1.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.

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

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

void inputDetails(Employee* employees, int n) {
   for (int i = 0; i < n; i++) {
      printf("\nEnter details for Employee %d:\n", i + 1);
      printf("ID: ");
      scanf("%d", &employees[i].id);
      printf("Name: ");
      scanf(" %[^\n]", employees[i].name);
      printf("Salary: ");
      scanf("%f", &employees[i].salary);
   }
}

void displayDetails(Employee* employees, int n) {
   printf("\nEmployee Details:\n");
   for (int i = 0; i < n; i++) {
      printf("Employee %d:\n", i + 1);
      printf("  ID: %d\n", employees[i].id);
      printf("  Name: %s\n", employees[i].name);
      printf("  Salary: %.2f\n", employees[i].salary);
```

```
    }
}

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

int main() {
    int n, searchID;
    Employee* employees;

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

    employees = (Employee*)malloc(n * sizeof(Employee));
    if (employees == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    inputDetails(employees, n);
    displayDetails(employees, n);

    printf("\nEnter the ID of the employee to search: ");
    scanf("%d", &searchID);
    searchByID(employees, n, searchID);

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

2. Problem 1: Book Inventory System
Problem Statement:
Write a C program to manage a book inventory system using dynamic memory allocation. The program should:
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.
Dynamically allocate memory for n books (where n is input by the user).
Implement the following features:
Input Details: Input details for each book (ID, title, and price).
Display Details: Display the details of all books.
Find Cheapest Book: Identify and display the details of the cheapest book.
Update Price: Allow the user to update the price of a specific book by entering its ID.

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

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

int main() {
  int n, bookIdToUpdate, found;
  float newPrice;

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

  struct Book *books = (struct Book *)malloc(n * sizeof(struct Book));
  if (books == NULL) {
    printf("Memory allocation failed!\n");
    return 1;
  }

  for (int i = 0; i < n; i++) {
    printf("\nEnter details of book %d:\n", i + 1);
    printf("ID: ");
    scanf("%d", &books[i].id);
    printf("Title: ");
    getchar();
    fgets(books[i].title, 100, stdin);
    books[i].title[strcspn(books[i].title, "\n")] = '\0';
    printf("Price: ");
    scanf("%f", &books[i].price);
  }

  printf("\nBook Inventory:\n");
  for (int i = 0; i < n; i++) {
    printf("ID: %d, Title: %s, Price: %.2f\n", books[i].id, books[i].title, books[i].price);
```

```
    }

    struct Book cheapest = books[0];
    for (int i = 1; i < n; i++) {
        if (books[i].price < cheapest.price) {
            cheapest = books[i];
        }
    }
    printf("\nCheapest Book:\n");
    printf("ID: %d, Title: %s, Price: %.2f\n", cheapest.id, cheapest.title, cheapest.price);

    printf("\nEnter the ID of the book to update price: ");
    scanf("%d", &bookIdToUpdate);

    found = 0;
    for (int i = 0; i < n; i++) {
        if (books[i].id == bookIdToUpdate) {
            printf("Enter new price for book %d: ", bookIdToUpdate);
            scanf("%f", &newPrice);
            books[i].price = newPrice;
            found = 1;
            break;
        }
    }

    if (found) {
        printf("Price updated successfully.\n");
    } else {
        printf("Book with ID %d not found.\n", bookIdToUpdate);
    }

    printf("\nUpdated Book Inventory:\n");
    for (int i = 0; i < n; i++) {
        printf("ID: %d, Title: %s, Price: %.2f\n", books[i].id, books[i].title, books[i].price);
    }

    free(books);

    return 0;
}
```

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:

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.
Dynamically allocate memory for n points (where n is input by the user).
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.

has context menu

```
#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((p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.y) * (p2.y - p1.y));
}

int main() {
  int n, p1Index, p2Index;
  float minDist, dist;

  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;
  }

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

```
  printf("\nEntered Points:\n");
  for (int i = 0; i < n; i++) {
    printf("Point %d: (%.2f, %.2f)\n", i + 1, points[i].x, points[i].y);
  }

  printf("\nEnter indices of two points to find distance (0-based index): ");
  scanf("%d %d", &p1Index, &p2Index);
  dist = calculateDistance(points[p1Index], points[p2Index]);
  printf("Distance between Point %d and Point %d: %.2f\n", p1Index + 1, p2Index + 1,
dist);

  minDist = calculateDistance(points[0], points[1]);
  int p1 = 0, p2 = 1;
  for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
      dist = calculateDistance(points[i], points[j]);
      if (dist < minDist) {
        minDist = dist;
        p1 = i;
        p2 = j;
      }
    }
  }

  printf("\nClosest pair of points:\n");
  printf("Point %d: (%.2f, %.2f)\n", p1 + 1, points[p1].x, points[p1].y);
  printf("Point %d: (%.2f, %.2f)\n", p2 + 1, points[p2].x, points[p2].y);
  printf("Minimum Distance: %.2f\n", minDist);

  free(points);

  return 0;
}
```

