DAY_14 ASSIGNMENT

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

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

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

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

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

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

// Enter ID to search for: 102
// Output:

```
// Employee Details:
// ID: 101, Name: Alice, Salary: 50000.00
// ID: 102, Name: Bob, Salary: 60000.00
// ID: 103, Name: Charlie, Salary: 55000.00

// Search Result:
// ID: 102, Name: Bob, Salary: 60000.00
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

void inputEmployeeDetails(struct Employee *emp, int n);
void displayEmployeeDetails(struct Employee *emp, int n);
void searchEmployeeById(struct Employee *emp, int n);
void freeMemory(struct Employee *emp);

int main() {
    int n;

    printf("Enter the number of employees: ");
    scanf("%d", &n);
    struct Employee *employee;
    employee = (struct Employee *)malloc(n * sizeof(struct Employee));
    if (employee == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }
    inputEmployeeDetails(employee, n);
    displayEmployeeDetails(employee, n);
    searchEmployeeById(employee, n);
    freeMemory(employee);

    return 0;
}


void inputEmployeeDetails(struct Employee *emp, int n) {
    for (int i = 0; i < n; i++) {
```

```
        int flag = 0;
        while (flag == 0) {
            flag = 1;
            printf("Enter Employee ID: ");
            scanf("%d", &emp[i].id);
            for (int j = 0; j < i; j++) {
                if (emp[i].id == emp[j].id) {
                    printf("ID already exists. Please enter a unique ID.\n");
                    flag = 0;
                    break;
                }
            }
        }
        printf("Name: ");
        scanf(" %[^\n]", emp[i].name);
        printf("Salary: ");
        scanf("%f", &emp[i].salary);
    }
}


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

void searchEmployeeById(struct Employee *emp, int n) {
    int searchId;
    printf("\nEnter ID to search for: ");
    scanf("%d", &searchId);

    for (int i = 0; i < n; i++) {
        if (emp[i].id == searchId) {
            printf("Search Result:\n");
            printf("ID: %d, Name: %s, Salary: %.2f\n", emp[i].id, emp[i].name, emp[i].salary);
            return;
        }
    }
    printf("Employee with ID %d not found.\n", searchId);
}

void freeMemory(struct Employee *emp) {
```

```
    free(emp);
    printf("\nMemory freed successfully.\n");
}
```

```
// 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;
};

void inputBookDetails(struct Book *books, int n);
void displayBookDetails(struct Book *books, int n);
void findCheapestBook(struct Book *books, int n);
void updateBookPrice(struct Book *books, int n);
void freeMemory(struct Book *books);

int main() {
    int n;
    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;
    }
    inputBookDetails(books, n);
    displayBookDetails(books, n);
    findCheapestBook(books, n);
    updateBookPrice(books, n);
    freeMemory(books);
    return 0;
}
void inputBookDetails(struct Book *books, int n) {
    for (int i = 0; i < n; i++) {
        printf("\nEnter details for book %d:\n", i + 1);
        int flag = 0;
        while (flag == 0) {
            flag = 1;
            printf("ID: ");
            scanf("%d", &books[i].id);
            for (int j = 0; j < i; j++) {
