1) //Accessing structure member through pointer using dynamic memory allocation

```
#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 SubjectNames 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+i)->marks);
   }

   free(ptr);
   return 0;
}
```

o/p

```
Enter the number of records: 2
Enter SubjectNames and Marks
maths 50
Enter SubjectNames and Marks
phy 50
Displaying Information:
maths   50
phy     50
```

2) 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>

// Define the Employee structure
struct Employee {
    int id;
    char name[50];
    float salary;
};

int main() {
    struct Employee *employees;
    int n, searchId, found = 0;
```

```
// Input the number of employees
printf("Enter the number of employees: ");
scanf("%d", &n);

// Validate input
if (n <= 0) {
    printf("Number of employees must be a positive integer.\n");
    return 1;
}

// Dynamically allocate memory for n employees
employees = (struct Employee *)malloc(n * sizeof(struct Employee));
if (employees == NULL) {
    printf("Memory allocation failed.\n");
    return 1;
}

// Input employee details
for (int i = 0; i < n; i++) {
    printf("\nEnter details of employee %d:\n", i + 1);
    printf("ID: ");
    scanf("%d", &employees[i].id);
    printf("Name: ");
    scanf("%s", employees[i].name);
    printf("Salary: ");
    scanf("%f", &employees[i].salary);
}

// Display employee details
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);
}

// Search for an employee by ID
printf("\nEnter ID to search for: ");
scanf("%d", &searchId);

printf("\nSearch Result:\n");
for (int i = 0; i < n; i++) {
    if (employees[i].id == searchId) {
        printf("ID: %d, Name: %s, Salary: %.2f\n",
               employees[i].id, employees[i].name, employees[i].salary);
        found = 1;
        break;
    }
}
```

```
    if (!found) {
        printf("No employee found with ID %d.\n", searchId);
    }

    // Free allocated memory
    free(employees);

    return 0;
}
```

o/p

```
Enter the number of employees: 2

Enter details of employee 1:
ID: 2
Name: joe
Salary: 10000

Enter details of employee 2:
ID: 4
Name: john
Salary: 23330

Employee Details:
ID: 2, Name: joe, Salary: 10000.00
ID: 4, Name: john, Salary: 23330.00

Enter ID to search for: 4

Search Result:
ID: 4, Name: john, Salary: 23330.00
```

3) 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 details;
};

void inputVehicleDetails(struct VehicleInfo *vehicle) {

    do {
        printf("Enter vehicle type (C for Car, B for Bike, S for Bus): ");
        scanf(" %c", &vehicle->type);
    } while (vehicle->type != 'C' && vehicle->type != 'B' && vehicle->type != 'S');


    if (vehicle->type == 'C') {
        printf("Enter car model: ");
        scanf(" %[^\n]", vehicle->details.car_model);
    } else if (vehicle->type == 'B') {
        printf("Enter bike engine capacity (CC): ");
        scanf("%d", &vehicle->details.bike_cc);
        printf("Enter number of seats in the bus: ");
```

```
        scanf("%d", &vehicle->details.bus_seats);
    }
}

void displayVehicleDetails(const struct VehicleInfo *vehicle) {
    printf("\nVehicle Type: ");
    if (vehicle->type == 'C') {
        printf("Car\n");
        printf("Car Model: %s\n", vehicle->details.car_model);
    } else if (vehicle->type == 'B') {
        printf("Bike\n");
        printf("Engine Capacity: %d CC\n", vehicle->details.bike_cc);
    } else if (vehicle->type == 'S') {
        printf("Bus\n");
        printf("Number of Seats: %d\n", vehicle->details.bus_seats);
    }
}

int main() {
    struct VehicleInfo vehicle;

    inputVehicleDetails(&vehicle);

    displayVehicleDetails(&vehicle);

    return 0;
}
```

4) // wap for enum

```
#include <stdio.h>

enum math{
    add = 3,
    sub,
    divi
};

int main()
{
    enum math var1 = sub;

    printf("%d ",var1);

    return 0;
```

```
}
```

o/p

4

5) // eg for enum

```
#include <stdio.h>

enum math{
   add = 1,
   sub,
   divi
};

int main()
{
   enum math var1 = divi;

   switch(var1){
      case 1:
      printf("addition operation");
      break;

      case 2:
      printf("subtraction operation");
      break;

       case 3:
      printf("division operation");
      break;

      default:
      printf("invalid operation");
      break;
   }


   return 0;
}
```

o/p

division operation

6) // eg 2 for enum

```
#include <stdio.h>

enum math{
   add = 1,
   sub,
   divi
};

int main()
{
   enum math var1 = sub;
   printf("size of var1 = %d\n",sizeof(var1));

   switch(var1){
      case 1:
      printf("addition operation");
      break;

      case 2:
      printf("subtraction operation");
      break;

       case 3:
      printf("division operation");
      break;

      default:
      printf("invalid operation");
      break;
   }


   return 0;
}
```

o/p

```
size of var1 = 4
subtraction operation
```

7) 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 Color {
    RED,
    YELLOW,
    GREEN
};

int main() {
    enum Color currentColor; // Use the enum type directly
    int input;

