ASSIGNMENT

Structure Padding

```c
#include<stdio.h>
struct test{
    char a;
    char d;
    short c;
    int b;
    int e;
};
int main() {
    printf("Size of struct test: %lu bytes\n", sizeof(struct test));
    return 0;
}
```

Output:

```
Size of struct test: 12 bytes
```

Accessing Structure member through pointer using Dynamic Memory Allocation

```c
#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 Name and Marks :");
        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+i)->marks);
    }
    free(ptr);
    return 0;
}
```

Output
Enter the number of Records : 3
Enter Subject Name and Marks :English 49
Enter Subject Name and Marks :Maths 45
Enter Subject Name and Marks :Physics 42
Displaying Information
English 49
Maths 45
Physics 42

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:
    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

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

struct Employee {
    int ID;
    float salary;
    char name[50];
};
int main() {
    struct Employee *ptr;
    int noOfEmployees;
    int searchID, found = 0;
    printf("Enter the number of employees: ");
    scanf("%d", &noOfEmployees);
    ptr = (struct Employee*)malloc(noOfEmployees * sizeof(struct Employee));

    for (int i = 0; i < noOfEmployees; i++) {
        printf("\nEnter details of employee %d:\n", i + 1);
        printf("ID: ");
        scanf("%d", &(ptr[i].ID));
        printf("Name: ");
        scanf("%s", ptr[i].name);
        printf("Salary: ");
        scanf("%f", &(ptr[i].salary));
    }
    printf("\nEmployee Details:\n");
    for (int i = 0; i < noOfEmployees; i++) {
        printf("ID: %d, Name: %s, Salary: %.2f\n", ptr[i].ID, ptr[i].name, ptr[i].salary);
    }
    printf("\nEnter ID to search for: ");
    scanf("%d", &searchID);
    for (int i = 0; i < noOfEmployees; i++) {
        if (ptr[i].ID == searchID) {
            printf("\nSearch Result:\n");
            printf("ID: %d, Name: %s, Salary: %.2f\n", ptr[i].ID, ptr[i].name, ptr[i].salary);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Employee with ID %d not found.\n", searchID);
    }
    free(ptr);
    return 0;
}
```

Output
Enter details of employee 3:

ID: 3
Name: sneha
Salary: 5000000

Employee Details:
ID: 100, Name: john, Salary: 25000.00
ID: 2, Name: sona, Salary: 30000.00
ID: 3, Name: sneha, Salary: 5000000.00

Enter ID to search for: 3

Search Result:
ID: 3, Name: sneha, Salary: 5000000.00

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:
    - 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 oid;
    float oprice;
    char otitle[100];
};
int isUniqueID(struct Book *books, int count, int id) {
    for (int i = 0; i < count; i++) {
        if (books[i].oid == id) {
            return 0;
        }
    }
    return 1;
}
int main() {
    struct Book *ptr;
    int noofBooks, searchID, found = 0;
    printf("Enter the number of books: ");
    scanf("%d", &noofBooks);
    ptr = (struct Book *)malloc(noofBooks * sizeof(struct Book));
    if (ptr == NULL) {
        printf("Memory allocation failed!\n");
```

```
        return 1;
    }
    for (int i = 0; i < noofBooks; i++) {
        printf("\nEnter details for Book %d:\n", i + 1);
        int id;
        do {
            printf("Book ID (must be unique): ");
            scanf("%d", &id);
            if (!isUniqueID(ptr, i, id)) {
                printf("Error: Book ID already exists. Try again.\n");
            }
        } while (!isUniqueID(ptr, i, id));
        (ptr + i)->oid = id;
        printf("Book Title: ");
        scanf(" %[^\n]%*c", (ptr + i)->otitle);
        printf("Book Price: ");
        scanf("%f", &(ptr + i)->oprice);
    }
    printf("\nDisplaying Book Details:\n");
    printf("ID\tTitle\t\t\tPrice\n");
    printf("-----------------------------------------\n");
    for (int i = 0; i < noofBooks; i++) {
        printf("%d\t%-20s\t%.2f\n", (ptr + i)->oid, (ptr + i)->otitle, (ptr + i)->oprice);
    }
    float min_price = (ptr + 0)->oprice;
    int min_index = 0;
    for (int i = 1; i < noofBooks; i++) {
        if ((ptr + i)->oprice < min_price) {
            min_price = (ptr + i)->oprice;
            min_index = i;
        }
    }
    printf("\nCheapest Book:\n");
    printf("ID: %d, Title: %s, Price: %.2f\n", (ptr + min_index)->oid, (ptr + min_index)-
>otitle, (ptr + min_index)->oprice);
    printf("\nEnter the ID of the book to search: ");
    scanf("%d", &searchID);
    for (int i = 0; i < noofBooks; i++) {
        if (ptr[i].oid == searchID) {
            printf("\nSearched Book Details:\n");
            printf("ID: %d, Title: %s, Price: %.2f\n", ptr[i].oid, ptr[i].otitle,
ptr[i].oprice);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Book Not Found! Check the Book ID.\n");
    }
    free(ptr);
    return 0;
}
```

