1. Implement Fuzzy and Crisp Composition using max-min and max-product mechanisms (Write 4 different files).

    a. Fuzzy max-min composition:

Code:

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

float min(float, float);
void fuzzy(int r,int c1,int c2,float **P,float **Q,float **R);

int main()
{
   int r1, c1, r2, c2;
   float **A, **B, **C;
   printf("Enter the order of the first matrix:\t");
   scanf("%d %d", &r1, &c1);
   printf("Enter the order of second matrix:\t");
   scanf("%d %d", &r2, &c2);

   if (c1 != r2)
   {
       printf("Invalid case");
       exit(0);
   }
   A = mem_alloc_2D(r1, c1);
   B = mem_alloc_2D(r2, c2);
   C = mem_alloc_2D(r1, c2);
   printf("\nEnter the elements of matrix A:\n");
   input(r1, c1, A);
   printf("\nEnter the elements of matrix B:\n");
   input(r2, c2, B);

   printf("\nMatrix A=\n");
   print(r1, c1, A);
   printf("\nMatrix B=\n");
   print(r2, c2, B);
```

```
    printf("\nResultant fuzzy matrix :\n");
    fuzzy(r1,c1,c2,A,B,C);
    print(r1, c2, C);
}
float min(float a, float b)
{
    if (a < b)
        return a;
    return b;
}
void fuzzy(int r, int c1, int c2, float **P, float **Q, float
**R)
{
    int i, j, k;
    float * temp;
    temp= (float *)calloc(c1,sizeof(float));
    for(i = 0; i < r; i++) {
        for(j = 0; j < c2; j++) {
            for(k = 0; k < c1; k++) {
                temp[k] = P[i][k] < Q[k][j] ? P[i][k] :
Q[k][j];
            }

            R[i][j] = temp[0];
            for(k = 1; k < c1; k++) {
                R[i][j] =R[i][j] > temp[k] ? R[i][j] : temp[k];
            }
        }
    }
}
```

Fuzzy_header.h:

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

float **mem_alloc_2D(int, int);
void input(int, int, float **a);
void print(int, int, float **a);
```

```c
float **mem_alloc_2D(int r, int c)
{
    int i;
    float **D;
    D = (float **)calloc(r, sizeof(float *));
    for (i = 0; i < r; i++)
        D[i] = (float *)calloc(c, sizeof(float));
    if (D == NULL)
    {
        printf("Memory not allocated.");
        exit(0);
    }
    return D;
}
void input(int r, int c, float **a)
{
    int i, j;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
        {
            printf("Enter the value of a[%d][%d]", i, j);
            scanf("%f", &a[i][j]);
            if (a[i][j] < 0 || a[i][j] > 1)
            {
                printf("Input should lie between 0 and 1\n");
                j--;
            }
        }
    }
}
void print(int r, int c, float **a)
{
    int i, j;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
            printf("%f\t", a[i][j]);
        printf("\n");
```

```
    }
}
```

OUTPUT:

```
surajit@surajit in ~/Documents/sem6/DSE4 via C
v12.2.1-gcc took 98ms
└─λ gcc max_min_fuzzy.c

┌─surajit@surajit in ~/Documents/sem6/DSE4 via
C v12.2.1-gcc took 385ms
└─λ ./a.out
Enter the order of the first matrix:    2 2
Enter the order of second matrix:       2 3

Enter the elements of matrix A:
Enter the value of a[0][0]0.7
Enter the value of a[0][1]0.5
Enter the value of a[1][0]1.2
Input should lie between 0 and 1
Enter the value of a[1][0]0.8
Enter the value of a[1][1]0.4

Enter the elements of matrix B:
Enter the value of a[0][0]0.9
Enter the value of a[0][1]0.6
Enter the value of a[0][2]0.2
Enter the value of a[1][0]0.1
Enter the value of a[1][1]0.7
Enter the value of a[1][2]0.5

Matrix A=
0.700000        0.500000
0.800000        0.400000
```

```
Matrix B=
0.900000        0.600000        0.200000
0.100000        0.700000        0.500000

Resultant fuzzy matrix :
0.700000        0.600000        0.500000
0.800000        0.600000        0.400000
```

b. Fuzzy Max_dot composition:

Code:

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

void fuzzy(int r,int c1,int c2,float **P,float **Q,float **R);

int main()
{
   int r1, c1, r2, c2;
   float **A, **B, **C;
   printf("Enter the order of the first matrix:\t");
   scanf("%d %d", &r1, &c1);
   printf("Enter the order of second matrix:\t");
   scanf("%d %d", &r2, &c2);

   if (c1 != r2)
   {
       printf("Invalid case");
       exit(0);
   }
   A = mem_alloc_2D(r1, c1);
   B = mem_alloc_2D(r2, c2);
   C = mem_alloc_2D(r1, c2);
   printf("\nEnter the elements of matrix A:\n");
   input(r1, c1, A);
   printf("\nEnter the elements of matrix B:\n");
   input(r2, c2, B);
```

