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

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

Sample Input/Output
Input:
*Enter the number of employees: 3*

*Enter details of employee 1:*
*ID: 101*
*Name: Alice*
*Salary: 50000*

*Enter details of employee 2:*
*ID: 102*
*Name: Bob*
*Salary: 60000*

*Enter details of employee 3:*
*ID: 103*
*Name: Charlie*
*Salary: 55000*

*Enter ID to search for: 102*
Output:
*Employee Details:*
*ID: 101, Name: Alice, Salary: 50000.00*
*ID: 102, Name: Bob, Salary: 60000.00*
*ID: 103, Name: Charlie, Salary: 55000.00*

*Search Result:*
*ID: 102, Name: Bob, Salary: 60000.00*

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

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

int main()
{
    int n, idd, found = 0;
    struct employee *ptr;
    printf("Enter the number of employees: ");
    scanf("%d", &n);
    ptr = (struct employee *)malloc(n * sizeof(struct employee));
    for (int i = 0; i < n; i++)
    {
        printf("\nEnter the details of employee %d:\n", i + 1);
        while (1)
        {
            printf("id: ");
            scanf("%d", &(ptr + i)->id);
            int dup= 0;
            for (int j = 0; j < i; j++)
            {
                if ((ptr + i)->id == (ptr + j)->id)
                {
                    printf("ID already exists.\n");
                    dup = 1;
                    break;
                }
            }
            if (!dup)
                break;
        }
        printf("Name: ");
        gets((ptr + i)->name);
        printf("Salary: ");
        scanf("%f", &(ptr + i)->salary);
    }
    printf("\nEmployee Details:\n");
    for (int i = 0; i < n; i++)
    {
        printf("ID: %d\n", (ptr + i)->id);
        printf("Name: %s\n", (ptr + i)->name);
        printf("Salary: %.2f\n\n", (ptr + i)->salary);
    }
    printf("Enter the employee ID to search: ");
    scanf("%d", &idd);