4.Compact Date Storage
Problem Statement:
Write a C program to store and display dates using bit-fields. The program should:
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).
Create an array of dates to store 5 different dates.
Allow the user to input 5 dates in the format DD MM YYYY and store them in the array.
Display the stored dates in the format DD-MM-YYYY.

```c
#include <stdio.h>

typedef struct {
  unsigned int day : 5;
  unsigned int month : 4;
  unsigned int year : 12;
} Date;

int main() {
  Date dates[5];

  for (int i = 0; i < 5; i++) {
    printf("Enter date %d (DD MM YYYY): ", i + 1);
    scanf("%u %u %u", &dates[i].day, &dates[i].month, &dates[i].year);
  }

  printf("\nStored Dates:\n");
  for (int i = 0; i < 5; i++) {
    printf("%02u-%02u-%04u\n", dates[i].day, dates[i].month, dates[i].year);
  }

  return 0;
}
```

5.Status Flags for a Device
Problem Statement:
Write a C program to manage the status of a device using bit-fields. The program should:
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.
Simulate the device status by updating the bit-fields based on user input:
Allow the user to set or reset each status.
Display the current status of the device in a readable format (e.g., Power: ON, Connection: DISCONNECTED, Error: NO).

```c
#include <stdio.h>

typedef struct {
  unsigned int power : 1;
  unsigned int connection : 1;
  unsigned int error : 1;
} DeviceStatus;

int main() {
```

```
  DeviceStatus device = {0, 0, 0};
  int choice;

  while (1) {
    printf("\n1. Set Power\n2. Set Connection\n3. Set Error\n4. Reset Power\n5. Reset
Connection\n6. Reset Error\n7. Display Status\n8. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1: device.power = 1; break;
      case 2: device.connection = 1; break;
      case 3: device.error = 1; break;
      case 4: device.power = 0; break;
      case 5: device.connection = 0; break;
      case 6: device.error = 0; break;
      case 7:
        printf("\nPower: %s\nConnection: %s\nError: %s\n",
          device.power ? "ON" : "OFF",
          device.connection ? "CONNECTED" : "DISCONNECTED",
          device.error ? "YES" : "NO");
        break;
      case 8: return 0;
      default: printf("Invalid choice.\n");
    }
  }
  return 0;
}
```

6. Storage Permissions
Problem Statement:
Write a C program to represent file permissions using bit-fields. The program should:
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.
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>

typedef struct {
  unsigned int read : 1;
```

```
  unsigned int write : 1;
  unsigned int execute : 1;
} FilePermissions;

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

  while (1) {
    printf("\n1. Grant Read\n2. Grant Write\n3. Grant Execute\n4. Revoke Read\n5.
Revoke Write\n6. Revoke Execute\n7. Display Permissions\n8. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1: file.read = 1; break;
      case 2: file.write = 1; break;
      case 3: file.execute = 1; break;
      case 4: file.read = 0; break;
      case 5: file.write = 0; break;
      case 6: file.execute = 0; break;
      case 7:
        printf("\nPermissions: R:%d W:%d X:%d\n", file.read, file.write, file.execute);
        break;
      case 8: return 0;
      default: printf("Invalid choice.\n");
    }
  }
  return 0;
}
```

7. Network Packet Header
Problem Statement:
Write a C program to represent a network packet header using bit-fields. The program should:
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.
Allow the user to input values for each field and store them in the structure.
Display the packet header details in a structured format.

```
#include <stdio.h>

typedef struct {
  unsigned int version : 4;
```

```
    unsigned int IHL : 4;
    unsigned int type_of_service : 8;
    unsigned int total_length : 16;
} PacketHeader;

int main() {
    PacketHeader packet;

    printf("Enter protocol version (0-15): ");
    scanf("%u", &packet.version);
    printf("Enter Internet Header Length (0-15): ");
    scanf("%u", &packet.IHL);
    printf("Enter type of service (0-255): ");
    scanf("%u", &packet.type_of_service);
    printf("Enter total length (0-65535): ");
    scanf("%u", &packet.total_length);

    printf("\nPacket Header Details:\n");
    printf("Version: %u\n", packet.version);
    printf("IHL: %u\n", packet.IHL);
    printf("Type of Service: %u\n", packet.type_of_service);
    printf("Total Length: %u\n", packet.total_length);

    return 0;
}
```

8.Employee Work Hours Tracking
Problem Statement:
Write a C program to track employee work hours using bit-fields. The program should:
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).
Allow the user to input the number of days worked and the average hours per day for an employee.
Calculate and display the total hours worked in the week.

