```c
#include <stdio.h>
#include<stdlib.h>
struct course{
   int marks;
   char sub[30];
};
int main()
{
   struct course *ptr;
   int noOfRecords;
   printf("enter number of records:");
   scanf("%d",&noOfRecords);
   ptr=(struct course*)malloc(noOfRecords*sizeof(struct course));
   for(int i=0;i<noOfRecords;i++){
      printf("enter sub name and marks\n");
      scanf(" %[^\n] %d",(ptr+i)->sub, &(ptr+i)->marks);
   }
   printf("display info\n");
   for(int i=0;i<noOfRecords;i++){
      printf("%s\t %d\t",(ptr+i)->sub, (ptr+i)->marks);
   }
   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[30];
   float salary;
};
void displayDetails(struct Employee *,int);
void addEmployees(struct Employee *,int);
int SearchByID(struct Employee *,int,int );
int main()
{

   int noOfEmployees;
   struct Employee *PtrToEmployee;

   int op;
   while(1){
      printf("enter operation:\n");
      printf("1-add Employee\n2-display employee details\n3-search employee by id\n4-exit\n");
      scanf("%d",&op);
      switch(op){
         case 1:
            printf("enter number of Employees:");
            scanf("%d",&noOfEmployees);
            PtrToEmployee=(struct Employee*)malloc(noOfEmployees*sizeof(struct Employee));
            addEmployees(PtrToEmployee,noOfEmployees);
            break;
         case 2:
            displayDetails(PtrToEmployee,noOfEmployees);
            break;
         case 3:
            int IDtoSearch;
            printf("enter id to search\n");
            scanf("%d",&IDtoSearch);
```

```
            SearchByID(PtrToEmployee,noOfEmployees,IDtoSearch);
            break;
        case 4:
            printf("exiting system\n");
            free(PtrToEmployee);
            return 0;
        default:
            printf("ivalid operation\n");
            break;
      }

   }

}
void addEmployees(struct Employee *PtrToEmployee,int noOfEmployees){


   for(int i=0;i<noOfEmployees;i++){
      printf("enter Employee id name and salary\n");
      scanf("%d %[^\n] %f",&(PtrToEmployee+i)->id, (PtrToEmployee+i)->name, &(PtrToEmployee+i)->salary);


   }
}
void displayDetails(struct Employee *PtrToEmployee,int noOfEmployees){
   for(int i=0;i<noOfEmployees;i++){
      printf("%d\t %s\t %f\n",(PtrToEmployee+i)->id,(PtrToEmployee+i)->name,(PtrToEmployee+i)->salary);
   }

}
int SearchByID(struct Employee *PtrToEmployee,int noOfEmployees,int IDtoSearch){
   int srchflag=0;
   for(int i=0;i<noOfEmployees;i++){
      if(IDtoSearch==(PtrToEmployee+i)->id){
         printf("id found\n");
         printf("%d\t %s\t %f\n",(PtrToEmployee+i)->id,(PtrToEmployee+i)->name,(PtrToEmployee+i)->salary);
```

```
        srchflag=1;
        return 0;
      }

   }
   if(srchflag==0){
      printf("id not found");
      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.

```
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
struct library {
   char *title;
   char *author;
   float  price;
};
void cheapestBook(struct library *,int);
void inputBook(struct library*,int);
int updatePrice(struct library*,char*,int);
```

```
void displayBook(struct library *,int);
int main(){
   int no,option;
   struct library *books;
   int flag = 0;
   while(1){
      printf("Enter your choice\n1-add book\n2-dispaly data\n3-update price\n4-cheapest\n5-exit:\n");
      scanf("%d", &option);
      switch(option){
         case 1:
            printf("enter number of books details to be entered:");
            scanf("%d",&no);
            books=(struct library*)malloc(no*sizeof(struct library));
            inputBook(books,no);
            flag=1;
            break;
         case 2:
            printf("all available books\ntitle|\t\tauthor|\t\tprice|\t\t\n");
            displayBook(books,no);
            break;
         case 3:
            if(flag==0){
               printf("no data enterd.\n");
               return 0;
            }
            printf("enter title of the book\n");
            char nTitle[20];
            scanf(" %[^\n]",nTitle);
            updatePrice(books,nTitle,no);
            break;
         case 4:
            cheapestBook(books,no);
            break;
         case 5:
            free(books);
            free(books->title);
```