    for (int i = 0; i < n; i++)
    {
        if (idd == (ptr + i)->id)
        {
            printf("\nEmployee Found:\n");
            printf("id: %d\n", (ptr + i)->id);
            printf("Name: %s\n", (ptr + i)->name);
            printf("Salary: %.2f\n", (ptr + i)->salary);
            found = 1;
            break;
        }
    }
    if (!found)
    {
        printf("Employee not found.\n");
    }
    free(ptr);
    return 0;
}
```

**Problem 1: Book Inventory System**

**Problem Statement:**

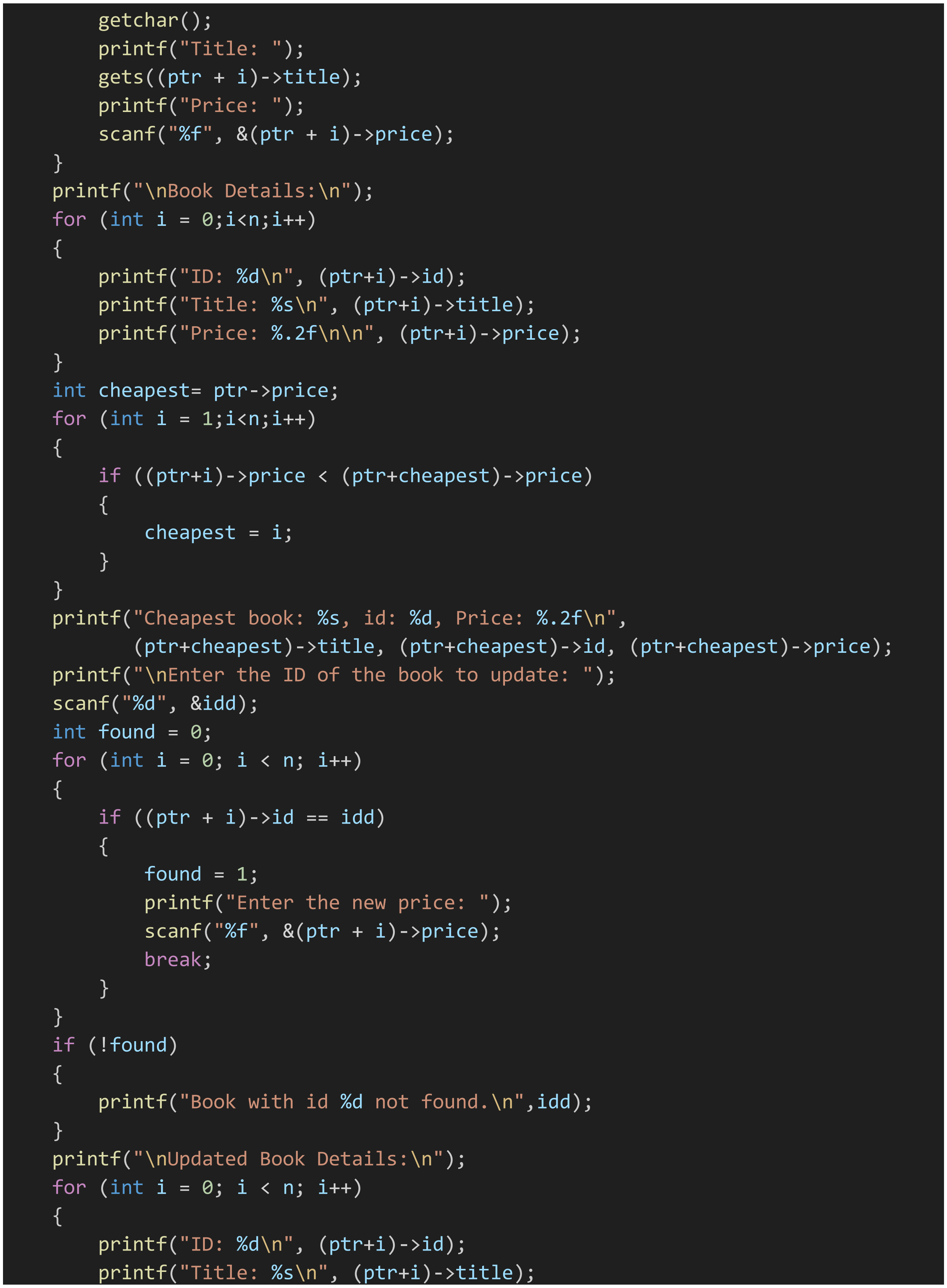
Write a C program to manage a book inventory system using dynamic memory allocation. The program should:

1. Define a structure named Book with the following fields:
    - id (integer): The book's unique identifier.
    - title (character array of size 100): The book's title.

    - price (float): The price of the book.
2. Dynamically allocate memory for n books (where n is input by the user).
3. Implement the following features:
    - **Input Details**: Input details for each book (ID, title, and price).
    - **Display Details**: Display the details of all books.
    - **Find Cheapest Book**: Identify and display the details of the cheapest book.
    - **Update Price**: Allow the user to update the price of a specific book by entering its ID.

```c
#include <stdio.h>

#include <stdlib.h>
#include <string.h>
struct book
{
    int id;
    char title[100];
    float price;
};
int main()
{
    int n, idd;
    struct book *ptr;
    printf("Enter the number of books: ");
    scanf("%d", &n);
    ptr = (struct book *)malloc(n *sizeof(struct book));
    for (int i = 0; i < n; i++)
    {
        printf("\nEnter the details of book %d:\n", i + 1);
        while (1)
        {
            printf("id: ");
            scanf("%d", &(ptr+i)->id);
            int dup = 0;
            for (int j=0;j<i;j++)
            {
                if ((ptr+i)->id==(ptr+j)->id)
                {
                    printf("id already exists. Please enter a unique id.\n");
                    dup = 1;
                    break;
                }
            }
            if (!dup)
                break;
        }
```