Output:

Enter the number of books: 2

Enter details for Book 1:

Book ID (must be unique): 1

Book Title: good

Book Price: 300

Enter details for Book 2:

Book ID (must be unique): 1

Error: Book ID already exists. Try again.

Book ID (must be unique): 2

Book Title: happy

Book Price: 250

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:
    - 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 ox;
    float oy;
};
float calculateDistance(struct Point p1, struct Point p2) {
    return sqrt(pow(p1.ox - p2.ox, 2) + pow(p1.oy - p2.oy, 2));
}
int main() {
    struct Point *points;
    int n;
    printf("Enter the number of points: ");
```

```
    scanf("%d", &n);
    points = (struct Point *)malloc(n * sizeof(struct Point));
    if (points == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }
    for (int i = 0; i < n; i++) {
        printf("Enter coordinates for Point %d (x y): ", i + 1);
        scanf("%f %f", &(points[i].ox), &(points[i].oy));
    }
    printf("\nPoints in 2D space:\n");
    for (int i = 0; i < n; i++) {
        printf("Point %d: (%.2f, %.2f)\n", i + 1, points[i].ox, points[i].oy);
    }
    int idx1, idx2;
    printf("Enter the indices of two points to find the distance (1 to %d): ", n);
    scanf("%d %d", &idx1, &idx2);
    if (idx1 < 1 || idx1 > n || idx2 < 1 || idx2 > n) {
        printf("Invalid indices! Please enter values between 1 and %d.\n", n);
    } else {
        float distance = calculateDistance(points[idx1 - 1], points[idx2 - 1]);
        printf("Distance between Point %d and Point %d: %.2f\n", idx1, idx2, distance);
    }
    float minDistance = calculateDistance(points[0], points[1]);
    int p1 = 0, p2 = 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;
                p1 = i;
                p2 = j;
            }
        }
    }
    printf("\nClosest pair of points:\n");
    printf("Point %d: (%.2f, %.2f)\n", p1 + 1, points[p1].ox, points[p1].oy);
    printf("Point %d: (%.2f, %.2f)\n", p2 + 1, points[p2].ox, points[p2].oy);
    printf("Minimum distance: %.2f\n", minDistance);
    free(points);
    return 0;
}
```

Output

Enter the number of points: 2
Enter coordinates for Point 1 (x y): 1 2
Enter coordinates for Point 2 (x y): 2 4

Points in 2D space:
Point 1: (1.00, 2.00)
Point 2: (2.00, 4.00)
Enter the indices of two points to find the distance (1 to 2): 2
4
Invalid indices! Please enter values between 1 and 2.

Closest pair of points:
Point 1: (1.00, 2.00)
Point 2: (2.00, 4.00)
Minimum distance: 2.24

UNION AND STRUCTURE

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

struct course{
    int marks;
    char subject;
};
union course1{
    int marks;
    char subject;
};
int main(){
    struct course strVar;
    union course1 uniVar;
    printf("strVar =%d ,uniVar = %d\n",sizeof(strVar),sizeof(uniVar));
    return 0;
}
```

Output

strVar =8 ,uniVar = 4

```c
#include<stdio.h>
union{
    int a;
    int b;

}var;
int main(){
    var.a=10;
    var.b=20;
    printf("a = %d , b = %d ",var.a,var.b);
    return 0;
}
```

Output
a = 20 , b = 20

UNION AND POINTERS

```c
#include <stdio.h>

union test {
    int a;
    int b;
};
int main() {
    union test var;
    union test *ptr = &var;
    ptr->a = 10;
    printf("001a = %d, b = %d\n", ptr->a, ptr->b);
    ptr->b = 20;
    printf("002a = %d, b = %d\n", ptr->a, ptr->b);
    return 0;
}
```

