struct Point *p;
   int n;
   int j,k;
   printf("Enter the no of points: ");
   scanf("%d",&n);
   p = (struct Point *)malloc(n*sizeof(struct Point));
   for(int i=0;i<n;i++){
```

```
    printf("Enter the x and y coordinates of point%d: ",i+1);
    scanf("%f %f",&(p+i)->x,&(p+i)->y);
  }
  for(int i=0;i<n;i++){
    printf("P%d(%.2f %.2f)",i+1,(p+i)->x,(p+i)->y);
    printf("\n");
  }
  printf("Enter the indices of points whose Euclidean distance to be calculated: ");
  scanf("%d %d",&j,&k);
  Distance(p,j,k);

}
void Distance(struct Point *p,int j,int k){
  int d = sqrt(((p+j)->x-(p+i)->x)*(p+j)->x-(p+i)->x)
}
```

UNION

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>
union Vehicle{
   char car_model[50];
   int bike_cc;
   int bus_seats;
};
struct VehicleInfo{
   char type;
   union Vehicle v2;
};
void input(struct VehicleInfo v1);
void display(struct VehicleInfo v1);
int main(){
   struct VehicleInfo v1;
   union Vehicle v2;
   do{
      printf("Enter the type of vehicle(C for car, B for bike and S for bus): ");
      scanf(" %c",&v1.type);
      if (v1.type == 'C' || v1.type == 'B' || v1.type == 'S') {
         break;
      } else {
         printf("Incorrect type of vehicle. Please enter a valid type (C, B, or S).\n");
      }
   }while(1);

   return 0;
}
void input(struct VehicleInfo v1){
   while(1){
      if(v1.type=='C'){
      printf("Enter the model name: ");
```

```
        scanf("%s",v2.car_model);
        }
        else if(v1.type=='B'){
        printf("Enter the engine capacity in CC: ");
        scanf("%d",&v2.bike_cc);
        }
        else if(v1.type=='S'){
        printf("Enter the number of seats: ");
        scanf("%d",&v2.bus_seats);
        }
        else{
        printf("Incorrect type of vehicle");
        }
    }
}
void display(struct VehicleInfo v1){
     if(v1.type=='C'){
        printf("Model name of car: %s",v2.car_model);
    }
    else if(v1.type=='B'){
        printf("Engine capacity of bike: %d",v2.bike_cc);
    }
    else if(v1.type=='S'){
        printf("Number of seats in the bus: %d",v2.bus_seats);
    }
}
```

ENUM

```
#include<stdio.h>

enum math{
```

```
    add=1,
    sub,
    divi
};
int main(){
    enum math var1=divi;
    printf("%d",var1);
}

#include <stdio.h>

enum math{
    add = 1,
    sub
};



int main(){
  enum math var1 = add;
  printf("size of var1 = %d \n",sizeof(var1));
  switch(var1){
      case 1:
      printf("Addition opration\n");
      break;
      case 2:
      printf("Substraction Operation\n");
      break;
      case 3:
      printf("Division Opeartion\n");
      break;
      default:
```

```
    printf("Wrong Option\n");
    break;
  }
   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 color;
  printf("Enter the current light color(0 for RED,1 for YELLOW, 2 for GREEN): ");
  scanf("%d",&color);
  switch(color){
```

```
    case 0:
    printf("Stop\n");
    break;
    case 1:
    printf("Ready to move\n");
    break;
    case 2:
    printf("Go\n");
    break;
    default:
    printf("Wrong Option\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.
    - Weekends: SATURDAY and SUNDAY
    - Weekdays: The rest

```
#include <stdio.h>

enum TrafficLight{
   RED,
   YELLOW,
```

```
    GREEN
};