```c
        getchar();
        printf("Title: ");
        gets((ptr + i)->title);
        printf("Price: ");
        scanf("%f", &(ptr + i)->price);
    }
    printf("\nBook Details:\n");
    for (int i = 0;i<n;i++)
    {
        printf("ID: %d\n", (ptr+i)->id);
        printf("Title: %s\n", (ptr+i)->title);
        printf("Price: %.2f\n\n", (ptr+i)->price);
    }
    int cheapest= ptr->price;
    for (int i = 1;i<n;i++)
    {
        if ((ptr+i)->price < (ptr+cheapest)->price)
        {
            cheapest = i;
        }
    }
    printf("Cheapest book: %s, id: %d, Price: %.2f\n",
           (ptr+cheapest)->title, (ptr+cheapest)->id, (ptr+cheapest)->price);
    printf("\nEnter the ID of the book to update: ");
    scanf("%d", &idd);
    int found = 0;
    for (int i = 0; i < n; i++)
    {
        if ((ptr + i)->id == idd)
        {
            found = 1;
            printf("Enter the new price: ");
            scanf("%f", &(ptr + i)->price);
            break;
        }
    }
    if (!found)
    {
        printf("Book with id %d not found.\n",idd);
    }
    printf("\nUpdated Book Details:\n");
    for (int i = 0; i < n; i++)
    {
        printf("ID: %d\n", (ptr+i)->id);
        printf("Title: %s\n", (ptr+i)->title);
        printf("Price: %.2f\n\n", (ptr+i)->price);
    }
    free(ptr);
    return 0;
}
```

---

**Problem 2: Dynamic Point Array**

**Problem Statement:**

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

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct Point {
    float x;
    float y;
};
float calculateDistance(struct Point p1, struct Point p2) {
    return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));
}
void findClosestPair(struct Point* points, int n) {
    if (n < 2) {
        printf("Not enough points to find the closest pair.\n");
        return;
    }

    float minDistance = calculateDistance(points[0], points[1]);
    int point1 = 0, point2 = 1;

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            float dist = calculateDistance(points[i], points[j]);
            if (dist < minDistance) {
                minDistance = dist;
                point1 = i;
                point2 = j;
            }
        }
    }
    printf("The closest pair of points are:\n");
    printf("Point 1: (%.2f, %.2f)\n", points[point1].x, points[point1].y);
    printf("Point 2: (%.2f, %.2f)\n", points[point2].x, points[point2].y);
    printf("Distance between them: %.2f\n", minDistance);
}
int main() {
    int n;
    printf("Enter the number of points: ");
    scanf("%d", &n);
    struct Point* points = (struct Point*)malloc(n * sizeof(struct Point));
    if (points == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }
    printf("Enter the coordinates of the points (x y):\n");
    for (int i = 0; i < n; i++) {
        printf("Point %d: ", i + 1);
        scanf("%f %f", &points[i].x, &points[i].y);
    }
    printf("\nCoordinates of the points:\n");
    for (int i = 0; i < n; i++) {
        printf("Point %d: (%.2f, %.2f)\n", i + 1, points[i].x, points[i].y);
    }
    int index1, index2;
    printf("\nEnter the indices of two points to calculate the distance (1 to
%d): ", n);
    scanf("%d %d", &index1, &index2);
    if (index1 >= 1 && index1 <= n && index2 >= 1 && index2 <= n) {
        float distance = calculateDistance(points[index1 - 1], points[index2 -
1]);
        printf("Distance between Point %d and Point %d: %.2f\n", index1, index2,
distance);
    } else {
        printf("Invalid indices!\n");
    }
    findClosestPair(points, n);
    free(points);
    return 0;
}
```

---

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:
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.
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.
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.
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:

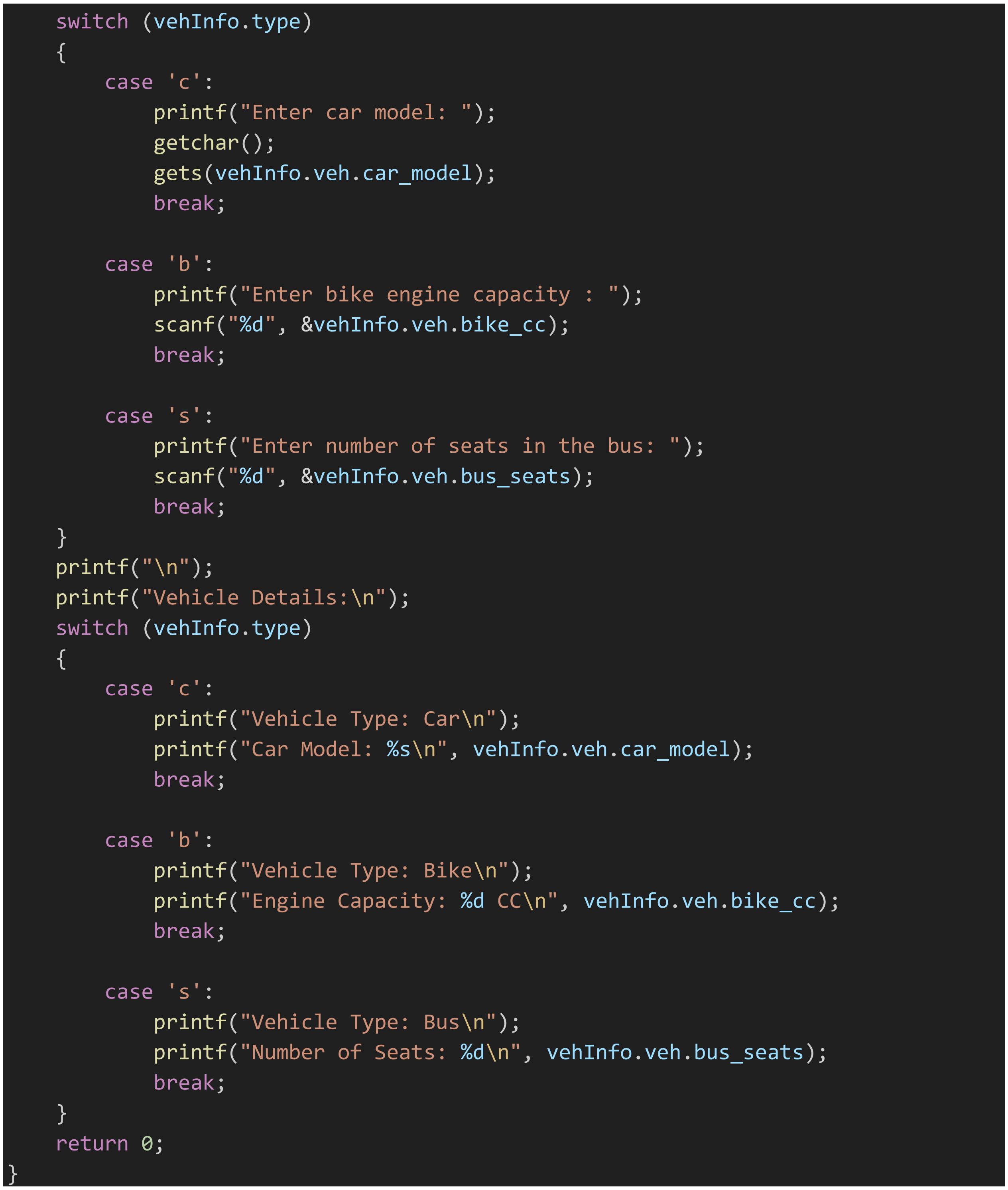
*Vehicle Type: Car*
*Car Model: Toyota Corolla*