```
            free(books->author);
            printf("exiting!!");
            return 0;
          default:
            printf("invalid option");
      }
   }
}
void inputBook(struct library *books,int no){
   printf("enter %d books\n",no);

   for(int i=0;i<no;i++){
      books[i].title=(char*)malloc(20*sizeof(char));
      books[i].author=(char*)malloc(20*sizeof(char));

      printf("title: ");
      scanf(" %[^\n]", books[i].title);

      printf("author: ");
      scanf(" %[^\n]", books[i].author);

      printf("price: ");
      scanf("%f", &books[i].price);
   }

}
void displayBook(struct library *books,int no){
   for(int i=0;i<no;i++){
      printf("%s\t\t%s\t\t %f\t\t\n", books[i].title,books[i].author,(books[i].price));
   }
}
int updatePrice(struct library *books,char *nTitle,int no){
   int i=0;
   int book_title=0;
   while(i<no){
      if(strcmp(nTitle,books[i].title)==0){
```

```
        printf("book found\n");
        printf("enter new price:");
        scanf("%f",&books[i].price);
        book_title=1;
        printf("updated  price\n");
        return i;
      }
      i++;
    }
    if(book_title==0){
      printf("book not found\n");
    }
}
void cheapestBook(struct library *books,int no){
    int temp;
    for(int i=0;i<no-1;i++){
      if(books[i].price>books[i+1].price){
        temp=books[i].price;
        books[i].price=books[i+1].price;
        books[i+1].price=books[i].price;
      }
    }
    printf("cheapest book is\n");
    displayBook(books,1);
}
```

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>



typedef struct {
   float x;
   float y;
} Point;
float distance(Point p1, Point p2) {
   return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}


int main() {
   int n, i, j, p1, p2;
   printf("Enter the number of points: ");
   scanf("%d", &n);
   Point *points = malloc(n * sizeof(Point));
   for (i = 0; i < n; i++) {
      printf("Enter point %d (x y): ", i + 1);
      scanf("%f %f", &points[i].x, &points[i].y);
   }
   printf("\nPoints:\n");
   for (i = 0; i < n; i++) {
      printf("Point %d: (%.2f, %.2f)\n", i + 1, points[i].x, points[i].y);
   }
   printf("\nEnter two points to calculate distance (1-%d): ", n);
   scanf("%d %d", &p1, &p2);
   if (p1 >= 1 && p1 <= n && p2 >= 1 && p2 <= n) {
      printf("Distance: %.2f\n", distance(points[p1 - 1], points[p2 - 1]));
   } else {
      printf("Invalid indices!\n");
   }
```

```
    float minDist = 1e9;
    int closest1 = -1, closest2 = -1;
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            float d = distance(points[i], points[j]);
            if (d < minDist) {
                minDist = d;
                closest1 = i;
                closest2 = j;
            }
        }
    }
    if (closest1 != -1) {
        printf("\nClosest points are Point %d and Point %d with distance %.2f\n",
               closest1 + 1, closest2 + 1, minDist);
    }
    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:

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<stdlib.h>
#include<string.h>
union vehicle{
   char car_model[50];
   int bike_cc;
   int bus_seats;
};
struct VehicleInfo{
```

```
    char type[10];
    union vehicle vehicle1;
};
void enterData(struct VehicleInfo* ,char );
void displayData(struct VehicleInfo*,char op);
int main(){
    char v_type;
    union vehicle v1;
    struct VehicleInfo V_Info;
    while(1){

        printf("enter\nC-car\nB-bike\nS-bus\n");
        scanf(" %c",&v_type);
        switch(v_type){
            case 'C':
                enterData(&V_Info,v_type);
                displayData(&V_Info,v_type);
                break;
            case 'B':
                enterData(&V_Info,v_type);
                displayData(&V_Info,v_type);
                break;
            case 'S':
                enterData(&V_Info,v_type);
                displayData(&V_Info,v_type);
                break;
            default:
                printf("Invalid operation\n");
                break;
        }
    }
    return 0;
}
void enterData(struct VehicleInfo *V_Info,char op){
    if(op=='C'){
        printf("enter car model:");
```

```
    scanf(" %[^\n]",V_Info->vehicle1.car_model);
    strcpy(V_Info->type,"car");
  }
  else if(op=='B'){
    printf("enter bike engine cc:");
    scanf("%d",&V_Info->vehicle1.bike_cc);
    strcpy(V_Info->type,"bike");
  }
  else if(op=='S'){
    printf("enter no of seat :");
    scanf("%d",&V_Info->vehicle1.bus_seats);
    strcpy(V_Info->type,"bus");
  }
}
void displayData(struct VehicleInfo *V_Info,char op){
  if(op=='C'){
    printf("vehicle type:%s\n",V_Info->type);
    printf("car model:%s\n",V_Info->vehicle1.car_model);
  }
  else if(op=='B'){
    printf("vehicle type:%s\n",V_Info->type);
    printf("bike cc:%d\n",V_Info->vehicle1.bike_cc);
  }
  else if(op=='S'){
    printf("vehicle type:%s\n",V_Info->type);
    printf("number of seats:%d\n",V_Info->vehicle1.bus_seats);
  }
}
```