Output
001a = 10, b = 10
002a = 20, b = 20

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

```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 vehicle;
};
int main() {
    struct VehicleInfo vinfo;
    while (1) {
        printf("Enter Vehicle Type (For Example: C for Car, B for Bike, S for Bus): ");
        scanf(" %c", &vinfo.type);
        if (vinfo.type == 'C' || vinfo.type == 'B' || vinfo.type == 'S') {
            break;
        }
        printf("Invalid input... 'C', 'B', or 'S' should only be valid...\n");
    }
    switch (vinfo.type) {
        case 'C':
            printf("Enter the Car Model: ");
            scanf(" %[^\n]", vinfo.vehicle.car_model);
            printf("\nVehicle Details:\n");
            printf("Vehicle Type: Car\n");
            printf("Car Model: %s\n", vinfo.vehicle.car_model);
            break;
        case 'B':
            printf("Enter Bike Capacity (CC): ");
            scanf("%d", &vinfo.vehicle.bike_cc);
```

```
            printf("\nVehicle Details:\n");
            printf("Vehicle Type: Bike\n");
            printf("Engine Capacity: %d CC\n", vinfo.vehicle.bike_cc);
            break;
        case 'S':
            printf("Enter number of seats in the bus: ");
            scanf("%d", &vinfo.vehicle.bus_seats);
            printf("\nVehicle Details:\n");
            printf("Vehicle Type: Bus\n");
            printf("Number of Seats: %d\n", vinfo.vehicle.bus_seats);
            break;
    }
    return 0;
}
```

Enumerator

```
#include<stdio.h>

enum math{
    add=5,
    sub=20,
    divi
};
int main(){
    enum math var1=add;
    printf("%d",var1);
    return 0;
}
```

Output

21

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 trafficLightSystem{
    red=0,
    yellow=1,
    green=2
};
int main(){
    int user_input;
    printf("Enter Traffic Signal Input (eg. Red=0,Yellow=1,Green=2)");
    scanf("%d",&user_input);
```

```
    enum trafficLightSystem var1=user_input;
    switch(var1){
        case 0:
            printf("Stop !!");
            break;
        case 1:
            printf("Ready to Move!");
            break;
        case 2:
            printf("Go....");
            break;
        default:
            printf("Invalid Input!.!");
    }
    return 0;
}
```

Output:
Enter Traffic Signal Input (eg. Red=0,Yellow=1,Green=2)0
Stop !!

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,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday
};
int main(){
    int user_input;
    printf("Enter a User Input (between 0-6)");
    scanf("%d",&user_input);
    enum Weekday var1=user_input;
    switch(var1){
        case 0:
            printf("It's Monday ..\n Weekday");
            break;
        case 1:
            printf("It's Tuesday ..\n Weekday");
            break;
        case 2:
            printf("It's Wednesday ..\n Weekday");
            break;
        case 3:
```

```
            printf("It's Thursday ..\n Weekday");
            break;
        case 4:
            printf("It's Friday ..\n Weekday");
            break;
        case 5:
            printf("It's Saturday ..\n Weekend");
            break;
        case 6:
            printf("It's Sunday ..\n Weekend");
            break;
        default:
            printf("Invalid Input");

    }
    return 0;
}
```

Output
Enter a User Input (between 0-6)3
It's Thursday ..
Weekday

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 Shapes{
    Circle,
    Rectangle,
    Triangle
};
int main(){
    int user_input;
    printf("Enter a User Input (eg. 0 : Circle, 1 : Rectangle, 2 : Triangle)");
    scanf("%d",&user_input);
    enum Shapes var1=user_input;
    int radius,length,breadth,base,height;
    switch(var1){
        case 0:

            printf("Its a Circle..");
```

```
            printf("Enter Radius :");
            scanf("%d",&radius);
            printf("Area of Circle : %.2f",3.14*radius*radius);
            break;
        case 1:

            printf("Its a Rectangle..");
            printf("Enter Length & Breadth :");
            scanf("%d %d",&length,&breadth);
            printf("Area of Rectangle : %d",length*breadth);
            break;
        case 2:

            printf("Its a Triangle..");
            printf("Enter Base and Height :");
            scanf("%d %d",&base,&height);
            printf("Area of Triangle : %.2f",0.5*base*height);
            break;
        default:
            printf("Please enter valid input");
    }
    return 0;
}
```

Output:
Its a Rectangle..Enter Length & Breadth :2
2
Area of Rectangle : 4

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 = 1,
    ACCESS_DENIED = 2,
    OUT_OF_MEMORY = 3,
    UNKNOWN_ERROR = 4
};

enum ErrorCode simulateFileOperation(int scenario);
void printErrorMessage(enum ErrorCode code);
```

```c
int main() {
    int scenario;

    printf("Enter scenario number (0 for success, 1 for file not found, 2 for access denied, 3
for out of memory, 4 for unknown error): ");
    scanf("%d", &scenario);
    enum ErrorCode code = simulateFileOperation(scenario);
    printErrorMessage(code);
```

```c
    return 0;
}
```

```c
enum ErrorCode simulateFileOperation(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;
        case 4:
            return UNKNOWN_ERROR;
        default:
            return UNKNOWN_ERROR;
    }
}
void printErrorMessage(enum ErrorCode code) {
    switch (code) {
        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: Invalid error code.\n");
            break;
    }
}
```

Output:
Enter scenario number (0 for success, 1 for file not found, 2 for access denied, 3 for out of memory, 4 for unknown error): 3
Error: Out of memory.