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*

```c
#include <stdio.h>
#include <string.h>
union Vehicle
{
    char car_model[50];
    int bike_cc;
    int bus_seats;
};
struct vehicleInfo
{
    char type;
    union Vehicle veh;
};

int main()
{
    struct vehicleInfo vehInfo;
    while(1)
    {
        printf("Enter vehicle type (C for Car, B for Bike, S for Bus): ");
        scanf(" %c", &vehInfo.type);
        if (vehInfo.type == 'c' || vehInfo.type == 'b' || vehInfo.type == 's')
        {
            break;
        }
        else
            printf("Invalid....please try again.\n");
    }
```



```c
    switch (vehInfo.type)
    {
        case 'c':
            printf("Enter car model: ");
            getchar();
            gets(vehInfo.veh.car_model);
            break;

        case 'b':
            printf("Enter bike engine capacity : ");
            scanf("%d", &vehInfo.veh.bike_cc);
            break;

        case 's':
            printf("Enter number of seats in the bus: ");
            scanf("%d", &vehInfo.veh.bus_seats);
            break;
    }
    printf("\n");
    printf("Vehicle Details:\n");
    switch (vehInfo.type)
    {
        case 'c':
            printf("Vehicle Type: Car\n");
            printf("Car Model: %s\n", vehInfo.veh.car_model);
            break;

        case 'b':
            printf("Vehicle Type: Bike\n");
            printf("Engine Capacity: %d CC\n", vehInfo.veh.bike_cc);
            break;

        case 's':
            printf("Vehicle Type: Bus\n");
            printf("Number of Seats: %d\n", vehInfo.veh.bus_seats);
            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"

```c
#include <stdio.h>
enum TrafficLight
{
    red,
    yellow,
    green
};
int main()
{
    enum TrafficLight l;
    int n;
    printf("Enter the value: ");
    scanf("%d", &n);
    l = (enum TrafficLight)n;
    switch (l)
    {
        case red:
            printf("Stop \n");
            break;
        case yellow:
            printf("Ready to move \n");
            break;
        case green:
            printf("Go \n");
            break;
        default:
            printf("Invalid light state.\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

```c
#include<stdio.h>
enum weekday
{
    monday=1, tuesday, wednesday, thursday, friday, saturday, sunday
};
int main()
{
    enum weekday day;
    int num;
    printf("enter a number \n");
    scanf("%d",&num);
    day=(enum weekday)num;
    switch(day)
    {
        case monday:
            printf("weekday");
            break;
        case tuesday:
            printf("weekday");
            break;
        case wednesday:
            printf("weekday");
            break;
        case thursday:
            printf("weekday");
            break;
        case friday:
            printf("weekday");
            break;
        case saturday:
            printf("weekend");
            break;
        case sunday:
            printf("weekend");
            break;
    }
}
```

**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.

```c
#include<stdio.h>
enum shapes
{
    circle, rectangle, triangle
};
int main()
{
    int n;
    float area;
    enum shapes sh;
    printf("enter a value \n");
    scanf("%d",&n);
    sh=(enum shapes)n;
    switch(sh)
    {
        case circle:
            printf("enter the radius of the circle \n");
            int rad;
            scanf("%d",&rad);
            printf("area of the cicrle is %0.2f",3.14*rad*rad);
            break;
        case rectangle:
            printf("enter the length of sides of rectangle \n");
             int l,b;
            scanf("%d %d",&l,&b);
            printf("area of the rectangle is %0.2f",l*b);
            break;
        case triangle:
            printf("enter the length of sides of rectangle \n");
             int base,h;
            scanf("%d %d",&base,&h);
            printf("area of the rectangle is %0.2f",0.5*l*b);
            break;
    }
}
```

---

**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>
```

```c
enum ErrorCode
{
    success ,fileNotFound ,accessDenied,outOfMemory,unknownError
};
void func1();
void func2();
void func3();
void func4();
void func5();
int main()
{
    int n;
    printf("Enter a number : ");
    scanf("%d",&n);
    if(n>5)
        printf("wrong number");
    enum ErrorCode error;
    error=(enum ErrorCode)n;
    switch (error)
    {
        case success:
            func1();
            break;
        case fileNotFound:
            func2();
            break;
        case accessDenied:
            func3();
            break;
        case outOfMemory:
            func4();
            break;
        case unknownError:
            func5();
            break;
        default:
            printf("Unhandled error code.\n");
            break;
    }
    return 0;
}
void func1()
{
    printf("success.\n");
}
```