```
    printf("\nMatrix A=\n");
    print(r1, c1, A);
    printf("\nMatrix B=\n");
    print(r2, c2, B);
    printf("\nResultant fuzzy matrix :\n");
    fuzzy(r1,c1,c2,A,B,C);
    print(r1, c2, C);
}

void fuzzy(int r, int c1, int c2, float **P, float **Q, float
**R)
{
    int i, j, k;
    float m, max = -9999;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c2; j++)
        {
            max=-9999;
            for (k = 0; k < c1; k++)
            {
                m = P[i][k] * Q[k][j];
                if (m > max)
                    max = m;
            }
            R[i][j] = max;
        }
    }
}
```

Fuzzy_header.h:

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

float **mem_alloc_2D(int, int);
void input(int, int, float **a);
void print(int, int, float **a);
```

```c
float **mem_alloc_2D(int r, int c)
{
    int i;
    float **D;
    D = (float **)calloc(r, sizeof(float *));
    for (i = 0; i < r; i++)
        D[i] = (float *)calloc(c, sizeof(float));
    if (D == NULL)
    {
        printf("Memory not allocated.");
        exit(0);
    }
    return D;
}
void input(int r, int c, float **a)
{
    int i, j;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
        {
            printf("Enter the value of a[%d][%d]", i, j);
            scanf("%f", &a[i][j]);
            if (a[i][j] < 0 || a[i][j] > 1)
            {
                printf("Input should lie between 0 and 1\n");
                j--;
            }
        }
    }
}
void print(int r, int c, float **a)
{
    int i, j;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
            printf("%f\t", a[i][j]);
        printf("\n");
```

```
    }
}
```

OUTPUT:

```
┌─surajit@surajit in ~/Documents/sem6/DSE4 via
C v12.2.1-gcc took 1ms
└─λ gcc max_dot_fuzzy.c

┌─surajit@surajit in ~/Documents/sem6/DSE4 via
C v12.2.1-gcc took 44ms
└─λ ./a.out
Enter the order of the first matrix:    2 2
Enter the order of second matrix:       2 3

Enter the elements of matrix A:
Enter the value of a[0][0]0.7
Enter the value of a[0][1]0.5
Enter the value of a[1][0]0.8
Enter the value of a[1][1]0.4

Enter the elements of matrix B:
Enter the value of a[0][0]0.9
Enter the value of a[0][1]0.6
Enter the value of a[0][2]0.2
Enter the value of a[1][0]0.1
Enter the value of a[1][1]0.7
Enter the value of a[1][2]0.5

Matrix A=
0.700000        0.500000
0.800000        0.400000

Matrix B=
0.900000        0.600000        0.200000
```

```
0.100000        0.700000        0.500000

Resultant fuzzy matrix :
0.630000        0.420000        0.250000
0.720000        0.480000        0.200000
```

c. **Crisp max_min composition:**

Code:

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


void crisp(int r, int c1, int c2, int **P, int **Q, int **R);

int main()
{
   int r1, c1, r2, c2;
   int **A, **B, **C;
   printf("Enter the order of the first matrix:\t");
   scanf("%d %d", &r1, &c1);
   printf("Enter the order of second matrix:\t");
   scanf("%d %d", &r2, &c2);

   if (c1 != r2)
   {
       printf("Invalid case");
       exit(0);
   }
   A = mem_alloc_2D(r1, c1);
   B = mem_alloc_2D(r2, c2);
   C = mem_alloc_2D(r1, c2);
   printf("\nEnter the elements of matrix A:\n");
   input(r1, c1, A);
   printf("\nEnter the elements of matrix B:\n");
   input(r2, c2, B);

   printf("\nMatrix A=\n");
```

```
    print(r1, c1, A);
    printf("\nMatrix B=\n");
    print(r2, c2, B);
    printf("\nResultant crisp matrix :\n");
    crisp(r1, c1, c2, A, B, C);
    print(r1, c2, C);
}

void crisp(int r, int c1, int c2, int **P, int **Q, int **R)
{
    int i, j, k;
    float *temp;
    temp = (float *)calloc(c1, sizeof(float));
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c2; j++)
        {
            for (k = 0; k < c1; k++)
            {
                temp[k] = P[i][k] < Q[k][j] ? P[i][k] :
Q[k][j];
            }