                if (books[i].id == books[j].id) {
                    printf("This ID already exists. Please enter a unique ID.\n");
                    flag = 0;
                    break;
                }
            }
        }
        printf("Title: ");
        scanf("%s",books[i].title);
        printf("Price: ");
        scanf("%f", &books[i].price);
    }
}
void displayBookDetails(struct Book *books, int n) {
    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);
    }
}
void findCheapestBook(struct Book *books, int n) {
    int cheapestIndex = 0;

    for (int i = 1; i < n; i++) {
        if (books[i].price < books[cheapestIndex].price) {
```

```
            cheapestIndex = i;
        }
    }

    printf("\nCheapest Book:\n");
    printf("ID: %d, Title: %s, Price: %.2f\n", books[cheapestIndex].id, books[cheapestIndex].title,
books[cheapestIndex].price);
}
void updateBookPrice(struct Book *books, int n) {
    int idToUpdate;
    printf("\nEnter the ID of the book whose price you want to update: ");
    scanf("%d", &idToUpdate);

    int found = 0;
    for (int i = 0; i < n; i++) {
        if (books[i].id == idToUpdate) {
            printf("Enter the new price for the book '%s': ", books[i].title);
            scanf("%f", &books[i].price);
            printf("Price updated successfully.\n");
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Book with ID %d not found.\n", idToUpdate);
    }
}
void freeMemory(struct Book *books) {
    free(books);
    printf("\nMemory freed successfully.\n");
}
```

```
// 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:
// 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.

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

struct point{
   float x;
   float y;
};

 void add(struct point *p1,int n);
  void display(struct point *p1,int n);
float calculateDistance(struct point p1, struct point p2) ;
void findClosestPair(struct point *p1, int n) ;




int main(){
   struct point *p1;
   int n;
   printf("enter the size");
   scanf("%d",&n);
   p1=(struct point*)malloc(n*sizeof(struct point));
   add(p1,n);
   display(p1,n);
   findClosestPair(p1,n);

}
```

```
 void add(struct point *p1,int n){
   for(int i=0;i<n;i++){
       printf("enter the  x ");
       scanf("%f",&p1[i].x);
       printf("enter the second y");
       scanf("%f",&p1[i].y);
   }
 }
 void display(struct point *p1,int n){

   for(int i=0;i<n;i++){
        printf("Point %d: (%.2f, %.2f)\n", i + 1, p1[i].x, p1[i].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));
}
void findClosestPair(struct point *p1, int n) {
   if (n < 2) {
       printf("At least two points are required to find the closest pair.\n");
       return;
   }
   int closestPairIndex1 = 0, closestPairIndex2 = 1;
   float minDistance = calculateDistance(p1[0], p1[1]);
   for (int i = 0; i < n - 1; i++) {
       for (int j = i + 1; j < n; j++) {
           float distance = calculateDistance(p1[i], p1[j]);
           if (distance < minDistance) {
               minDistance = distance;
               closestPairIndex1 = i;
               closestPairIndex2 = j;
           }
       }
   }

   printf("\nThe closest pair of points are:\n");
   printf("Point %d: (%.2f, %.2f)\n", closestPairIndex1 + 1, p1[closestPairIndex1].x,
p1[closestPairIndex1].y);
   printf("Point %d: (%.2f, %.2f)\n", closestPairIndex2 + 1, p1[closestPairIndex2].x,
p1[closestPairIndex2].y);
   printf("Distance between the closest pair: %.2f\n", minDistance);
}
```