```c
void func2()
{
    printf("Error: File not found.\n");
}
void func3()
{
     printf("Error: Access denied.\n");
}
void func4()
{
    printf("Error: Out of memory.\n");
}
void func5()
{
printf("Error: Unknown error occurred.\n");
}
```

---

**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, editor, viewer, guest
};
int main()
{
    int n;
    enum userRole user;
    printf("enter a number \n");
    scanf("%d",&n);
    user=(enum userRole)n;
    switch(user)
    {
        case admin:
            printf("full access to the system. \n");
            break;
        case editor:
            printf("can edit content but not manage users. \n");
            break;
        case viewer:
            printf("can view content only \n");
            break;
        case guest:
            printf("Limited access, view public content only \n");
            break;
    }
}
```

**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;
};
```

```c
int main()
{
    int d,m,y;
    struct date dates[5];
    printf("enter the dates \n");
    for(int i=0;i<5;i++)
    {
    scanf("%d %d %d",&d,&m,&y);
    dates[i].day=d;
    dates[i].month=m;
    dates[i].year=y;
    }
    for(int i=0;i<5;i++)
    {
    printf("%d %d %d \n",dates[i].day,dates[i].month,dates[i].year);
    }
}
```

=============================================================================

**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).

```c
#include<stdio.h>
struct deviceStatus
{
    unsigned int power :1;
    unsigned int connection :1;
    unsigned int error :1;
};
```

```c
int main()
{
    int p,c,e;
    printf("power on or off?");
    scanf("%d",&p);
    printf("connection yes or no?");
    scanf("%d",&c);
    printf("error ?");
    scanf("%d",&e);
    (p==1?printf("ON \n") :printf("OFF \n") );
    (c==1?printf("Connected \n") :printf("Disconnected \n") );
    (e==1?printf("Yes") :printf("No") );
    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).

```c
#include<stdio.h>
struct filePermissions
{
    unsigned int read :1;
    unsigned int write :1;
    unsigned int execute :1;
};
int main()
{
    int p,c,e;
    printf("read ?");
    scanf("%d",&p);
    printf("write ?");
    scanf("%d",&c);
    printf("execute ?");
    scanf("%d",&e);
    (p==1?printf("R:1\n") :printf("R:0 \n") );
    (c==1?printf("W:1 \n") :printf("W:0 \n") );
    (e==1?printf("X:1") :printf("X:0") );
    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 typeOfService : 8;
    unsigned int totalLength : 16;
};
int main()
{
    struct packetHeader hp;
    int v,i,t,l;
    printf("enter the version \n");
    scanf("%d",&v);
```

```c
    hp.version=v;
    printf("enter the ihl \n");
    scanf("%d",&i);
    hp.ihl=i;
    printf("enter the type of service \n");
    scanf("%d",&t);
    hp.typeOfService=t;
    printf("enter the total length \n");
    scanf("%d",&l);
    hp.totalLength=l;
    printf("version: %d \nihl : %d\nservice type= %d\ntotal length: %d",hp.version,hp.ihl,hp.typeOfService,hp.totalLength);
    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

```c
#include<stdio.h>
struct workhours
{
    unsigned int daysWorked :7;
    unsigned int hoursPerDay :4;
};
int main()
{
    struct workhours wh;
    int days,hrs,total;
    printf("enter the number of days worked and average hours per day \n");

    scanf("%d %d",&days,&hrs);
    wh.daysWorked=days;
    Wh.hoursPerDay=hrs;
    total= wh.daysWorked* Wh.hoursPerDay;
    printf("total hours worked = %d",total);
}
```