            R[i][j] = temp[0];
            for (k = 1; k < c1; k++)
            {
                R[i][j] = R[i][j] > temp[k] ? R[i][j] :
temp[k];
            }
        }
    }
}
```

Crisp_header.h:

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

int **mem_alloc_2D(int r, int c);
void input(int r, int c, int **a);
```

```c
void print(int r, int c, int **a);

int **mem_alloc_2D(int r, int c)
{
   int i, **D;
   D = (int **)calloc(r, sizeof(int *));
   for (i = 0; i < r; i++)
       D[i] = (int *)calloc(c, sizeof(int));
   if (D == NULL)
   {
       printf("Memory not allocated.");
       exit(0);
   }
   return D;
}
void input(int r, int c, int **a)
{
   int i, j;
   for (i = 0; i < r; i++)
   {
       for (j = 0; j < c; j++)
       {
           printf("Enter the value of a[%d][%d]", i, j);
           scanf("%d", &a[i][j]);
           if (a[i][j] != 0 && a[i][j] != 1)
           {
               printf("Input should be either 0 or 1\n");
               j--;
           }
       }
   }
}
void print(int r, int c, int **a)
{
   int i, j;
   for (i = 0; i < r; i++)
   {
       for (j = 0; j < c; j++)
           printf("%d\t", a[i][j]);
       printf("\n");
```

```
    }
}
```

OUTPUT:

```
┌─surajit@surajit in ~/Documents/sem6/DSE4 via C
v12.2.1-gcc took 1ms
└─λ gcc max_min_crisp.c

┌─surajit@surajit in ~/Documents/sem6/DSE4 via C
v12.2.1-gcc took 45ms
└─λ ./a.out
Enter the order of the first matrix:    2 2
Enter the order of second matrix:       2 3

Enter the elements of matrix A:
Enter the value of a[0][0]1
Enter the value of a[0][1]1
Enter the value of a[1][0]0
Enter the value of a[1][1]1

Enter the elements of matrix B:
Enter the value of a[0][0]0
Enter the value of a[0][1]1
Enter the value of a[0][2]0
Enter the value of a[1][0]1
Enter the value of a[1][1]1
Enter the value of a[1][2]0

Matrix A=
1       1
0       1

Matrix B=
```

```
0       1       0
1       1       0

Resultant crisp matrix :
1       1       0
1       1       0
```

d. **Crisp max_dot composition:**

Code:

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

void crisp(int r, int c1, int c2, int **P, int **Q, int **R);

int main()
{
   int r1, c1, r2, c2;
   int **A, **B, **C;
   printf("Enter the order of the first matrix:\t");
   scanf("%d %d", &r1, &c1);
   printf("Enter the order of second matrix:\t");
   scanf("%d %d", &r2, &c2);

   if (c1 != r2)
   {
       printf("Invalid case");
       exit(0);
   }
   A = mem_alloc_2D(r1, c1);
   B = mem_alloc_2D(r2, c2);
   C = mem_alloc_2D(r1, c2);
   printf("\nEnter the elements of matrix A:\n");
   input(r1, c1, A);
   printf("\nEnter the elements of matrix B:\n");
   input(r2, c2, B);

   printf("\nMatrix A=\n");
```

```
    print(r1, c1, A);
    printf("\nMatrix B=\n");
    print(r2, c2, B);
    printf("\nResultant crisp matrix :\n");
    crisp(r1, c1, c2, A, B, C);
    print(r1, c2, C);
}

void crisp(int r, int c1, int c2, int **P, int **Q, int **R)
{
    int i, j, k, m, max = -9999;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c2; j++)
        {
            max = -9999;
            for (k = 0; k < c1; k++)
            {
                m = P[i][k] * Q[k][j];
                if (m > max)
                    max = m;
            }
            R[i][j] = max;
        }
    }
}
```

Crisp_header.h:

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

int **mem_alloc_2D(int r, int c);
void input(int r, int c, int **a);
void print(int r, int c, int **a);

int **mem_alloc_2D(int r, int c)
{
    int i, **D;
    D = (int **)calloc(r, sizeof(int *));
```

```
    for (i = 0; i < r; i++)
        D[i] = (int *)calloc(c, sizeof(int));
    if (D == NULL)
    {
        printf("Memory not allocated.");
        exit(0);
    }
    return D;
}
void input(int r, int c, int **a)
{
    int i, j;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
        {
            printf("Enter the value of a[%d][%d]", i, j);
            scanf("%d", &a[i][j]);
            if (a[i][j] != 0 && a[i][j] != 1)
            {
                printf("Input should be either 0 or 1\n");
                j--;
            }
        }
    }
}
void print(int r, int c, int **a)
{
    int i, j;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
            printf("%d\t", a[i][j]);
        printf("\n");
    }
}
```