// Problem Statement: Vehicle Registration System
// Write a C program to simulate a vehicle registration system using unions to handle different types of vehicles. The program should:
// Define a union named Vehicle with the following members:
// car_model (character array of size 50): To store the model name of a car.
// bike_cc (integer): To store the engine capacity (in CC) of a bike.
// bus_seats (integer): To store the number of seats in a bus.
// Create a structure VehicleInfo that contains:
// type (character): To indicate the type of vehicle (C for car, B for bike, S for bus).
// Vehicle (the union defined above): To store the specific details of the vehicle based on its type.
// Implement the following features:
// Input Details: Prompt the user to input the type of vehicle and its corresponding details:
// For a car: Input the model name.
// For a bike: Input the engine capacity.
// For a bus: Input the number of seats.
// Display Details: Display the details of the vehicle based on its type.
// Use the union effectively to save memory and ensure only relevant information is stored.

// Constraints
// The type of vehicle should be one of C, B, or S.
// For invalid input, prompt the user again.
// Sample Input/Output

// Input:
// Enter vehicle type (C for Car, B for Bike, S for Bus): C
// Enter car model: Toyota Corolla
// Output:
// Vehicle Type: Car
// Car Model: Toyota Corolla

// Input:
// Enter vehicle type (C for Car, B for Bike, S for Bus): B
// Enter bike engine capacity (CC): 150
// Output:
// Vehicle Type: Bike

```
// Engine Capacity: 150 CC

// Input:
// Enter vehicle type (C for Car, B for Bike, S for Bus): S
// Enter number of seats in the bus: 50
// Output:
// Vehicle Type: Bus
// Number of Seats: 50


 #include<stdio.h>

 union Vehicle{
    char carmodel[50];
    int bikecc;
    int busseat;
 };
 struct vechicleinfo{
    char type;
    union Vehicle vehicle;
 };
  void add(struct vechicleinfo *v1);
   void display(struct vechicleinfo *v1);


 int main(){
    struct vechicleinfo v1;
    add(&v1);
    display(&v1);

 }
 void add(struct vechicleinfo *v1){
    printf("enter C for car b for bike s for bus");
    scanf("%c",&v1->type);
    if(v1->type!='C' && v1->type!='B'&& v1->type!='S'){
        printf("invalid input please enter c b or s");
        scanf("%c",&v1->type);
    }
    else if(v1->type=='C'){
        printf("enter the car model");
        scanf("%s",v1->vehicle.carmodel);

    }
    else if(v1->type=='B'){
```

```
        printf("enter the car model");
       scanf("%d",&v1->vehicle.bikecc);

   }
   else if (v1->type=='S')
   {
        printf("enter the car model");
       scanf("%d",&v1->vehicle.busseat);

   }

}
void display(struct vechicleinfo *v1){
   if(v1->type=='C'){
       printf("the car model");
       printf("%s",v1->vehicle.carmodel);
   }
   else if(v1->type=='B'){
       printf("the bike model is");
       printf("%d",v1->vehicle.bikecc);
   }
   else if(v1->type=='S'){
       printf("the bus model");
       printf("%d",v1->vehicle.busseat);
   }
}
```

// 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:
// 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."
// has context menu


#include <stdio.h>
enum UserRole {
    ADMIN = 0,
    EDITOR,
    VIEWER,
    GUEST
};

int main() {
    enum UserRole role;
    printf("Select a user role:\n");
    printf("0: ADMIN\n");
    printf("1: EDITOR\n");
    printf("2: VIEWER\n");
    printf("3: GUEST\n");
    printf("Enter your choice: ");
    scanf("%d", &role);
    switch (role) {
        case 0:
            printf(" Full access to the system.\n");
            break;
        case 1:
            printf(" Can edit content but not manage users.\n");
            break;
        case 2:
            printf(" Can view content only.\n");
            break;
        case 3:
            printf("Limited access, view public content only.\n");
            break;
        default:
            printf("Please enter a number between 0 and 3.\n");
            break;
    }
```

```
    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:
// 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>

enum shapes {
    circle = 0,
    rectangle,
    triangle
};

int main() {
    enum shapes var;
    float area;

    printf("0 for CIRCLE\n1 for RECTANGLE\n2 for TRIANGLE\n");
    printf("Enter your choice: ");
    scanf("%d", &var);

    switch (var) {
        case 0: {
            float radius;
            printf("Enter the radius: ");
            scanf("%f", &radius);
            area = 3.14 * radius * radius;
            printf("The area of the circle is: %.2f\n", area);
            break;
        }
        case 1: {
```

```
            int length, breadth;
            printf("Enter the length: ");
            scanf("%d", &length);
            printf("Enter the breadth: ");
            scanf("%d", &breadth);
            area = length * breadth;
            printf("The area of the rectangle is: %.2f\n", area);
            break;
        }
        case 2: {
            float base, height;
            printf("Enter the base of the triangle: ");
            scanf("%f", &base);
            printf("Enter the height of the triangle: ");
            scanf("%f", &height);
            area = 0.5 * base * height;
            printf("The area of the triangle is: %.2f\n", area);
            break;
        }
        default:
            printf("Invalid choice. Please enter 0, 1, or 2.\n");
            break;
    }

    return 0; // Return statement added
}
```

```
// 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:
// 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: The rest