ENUMERATION

```
#include<stdio.h>
enum math{
  add =1,
  sub,
  divi,
};
```

```
int main(){
   enum math var1=divi;
   printf("%d",var1);
   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(){
   int currentState;
   while(1){
   printf("enter 0,1,2:");
   scanf("%d",&currentState);
   switch(currentState){
      case red:
         printf("stop\n");
         break;
      case yellow:
         printf("ready to move\n");
         break;
      case green:
         printf("go\n");
         break;
```

```
    default:
      printf("invalid entry\n");
  }
  }
  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 days {
  monday = 1,
  tuesday,
  wednesday,
  thursday,
  friday,
  saturday,
  sunday,
};
int main() {
  int currentDay;
  while (1) {
    printf("Enter a number (1 for Monday to 7 for Sunday, 0 to exit): ");
    scanf("%d",&currentDay);
    switch (currentDay) {
      case monday:
        printf("Day is Monday and it is a weekday.\n");
        break;
      case tuesday:
        printf("Day is Tuesday and it is a weekday.\n");
        break;
```

```
        case wednesday:
          printf("Day is Wednesday and it is a weekday.\n");
          break;
        case thursday:
          printf("Day is Thursday and it is a weekday.\n");
          break;
        case friday:
          printf("Day is Friday and it is a weekday.\n");
          break;
        case saturday:
          printf("Day is Saturday and it is a weekend.\n");
          break;
        case sunday:
          printf("Day is Sunday and it is a weekend.\n");
          break;
        default:
          printf("Invalid input. Please enter a number between 1 and 7.\n");
      }
    }
    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(){

  int op;
  while(1){
    printf("enter 0,1,2:");
    scanf("%d",&op);
    switch(op){
    case circle:
      int radius;
      printf("radius:");
      scanf("%d",&radius);
      printf("area=%f\n",3.14*radius*radius);
      break;
    case rectangle:
      int len,wid;
      printf("enter length and width");
      scanf("%d %d",&len,&wid);
      printf("area of rectangle is %d\n",len*wid);
      break;
    case triangle:
      int base,height;
      printf("enter the base and height of triangle:");
      scanf("%d %d",&base,&height);
      printf("area of triangle is %f\n",0.5*base*height);
      break;
    default:
      printf("invalid entry\n");
  }

  }
  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
};

int main() {
  int scenario;
  enum ErrorCode errorCode = SUCCESS;
  printf("Enter a scenario number (1 to 4) to simulate an error:\n");
  printf("1 - File not found\n");
  printf("2 - Access denied\n");
  printf("3 - Out of memory\n");
  printf("Any other number - Unknown error\n");

  scanf("%d", &scenario);

  switch (scenario) {
    case 1:
      errorCode = FILE_NOT_FOUND;
      printf("Error: File not found.\n");
      break;
    case 2:
      errorCode = ACCESS_DENIED;
```

```
            printf("Error: Access denied.\n");
            break;
        case 3:
            errorCode = OUT_OF_MEMORY;
            printf("Error: Out of memory.\n");
            break;
        default:
            errorCode = UNKNOWN_ERROR;
            printf("Error: Unknown error occurred.\n");
            break;
    }

    if (errorCode == SUCCESS) {
        printf("Operation completed successfully.\n");
    } else if (errorCode == FILE_NOT_FOUND) {
        printf("Error: File not found.\n");
    } else if (errorCode == ACCESS_DENIED) {
        printf("Error: Access denied.\n");
    } else if (errorCode == OUT_OF_MEMORY) {
        printf("Error: Out of memory.\n");
    } else {
        printf("Error: Unknown error occurred.\n");
    }

    return 0;
}
```