int main(){
  enum TrafficLight color;
  printf("Enter the current light color(0 for RED,1 for YELLOW, 2 for GREEN): ");
  scanf("%d",&color);
  switch(color){
      case 0:
      printf("Stop\n");
      break;
      case 1:
      printf("Ready to move\n");
      break;
      case 2:
      printf("Go\n");
      break;
      default:
      printf("Wrong Option\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:
    - For CIRCLE: Radius
    - For RECTANGLE: Length and breadth
    - For TRIANGLE: Base and height
4. Calculate and display the area of the selected shape.

```
#include <stdio.h>

enum Shape{
   CIRCLE,
   RECTANGLE,
   TRIANGLE
};



int main(){
  enum Shape sh;
  int r,l,b,ba,h;
  float area,pi=3.14;
  printf("Enter the shape(0 for CIRCLE,1 for RECTANGLE, 2 for TRIANGLE): ");
  scanf("%d",&sh);
  switch(sh){
     case 0:
     printf("Enter the radius of circle: \n");
     scanf("%d",&r);
     area= pi*r*r;
     printf("Area of circle: %.2f",area);
     break;
     case 1:
     printf("Enter the length and breadth of rectangle: \n");
     scanf("%d %d",&l,&b);
```

```
    area = l*b;
    printf("Area of rectangle: %.2f",area);
    break;
    case 2:
    printf("Enter the base and height of triangle: \n");
    scanf("%d %d",&ba,&h);
    area = 0.5*ba*h;
    printf("Area of triangle: %.2f",area);
    break;
    default:
    printf("Wrong Option\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:
    - 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 = 0,
    FILE_NOT_FOUND = 1,
    ACCESS_DENIED = 2,
    OUT_OF_MEMORY = 3,
    UNKNOWN_ERROR = 4
};
enum ErrorCode simulateError(int scenario);
int main() {
    int scenario;
    printf("Enter scenario number (0 to 3): ");
    scanf("%d", &scenario);
    enum ErrorCode errorCode = simulateError(scenario);
    switch (errorCode) {
        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");
            break;
```

```
    }

    return 0;
}
 enum ErrorCode simulateError(int scenario) {
    switch (scenario) {
        case 0:
            return SUCCESS;
        case 1:
            return FILE_NOT_FOUND;
        case 2:
            return ACCESS_DENIED;
        case 3:
            return OUT_OF_MEMORY;
        default:
            return UNKNOWN_ERROR;
    }
}
```

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."

```c
#include <stdio.h>
enum UserRole{
   ADMIN = 0,
   EDITOR = 1,
   VIEWER = 2,
   GUEST = 3
};
enum UserRole permissions(int role);
int main() {
   int role;
   while(1){
      printf("Enter the user role (0 to 3): ");
      scanf("%d", &role);
      if (role == -1) {
         printf("Exiting the program.\n");
         break;
      }
   enum UserRole userrole = permissions(role);
   switch (userrole) {
      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("Unknown user.\n");
            break;
    }
    }

    return 0;
}
 enum UserRole permissions(int role) {
    switch (role) {
        case 0:
            return ADMIN;
        case 1:
            return EDITOR;
        case 2:
            return VIEWER;
        case 3:
            return GUEST;
        }
}
```