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

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

    currentColor = (enum Color)input;

    switch (currentColor) {
        case RED:
            printf("Stop\n");
            break;
        case YELLOW:
            printf("Ready to move\n");
            break;
        case GREEN:
            printf("Go\n");
            break;
        default:
            printf("Invalid traffic light state.\n");
    }

    return 0;
```

}

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

int main() {
    enum Weekday day; // Use the enum type
    int input;

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

    if (input < 1 || input > 7) {
        printf("Invalid input! Please enter a number between 1 and 7.\n");
        return 1; // Exit with error code
    }

    day = (enum Weekday)input;
    switch (day) {
        case MONDAY:
            printf("Monday - Weekday\n");
            break;
```

```
        case TUESDAY:
            printf("Tuesday - Weekday\n");
            break;
        case WEDNESDAY:
            printf("Wednesday - Weekday\n");
            break;
        case THURSDAY:
            printf("Thursday - Weekday\n");
            break;
        case FRIDAY:
            printf("Friday - Weekday\n");
            break;
        case SATURDAY:
            printf("Saturday - Weekend\n");
            break;
        case SUNDAY:
            printf("Sunday - Weekend\n");
            break;
        default:
            printf("Invalid day.\n");
    }

    return 0;
}
```

9) 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>
#define PI 3.14
```

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

int main() {
    enum Shape selectedShape;
    int input;
    double area;

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

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

       selectedShape = (enum Shape)input;

        switch (selectedShape) {
        case CIRCLE: {
            double radius;
            printf("Enter the radius of the circle: ");
            scanf("%lf", &radius);
            if (radius < 0) {
                printf("Radius cannot be negative!\n");
                return 1;
            }
            area = PI * radius * radius;
            printf("The area of the circle is: %.2lf\n", area);
            break;
        }
        case RECTANGLE: {
            double length, breadth;
            printf("Enter the length and breadth of the rectangle: ");
            scanf("%lf %lf", &length, &breadth);
            if (length < 0 || breadth < 0) {
                printf("Length and breadth cannot be negative!\n");
                return 1;
            }
            area = length * breadth;
            printf("The area of the rectangle is: %.2lf\n", area);
            break;
```

```
        }
        case TRIANGLE: {
            double base, height;
            printf("Enter the base and height of the triangle: ");
            scanf("%lf %lf", &base, &height);
            if (base < 0 || height < 0) {
                printf("Base and height cannot be negative!\n");
                return 1;
            }
            area = 0.5 * base * height;
            printf("The area of the triangle is: %.2lf\n", area);
            break;
        }
        default:
            printf("Invalid shape.\n");
    }

    return 0;
}
```

10) 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 simulateErrorScenario(int scenario) {
    switch (scenario) {
        case 1:
            return FILE_NOT_FOUND;
        case 2:
            return ACCESS_DENIED;
        case 3:
            return OUT_OF_MEMORY;
        default:
            return UNKNOWN_ERROR;
    }
}

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

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

    error = simulateErrorScenario(scenario);

    switch (error) {
        case SUCCESS:
            printf("Operation completed successfully.\n");
            break;
        case FILE_NOT_FOUND:
            printf("Error: File not found.\n");
            break;
        case ACCESS_DENIED:
            printf("Error: Access denied.\n");
            break;
        case OUT_OF_MEMORY:
            printf("Error: Out of memory.\n");
            break;
        case UNKNOWN_ERROR:
            printf("Error: Unknown error occurred.\n");
            break;
        default:
            printf("Invalid error code.\n");
    }

    return 0;
}
```

11) 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>

enum UserRole {
    ADMIN = 0,
    EDITOR = 1,
    VIEWER = 2,
    GUEST = 3
};

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

    printf("Select a user role:\n");
    printf("0 for ADMIN\n");
    printf("1 for EDITOR\n");
    printf("2 for VIEWER\n");
    printf("3 for GUEST\n");
    printf("Enter your choice: ");
    scanf("%d", &input);

    if (input >= ADMIN && input <= GUEST) {
        role = (enum UserRole)input;
    } else {
        printf("Invalid role! Please enter a number between 0 and 3.\n");
        return 1;
    }

    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("Unknown role.\n");
    }

    return 0;
}
```