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."

```c
#include<stdio.h>

enum UserRole{
    ADMIN,
    EDITOR,
    VIEWER,
    GUEST,
};
int main(){
    int user_input;
    printf("Enter a User Input (between 0-3)");
    scanf("%d",&user_input);
    enum UserRole role=user_input;
    switch(role){
        case 0:
            printf("ADMIN\nFull access to the system");
            break;
        case 1:
            printf("EDITOR\nCan edit content but not manage users");
            break;
        case 2:
            printf("VIEWER\nCan view content only");
            break;
        case 3:
            printf("GUEST\nLimited access, view public content only");
            break;
        default:
            printf("Invalid Input");

    }
    return 0;
}
```

Output:
Enter a User Input (between 0-3)3
GUEST
Limited access, view public content only

BITFIELDS IN C

```c
#include<stdio.h>

struct date{
    int day :5;
    int month :4;
    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:
    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 user_day,user_month,user_year;
    for(int i=0;i<5;i++){
        printf("Enter DD-MM-YYYY",i+1);
        scanf("%d-%d-%d",&user_day,&user_month,&user_year);
        dates[i].day=user_day;
        dates[i].month=user_month;
        dates[i].year=user_year;
    }
    printf("Stored Dates:\n");
    for(int i=0;i<5;i++){
        printf("Date [%d]: %d-%d-%d\n",i+1,dates[i].day,dates[i].month,dates[i].year);
    }
    return 0;
}
```

Output:
Enter DD-MM-YYYY22-09-2008
Enter DD-MM-YYYY23-12-2024
Stored Dates:
Date [1]: 23-2-2001
Date [2]: 18-1-2006
Date [3]: 4-4-2004
Date [4]: 22-9-2008
Date [5]: 23-12-2024

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).

```c
#include <stdio.h>
struct DeviceStatus {
    unsigned int power : 1;
    unsigned int connection : 1;
    unsigned int error : 1;
};
int main() {
    struct DeviceStatus status = {0, 0, 0};
    int input;
    printf("Set Power Status (1 for ON, 0 for OFF): ");
    scanf("%d", &input);
    status.power = input;
    printf("Set Connection Status (1 for CONNECTED, 0 for DISCONNECTED): ");
    scanf("%d", &input);
    status.connection = input;
    printf("Set Error Status (1 for ERROR, 0 for NO ERROR): ");
    scanf("%d", &input);
    status.error = input;
    printf("\nCurrent Device Status:\n");
    printf("Power: ");
    if (status.power) {
        printf("ON\n");
    } else {
        printf("OFF\n");
    }
    printf("Connection: ");
    if (status.connection) {
        printf("CONNECTED\n");
    } else {
        printf("DISCONNECTED\n");
    }
    printf("Error: ");
    if (status.error) {
        printf("ERROR\n");
    } else {
        printf("NO ERROR\n");
    }
    return 0;
}
```

Set Power Status (1 for ON, 0 for OFF): 1
Set Connection Status (1 for CONNECTED, 0 for DISCONNECTED): 0
Set Error Status (1 for ERROR, 0 for NO ERROR): 0

Current Device Status:
Power: ON
Connection: DISCONNECTED
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).

```c
#include <stdio.h>

struct FilePermissions {
    unsigned int read : 1;
    unsigned int write : 1;
    unsigned int execute : 1;
};
int main() {
    struct FilePermissions permissions = {0, 0, 0};
    int input;
    printf("Set Read Permission (1 for granted, 0 for denied): ");
    scanf("%d", &input);
    permissions.read = input;
    printf("Set Write Permission (1 for granted, 0 for denied): ");
    scanf("%d", &input);
    permissions.write = input;
    printf("Set Execute Permission (1 for granted, 0 for denied): ");
    scanf("%d", &input);
    permissions.execute = input;
    printf("\nCurrent File Permissions:\n");
    printf("R:%d W:%d X:%d\n", permissions.read, permissions.write, permissions.execute);
    return 0;
}
```

Set Read Permission (1 for granted, 0 for denied): 0
Set Write Permission (1 for granted, 0 for denied): 1
Set Execute Permission (1 for granted, 0 for denied): 1

Current File Permissions:
R:0 W:1 X:1