Bit fields

```
#include<stdio.h>
struct date{
    int day : 5; //to represent values from 1 to 31
    int month : 4; //to represent values from 1 to 12
    int year;
};
int main(){
    printf("Size of date is %d\n",sizeof(struct date));
    struct date d1 = {25,11,2024};
```

```
    printf("Date is %d-%d-%d",d1.day,d1.month,d1.year);
    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;    // to represent values from 1 to 31
  unsigned int month : 4;  // to represent values from 1 to 12
  unsigned int year : 12;
};

int main() {
    struct Date dates[5];
    int tempDay, tempMonth, tempYear;

    printf("Enter 5 dates in the format DD MM YYYY: ");
    for(int i = 0; i < 5; i++) {
        scanf("%d %d %d", &tempDay, &tempMonth, &tempYear);
```

```
        dates[i].day = tempDay;
        dates[i].month = tempMonth;
        dates[i].year = tempYear;
    }

    for(int i = 0; i < 5; i++) {
        printf("%d-%d-%d\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;        // 1 bit for power (ON/OFF)
    unsigned int connection : 1;    // 1 bit for connection (CONNECTED/DISCONNECTED)
    unsigned int error : 1;        // 1 bit for error (NO/YES)
};
```

```
void displayStatus(struct DeviceStatus status) {
    if (status.power == 1) {
        printf("Power: ON\n");
    } else {
        printf("Power: OFF\n");
    }

    if (status.connection == 1) {
        printf("Connection: CONNECTED\n");
    } else {
        printf("Connection: DISCONNECTED\n");
    }

    if (status.error == 1) {
        printf("Error: YES\n");
    } else {
        printf("Error: NO\n");
    }
}

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

    while (1) {
        printf("1. Set/Reset Power\n");
        printf("2. Set/Reset Connection\n");
        printf("3. Set/Reset Error\n");
        printf("4. Display Current Status\n");
        printf("5. Exit\n");
```

```
printf("Enter your choice (1-5): ");
scanf("%d", &choice);

switch(choice) {
    case 1:
        printf("Enter 1 to turn ON or 0 to turn OFF the power: ");
        scanf("%d", &value);
        device.power = value;
        displayStatus(device);
        break;
    case 2:
        printf("Enter 1 to connect or 0 to disconnect: ");
        scanf("%d", &value);
        device.connection = value;
        displayStatus(device);
        break;
    case 3:
        printf("Enter 1 for error or 0 for no error: ");
        scanf("%d", &value);
        device.error = value;
        displayStatus(device);
        break;
    case 4:
        displayStatus(device);
        break;
    case 5:
        printf("Exiting...\n");
        return 0;
    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;     // 1 bit for read permission (R)
    unsigned int write : 1;    // 1 bit for write permission (W)
    unsigned int execute : 1;  // 1 bit for execute permission (X)
};
void displayPermissions(struct FilePermissions permissions) {
    printf("R:%d W:%d X:%d\n", permissions.read, permissions.write, permissions.execute);
}

int main() {
    struct FilePermissions file = {0, 0, 0};
    int choice, value;

    while (1) {
```

```
printf("1. Set/Unset Read Permission\n");
printf("2. Set/Unset Write Permission\n");
printf("3. Set/Unset Execute Permission\n");
printf("4. Exit\n");

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

switch(choice) {
   case 1:
      printf("Enter 1 to grant read permission or 0 to deny it: ");
      scanf("%d", &value);
      file.read = value;
      break;
   case 2:
      printf("Enter 1 to grant write permission or 0 to deny it: ");
      scanf("%d", &value);
      file.write = value;
      break;
   case 3:
      printf("Enter 1 to grant execute permission or 0 to deny it: ");
      scanf("%d", &value);
      file.execute = value;
      break;
   case 4:
      printf("Exiting...\n");
      return 0;
   default:
      printf("Invalid choice, please try again.\n");
}
displayPermissions(file);
```

```
    }

    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>

// Define the PacketHeader structure with bit-fields
struct PacketHeader {
    unsigned int version : 4;          // 4 bits for version (0-15)
    unsigned int IHL : 4;              // 4 bits for Internet Header Length (0-15)
    unsigned int type_of_service : 8;  // 8 bits for Type of Service (0-255)
    unsigned int total_length : 16;    // 16 bits for Total Length (0-65535)
};

// Function to display the packet header details in a structured format
void displayPacketHeader(struct PacketHeader packet) {
    printf("\nPacket Header Details:\n");
    printf("Version: %u\n", packet.version);
    printf("Internet Header Length (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; // Declare a PacketHeader structure

    // Input values for each field
    printf("Enter the protocol version (0-15): ");
    scanf("%u", &packet.version);
    if (packet.version > 15) {
        printf("Invalid version! It should be between 0 and 15.\n");
        return 1; // Exit the program if the version is out of bounds
    }

    printf("Enter the Internet Header Length (0-15): ");
    scanf("%u", &packet.IHL);
    if (packet.IHL > 15) {
        printf("Invalid IHL! It should be between 0 and 15.\n");
        return 1; // Exit the program if IHL is out of bounds
    }

    printf("Enter the Type of Service (0-255): ");
    scanf("%u", &packet.type_of_service);
    if (packet.type_of_service > 255) {
        printf("Invalid Type of Service! It should be between 0 and 255.\n");
        return 1; // Exit the program if Type of Service is out of bounds
    }

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

```
    if (packet.total_length > 65535) {
        printf("Invalid Total Length! It should be between 0 and 65535.\n");
        return 1; // Exit the program if Total Length is out of bounds
    }

    // Display the packet header details
    displayPacketHeader(packet);

    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 : 7;    // 7 bits for number of days worked (0-7)
    unsigned int hours_per_day : 4;  // 4 bits for average hours per day (0-15)
};
int calculateTotalHours(struct WorkHours employee) {
    return employee.days_worked * employee.hours_per_day;
}

int main() {
```

```
    struct WorkHours employee;
    unsigned int temp_days, temp_hours;  // Temporary variables to store input
    printf("Enter the number of days worked in the week (0-7): ");
    scanf("%u", &temp_days);
    if (temp_days > 7) {
        printf("Invalid input! Number of days worked should be between 0 and 7.\n");
        return 1;
    }
    employee.days_worked = temp_days;

    printf("Enter the average number of hours worked per day (0-15): ");
    scanf("%u", &temp_hours);  // Use a temporary variable for input
    if (temp_hours > 15) {
        printf("Invalid input! Hours per day should be between 0 and 15.\n");
        return 1;
    }
    employee.hours_per_day = temp_hours;
    int total_hours = calculateTotalHours(employee);
    printf("\nTotal hours worked in the week: %d hours\n", total_hours);

    return 0;
}
```