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() {
  int role;
  printf("0 for ADMIN, 1 for EDITOR, 2 for VIEWER, 3 for GUEST: ");
  scanf("%d", &role);



  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("Invalid role! Please enter a number between 0 and 3.\n");
      break;
  }

  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;
  unsigned int month:4;
  unsigned int year:12;
};
int main(){
  struct date arrDate[5];
  unsigned int day,month,year;
  for(int i=0;i<5;i++){
    printf("ente rdate in dd-mm-yy format\n");
    scanf("%u %u %u",&day,&month,&year);
    arrDate[i].day=day;
    arrDate[i].month=month;
    arrDate[i].year=year;
  }
  for(int i=0;i<5;i++){
    printf("%u-%u-%u\n",arrDate[i].day,arrDate[i].month,arrDate[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;
  unsigned int connection:1;
  unsigned int error:1;
};
int main(){
  struct DeviceStatus flag[10];
  printf("enter number of flag registers:");
  int no;
  scanf("%d",&no);
  unsigned int power, connection, error;
  printf("enter the status of registers\n");
  for(int i=0;i<no;i++){
    printf("status of %d register\n",i+1);
    printf("power->connection->error:");
    scanf("%u %u %u",&power,&connection,&error);
    flag[i].power=power;
    flag[i].connection=connection;
    flag[i].error=error;
  }
  for(int i=0;i<no;i++){
    printf("status of register %d",i+1);
    if (flag[i].power==1){
      printf("power=%s\t","ON");
```

```
    }
    else(printf("power=%s\t","OFF"));
    if (flag[i].connection==1){
      printf("connection=%s\t","connected");

    }
    else(printf("connection=%s\t","Disconnected"));
    if (flag[i].error==1){
      printf("error=%s\t","YES");

    }
    else(printf("error=%s\n","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).

```
#include<stdio.h>
struct FilePermissions{
  unsigned int read :1;
  unsigned int write:1;
  unsigned int execute:1;
};
int main(){
  struct FilePermissions permission[10];
  printf("enter number of file permissions:");
  int no;
```

```
  scanf("%d",&no);
  unsigned int read,write,execute;
  printf("enter the status of file permission\n");
  for(int i=0;i<no;i++){
    printf("status of %d file\n",i+1);
    printf("read->write->execute:");
    scanf("%u %u %u",&read,&write,&execute);
    permission[i].read=read;
    permission[i].write=write;
    permission[i].execute=execute;
  }
  for(int i=0;i<no;i++){
    printf("status of %d file\n",i+1);
      printf("R:=%d",permission[i].read);



    printf("W:=%d",permission[i].write);


    printf("X:=%d\n",permission[i].execute);
  }
  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 total_length:16;

};
int main(){
    struct PacketHeader Packet1;
    unsigned int version,IHL,typeOfService,total_length;
    printf("enter the values to each field\n");
        scanf("%u %u %u %u",&version,&IHL,&typeOfService,&total_length);
        Packet1.version=version;
        Packet1.IHL=IHL;
        Packet1.typeOfService=typeOfService;
        Packet1.total_length=total_length;
        printf("packet header details\n");
        printf("version:%d",Packet1.version);
        printf("IHL:%d",Packet1.IHL);
        printf("typeOfService:%d",Packet1.typeOfService);
        printf("total_length:=%d\n",Packet1.total_length);

    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 : 3;
    unsigned int hours_per_day : 4;
```

```
};

int main() {
    struct WorkHours employee;

    unsigned int days, hours;
    printf("Enter the number of days worked (0-7): ");
    scanf("%u", &days);
    if (days > 7) {
        printf("Invalid input. Days worked should be between 0 and 7.\n");
        return 1;
    }
    printf("Enter the average hours worked per day (0-15): ");
    scanf("%u", &hours);
    if (hours > 15) {
        printf("Invalid input. Hours per day should be between 0 and 15.\n");
        return 1;
    }
    employee.days_worked = days;
    employee.hours_per_day = hours;
    unsigned int total_hours = employee.days_worked * employee.hours_per_day;
    printf("\nEmployee Work Hours Summary:\n");
    printf("Days Worked: %u\n", employee.days_worked);
    printf("Average Hours per Day: %u\n", employee.hours_per_day);
    printf("Total Hours Worked in the Week: %u\n", total_hours);

    return 0;
}
```