```
#include <stdio.h>

typedef struct {
    unsigned int days_worked : 7;
    unsigned int hours_per_day : 4;
} WorkHours;

int main() {
    WorkHours employee;
    unsigned int total_hours;
```

```
    printf("Enter days worked (0-7): ");
    scanf("%u", &employee.days_worked);
    printf("Enter average hours per day (0-15): ");
    scanf("%u", &employee.hours_per_day);

    total_hours = employee.days_worked * employee.hours_per_day;
    printf("\nTotal hours worked in the week: %u\n", total_hours);

    return 0;
}
```

9. Write a C program to simulate a traffic light system using enum. The program should: Define an enum named TrafficLight with the values RED, YELLOW, and GREEN. Accept the current light color as input from the user (as an integer: 0 for RED, 1 for YELLOW, 2 for GREEN). Display an appropriate message based on the current light: RED: "Stop" YELLOW: "Ready to move" GREEN: "Go"*/

```
#include <stdio.h>

enum TrafficLight {
    RED = 0,
    YELLOW = 1,
    GREEN = 2
};

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

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

    return 0;
```

}

10. Days of the Week Problem Statement: Write a C program that uses an enum to represent the days of the week. The program should: Define an enum named Weekday with values MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, and SUNDAY. Accept a number (1 to 7) from the user representing the day of the week. Print the name of the day and whether it is a weekday or a weekend. Weekends: SATURDAY and SUNDAY Weekdays:

```
#include <stdio.h>

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

int main() {
  int day;
  printf("Enter the day number (1 for MONDAY, 7 for SUNDAY): ");
  scanf("%d", &day);

  switch (day) {
    case MONDAY:
    case TUESDAY:
    case WEDNESDAY:
    case THURSDAY:
    case FRIDAY:
      printf("It's a Weekday\n");
      break;
    case SATURDAY:
    case SUNDAY:
      printf("It's a Weekend\n");
      break;
    default:
      printf("Invalid input\n");
      break;
  }

  return 0;
}
```

11. Write a C program to calculate the area of a shape based on user input using enum. The program should: Define an enum named Shape with values CIRCLE, RECTANGLE, and TRIANGLE. Prompt the user to select a shape (0 for CIRCLE, 1 for RECTANGLE, 2 for TRIANGLE). Based on the selection, input the required dimensions: For CIRCLE: Radius For RECTANGLE: Length and breadth For TRIANGLE: Base and height Calculate and display the area of the selected shape.

```
#include <stdio.h>
#include <math.h>

enum Shape {
  CIRCLE = 0,
  RECTANGLE = 1,
  TRIANGLE = 2
};

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

  if (shape == CIRCLE) {
    float radius;
    printf("Enter the radius of the circle: ");
    scanf("%f", &radius);
    printf("Area of Circle: %.2f\n", M_PI * radius * radius);
  } else if (shape == RECTANGLE) {
    float length, breadth;
    printf("Enter the length and breadth of the rectangle: ");
    scanf("%f %f", &length, &breadth);
    printf("Area of Rectangle: %.2f\n", length * breadth);
  } else if (shape == TRIANGLE) {
    float base, height;
    printf("Enter the base and height of the triangle: ");
    scanf("%f %f", &base, &height);
    printf("Area of Triangle: %.2f\n", 0.5 * base * height);
  } else {
    printf("Invalid selection\n");
  }

  return 0;
}
```

12. Write a C program to simulate error handling using enum. The program should: Define an enum named ErrorCode with values: SUCCESS (0) FILE_NOT_FOUND (1) ACCESS_DENIED (2) OUT_OF_MEMORY (3) UNKNOWN_ERROR (4) Simulate a

function that returns an error code based on a scenario. 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
};

void simulateError() {
  enum ErrorCode error = FILE_NOT_FOUND;
  switch (error) {
    case SUCCESS:
      printf("Operation successful\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\n");
      break;
  }
}

int main() {
  simulateError();
  return 0;
}
```

13. Write a C program to define user roles in a system using enum. The program should: Define an enum named UserRole with values ADMIN, EDITOR, VIEWER, and GUEST. Accept the user role as input (0 for ADMIN, 1 for EDITOR, etc.). 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() {
  int role;
  printf("Enter user role (0 for ADMIN, 1 for EDITOR, 2 for VIEWER, 3 for GUEST): ");
  scanf("%d", &role);

  switch (role) {
    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;
    default:
      printf("Invalid role\n");
      break;
  }

  return 0;
}
```