#include<stdio.h>

enum weekday{
    moday=1,
```

```
    tuesday,
    wednesday,
    thursday,
    friday,
    saturday,
    sunday
};
int main(){
    enum weekday var;
    printf("enter a number from 1 to 7");
    scanf("%d",&var);
    switch (var)
    {
    case 1:
        printf("monday is a week day");
        break;
     case 2:
        printf("tuesday is a week day");
        break;
     case 3:
        printf("wednesday is a week day");
        break;
     case 4:
        printf("thursday is a week day");
        break;
     case 5:
        printf("friday is a week day");
        break;
     case 6:
        printf("saturday is a weekend");
        break;
     case 7:
        printf("sunday is a weekend");
        break;

    default:
    printf("inavalid number");
        break;
    }
}
```

```
// Problem 1: Traffic Light System
// Problem Statement:
// 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 traffic{
   RED,
   YELLOW,
   GREEN
};
int main(){
   enum traffic var;
   printf("enter the values 0 for RED, 1 for YELLOW, 2 for GREEN");
   scanf("%d",&var);
   switch (var)
   {
   case 0:
      printf("stop");
       break;
   case 1:
      printf("ready to move");
       break;

   case 2:
      printf("go");
       break;

   default:
   printf("invalid input");
       break;
   }
```

```
}
```

```
// Problem 1: 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.

#include<stdio.h>

struct Date {
    unsigned int day : 5;
    unsigned int month : 4;
    unsigned int year : 12;
};
int main(){
     unsigned int day ;
    unsigned int month ;
    unsigned int year;


    struct Date date[5];
    for(int i=0;i<5;i++){
        printf("enter the dates %d",i+1);
        scanf("%u %u %u",&day,&month,&year);
        date[i].day = day;
        date[i].month = month;
        date[i].year = year;

    }
    for (int i = 0; i < 5; i++) {
```

```
        printf("%u-%u-%u\n", date[i].day, date[i].month, date[i].year);
    }
}
```

// Problem 2: Status Flags for a Device
// Problem Statement:
// Write a C program to manage the status of a device using bit-fields. The program should:
// 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).

```
#include <stdio.h>

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

void displayStatus(struct DeviceStatus device);

int main() {
    struct DeviceStatus device = {0, 0, 0};
    int choice;

    printf("Device Status Management\n");
```

```
while(1) {
    printf("\nMenu:\n");
    printf("1. Turn Power ON\n");
    printf("2. Turn Power OFF\n");
    printf("3. Connect Device\n");
    printf("4. Disconnect Device\n");
    printf("5. Set Error\n");
    printf("6. Clear Error\n");
    printf("7. Display Status\n");
    printf("8. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            device.power = 1;
            printf("Power turned ON.\n");
            break;
        case 2:
            device.power = 0;
            printf("Power turned OFF.\n");
            break;
        case 3:
            device.connection = 1;
            printf("Device connected.\n");
            break;
        case 4:
            device.connection = 0;
            printf("Device disconnected.\n");
            break;
        case 5:
            device.error = 1;
            printf("Error set.\n");
            break;
        case 6:
            device.error = 0;
            printf("Error cleared.\n");
            break;
        case 7:
            displayStatus(device);
            break;
        case 8:
            printf("Exiting...\n");
            break;
```