OUTPUT:

```
┌─surajit@surajit in ~/Documents/sem6/DSE4 via
C v12.2.1-gcc took 1ms
└─λ gcc max_dot_crisp.c

┌─surajit@surajit in ~/Documents/sem6/DSE4 via
C v12.2.1-gcc took 44ms
└─λ ./a.out
Enter the order of the first matrix:    2 2
Enter the order of second matrix:       2 3

Enter the elements of matrix A:
Enter the value of a[0][0]0
Enter the value of a[0][1]1
Enter the value of a[1][0]5
Input should be either 0 or 1
Enter the value of a[1][0]1
Enter the value of a[1][1]1

Enter the elements of matrix B:
Enter the value of a[0][0]1
Enter the value of a[0][1]0
Enter the value of a[0][2]1
Enter the value of a[1][0]1
Enter the value of a[1][1]0
Enter the value of a[1][2]0

Matrix A=
0       1
1       1

Matrix B=
1       0       1
1       0       0
```

```
Resultant crisp matrix :
1       0       0
1       0       1
```

2. **Check whether a Fuzzy relation satisfies the equivalence property or not, if no, then display the reason also.**

Code:

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

int reflexive(int, float **);
int symmetric(int, float **);
int transitive(int, float **);

int main()
{
   int r1, c1, x, y, z;
   float **a;
   printf("Enter the order of the matrix:\t");
   scanf("%d %d", &r1, &c1);
   if (r1 != c1)
   {
       printf("It is not a square matrix\n");
       return -1;
   }
   a = mem_alloc_2D(r1, c1);
   printf("Enter the elements of the matrix:\n");
   input(r1, c1, a);
   printf("The matrix is:\n");
   print(r1, c1, a);
   x = reflexive(r1, a);
   y = symmetric(r1, a);
   z = transitive(r1, a);
   if (x == 1 && y == 1 && z == 1)
       printf("The fuzzy relation is equivalence\n");
   else
```

```
        printf("So the fuzzy relation is not equivalence\n");
}
int reflexive(int n, float **a)
{
    for (int i = 0; i < n; i++)
    {
        if (a[i][i] != 1.0)
        {
            printf("Fuzzy relation is not reflexive\n");
            return -1;
        }
    }
    return 1;
}
int symmetric(int n, float **a)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            if (a[i][j] != a[j][i])
            {
                printf("Fuzzy relation is not symmetric\n");
                return -1;
            }
        }
    return 1;
}
int transitive(int n, float **a)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
            {
                if (a[i][j] && a[j][k] && !a[i][k])
                {
                    printf("Fuzzy relation is not transitive\n");
                    return -1;
                }
            }
    return 1;
```

```
}
```

Fuzzy_haeder.h:

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

float **mem_alloc_2D(int, int);
void input(int, int, float **a);
void print(int, int, float **a);

float **mem_alloc_2D(int r, int c)
{
   int i;
   float **D;
   D = (float **)calloc(r, sizeof(float *));
   for (i = 0; i < r; i++)
       D[i] = (float *)calloc(c, sizeof(float));
   if (D == NULL)
   {
       printf("Memory not allocated.");
       exit(0);
   }
   return D;
}
void input(int r, int c, float **a)
{
   int i, j;
   for (i = 0; i < r; i++)
   {
       for (j = 0; j < c; j++)
       {
           printf("Enter the value of a[%d][%d]", i, j);
           scanf("%f", &a[i][j]);
           if (a[i][j] < 0 || a[i][j] > 1)
           {
               printf("Input should lie between 0 and 1\n");
               j--;
           }
       }
```

```
    }
}
void print(int r, int c, float **a)
{
    int i, j;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
            printf("%f\t", a[i][j]);
        printf("\n");
    }
}
```

OUTPUT:

```
┌─surajit@surajit in ~/Documents/sem6/DSE4 via C
v12.2.1-gcc took 1ms
└─λ gcc fuzzy_equivalence.c

┌─surajit@surajit in ~/Documents/sem6/DSE4 via C
v12.2.1-gcc took 45ms
└─λ ./a.out
Enter the order of the matrix:  2 2
Enter the elements of the matrix:
Enter the value of a[0][0]0.5
Enter the value of a[0][1]0.7
Enter the value of a[1][0]0.2
Enter the value of a[1][1]0.5
The matrix is:
0.500000        0.700000
0.200000        0.500000
Fuzzy relation is not reflexive
Fuzzy relation is not symmetric
So the fuzzy relation is not equivalence
```