```
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
void displayStatus(struct DeviceStatus device) {
    printf("\nDevice Status:\n");
    printf("Power: %s\n", device.power ? "ON" : "OFF");
    printf("Connection: %s\n", device.connection ? "CONNECTED" : "DISCONNECTED");
    printf("Error: %s\n", device.error ? "YES" : "NO");
}
```

// Problem 3: 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>

struct FilePermissions {
    unsigned int read : 1;
    unsigned int write : 1;
    unsigned int execute : 1;
};


void displayPermissions(struct FilePermissions permissions) ;
```

```
int main() {
    struct FilePermissions file = {0, 0, 0}; // Initialize all permissions to denied
    int choice;

    printf("File Permissions Management\n");

    // Menu to set or clear permissions
    while(1) {
        printf("\nMenu:\n");
        printf("1. Grant Read Permission\n");
        printf("2. Revoke Read Permission\n");
        printf("3. Grant Write Permission\n");
        printf("4. Revoke Write Permission\n");
        printf("5. Grant Execute Permission\n");
        printf("6. Revoke Execute Permission\n");
        printf("7. Display Permissions\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                file.read = 1;
                printf("Read permission granted.\n");
                break;
            case 2:
                file.read = 0;
                printf("Read permission revoked.\n");
                break;
            case 3:
                file.write = 1;
                printf("Write permission granted.\n");
                break;
            case 4:
                file.write = 0;
                printf("Write permission revoked.\n");
                break;
            case 5:
                file.execute = 1;
                printf("Execute permission granted.\n");
                break;
            case 6:
                file.execute = 0;
```

```
            printf("Execute permission revoked.\n");
            break;
        case 7:
            displayPermissions(file);
            break;
        case 8:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}
void displayPermissions(struct FilePermissions permissions) {
   printf("\nCurrent File Permissions:\n");
   printf("R:%d W:%d X:%d\n", permissions.read, permissions.write, permissions.execute);
}
```

CLASS WORK

```
#include<stdio.h>
enum math{
   add=1,
   sub,
   divi=20
};
```

```
int main(){
    enum math var1=divi;
    printf("the value of var1 :%d",var1);
    return 0;
}
```

```
#include<stdio.h>

struct s2{
    int b;
    int e;
    short c;
    char a;
    char d;
};
```

```
int main(){
struct s2 s1;
printf("the size od s1=%d",sizeof(s1));

}
```

```
#include<stdio.h>

union{
    int a;
    int b;

}var;

int main(){
    var.a=10;
    var.b=20;
    //the output geting is 20 and 20
    // both are sharing same memory locations
```

```
    printf("a=%d,b=%d",var.a,var.b);
    return 0;
}
```

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



int main(){
    enum math var1=add;

    printf("size of var1 =%d\n",sizeof(var1));
    switch(var1){
        case 1:
        printf("additional operation");
        break;
        case 2:
        printf("substraction operation");
        break;
        case 3:
        printf("division operation");
        break;

    }
    printf("the value of var1 :%d",var1);
    return 0;
}
```

```c
#include<stdio.h>

union test{
    int a;
    int b;

}var;

int main(){
    union test *ptr;
    ptr = &var;
    ptr->a=10;
    printf("a=%d,b=%d",ptr->a,ptr->b);
    ptr->b=20;
    printf("a=%d,b=%d",ptr->a,ptr->b);

    return 0;
}
```

```c
//functions
#include<stdio.h>
union test{
    int a;
    int b;
}var;
void add(union test *);
int main(){
    union test *ptr=&var;


    var.a=10;
     printf("a=%d,b=%d\n",var.a,var.b);

     add(ptr);

    //var.b=20;
    //printf("a=%d,b=%d\n",var.a,var.b);

     //add(ptr);
```

```
     //ptr->b=20;
    //printf("a=%d,b=%d",ptr->a,ptr->b);

    return 0;
}
void add(union test *ptr1){
    ptr1->a=30;
    int sum=0;
    sum=ptr1->a+ptr1->b;
    printf("sum=%d\n",sum);
}
```

```
// forced allignment
#include<stdio.h>
struct  t1{
  int a:5;
  int b:8;
};
int main(){
    struct t1 test;
    printf("size of test:%ld\n",sizeof(test));



    return 0;
}
```