// Matrix Calculator for Inverse and Determinant and Transpose.cpp :

#include <cmath>

#include <cstdio>

#include <cstring>

#include <string>

#include <string.h>

#include <vector>

#include <iostream>

#include <algorithm>

#include <iomanip>

using namespace std;

double determinant(double matrix[100][100], int n)

{

double det = 0;

double submatrix[100][100];

if (n == 1)

return matrix[0][0];

else if (n == 2)

return ((matrix[0][0] \* matrix[1][1]) - (matrix[1][0] \* matrix[0][1]));

else

{

for (int x = 0; x < n; x++)

{

int a = 0;

for (int i = 1; i < n; i++)

{

int b = 0;

for (int j = 0; j < n; j++)

{

if (j == x)

continue;

submatrix[a][b] = matrix[i][j];

b++;

}

a++;

}

det = det + (pow(-1, x) \* matrix[0][x] \* determinant(submatrix, n - 1));

}

}

return det;

}

int main()

{

char s[1000];

char oper[10];

double matrix[100][100];

double inv[100][100];

double adj[100][100];

double Result[100][100];

double a[1000];

double x = 0;

char v;

int r = 1, c = 0;

cin.getline(s, 1000, '\n');

int nValues = 0;

for (int y = 0; y < strlen(s); y++) /\*Number of Rows in Matrix\*/

{

if (s[y] == ',')

r++;

}

for (int y = 1; s[y] != ','; y++) /\*Number of Columns in Matrix\*/

{

if (s[y] <= 1000 && s[y - 1] != '.' && s[y] != ' ' && (s[y - 1] == '[' || s[y - 1] == ' '))

c++;

if (s[y] <= 1000 && s[y + 1] == ']')

{

break;

}

}

char\* tokenptr = strtok(s, "[,] "); /\*Matrix convertion\*/

a[0] = stod(tokenptr); /\*Convert string into array of doubles\*/

matrix[0][0] = a[0];

while (tokenptr != NULL)

{

a[nValues] = stod(tokenptr);

tokenptr = strtok(NULL, "[,] ");

nValues++;

}

nValues = 0;

for (int w = 0; w <= r - 1; w++)

{

for (int u = 0; u <= c - 1; u++)

{

matrix[w][u] = a[nValues]; /\*Convert double array into 2D array\*/

nValues++;

}

}

char del[] = " ";

cin.getline(oper, 10, '\n');

char\* TPtr = strtok(oper, del);

v = \*TPtr;

if (v == 'T') /\*Transpose of Matrix\*/

{

for (int w = 0; w <= r - 1; w++)

{

for (int u = 0; u <= c - 1; u++)

{

Result[u][w] = matrix[w][u];

nValues++;

}

}

cout << "["; /\*The output of the Transpose Matrix in its form\*/

for (int w = 0; w <= c - 1; w++)

{

for (int u = 0; u <= r - 1; u++)

{

if (u < r - 1)

{

cout << Result[w][u] << " ";

}

else

{

cout << Result[w][u];

}

}

if (w < c - 1)

{

cout << "," << " ";

}

else

{

cout << "]" << endl;

}

}

}

else if (v == 'I') /\*Inverse of Matrix\*/

{

if (r == c)

{

if (r == c)

{

x = determinant(matrix, r);

if (x == 0)

{

cout << "ERROR!" << endl;

}

else

{

if (r == 2) /\*For Inverse Matrix 2x2\*/

{

adj[0][0] = matrix[1][1];

adj[0][1] = -matrix[0][1];

adj[1][0] = -matrix[1][0];

adj[1][1] = matrix[0][0];

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

inv[i][j] = adj[i][j] / x;

}

}

for (int w = 0; w <= r - 1; w++)

{

for (int u = 0; u <= c - 1; u++)

{

Result[w][u] = 0;

for (int p = 0; p <= c - 1; p++)

{

Result[w][u] += inv[w][p] \* matrix[p][u];

}

}

}

cout << "["; /\*The output matrix in its form\*/

for (int w = 0; w <= r - 1; w++)

{

for (int u = 0; u <= c - 1; u++)

{

if (u < c - 1)

{

if (Result[w][u] <= 1 && Result[w][u] >= -1)

{

cout << setprecision(2) << Result[w][u] << " ";

}

else

cout << setprecision(3) << Result[w][u] << " ";

}

else

{

if (Result[w][u] <= 1 && Result[w][u] >= -1)

{

cout << setprecision(2) << Result[w][u];

}

else

cout << setprecision(3) << Result[w][u];

}

}

if (w < r - 1)

{

cout << "," << " ";

}

else

{

cout << "]" << endl;

}

}

}

else if (r == 3) /\*For Inverse Matrix 3x3\*/

{

adj[0][0] = matrix[1][1] \* matrix[2][2] - matrix[2][1] \* matrix[1][2];

adj[0][1] = -(matrix[1][0] \* matrix[2][2] - matrix[2][0] \* matrix[1][2]);

adj[0][2] = matrix[1][0] \* matrix[2][1] - matrix[2][0] \* matrix[1][1];

adj[1][0] = -(matrix[0][1] \* matrix[2][2] - matrix[2][1] \* matrix[0][2]);

adj[1][1] = matrix[0][0] \* matrix[2][2] - matrix[2][0] \* matrix[0][2];

adj[1][2] = -(matrix[0][0] \* matrix[2][1] - matrix[2][0] \* matrix[0][1]);

adj[2][0] = matrix[0][1] \* matrix[1][2] - matrix[1][1] \* matrix[0][2];

adj[2][1] = -(matrix[0][0] \* matrix[1][2] - matrix[1][0] \* matrix[0][2]);

adj[2][2] = matrix[0][0] \* matrix[1][1] - matrix[1][0] \* matrix[0][1];

for (int w = 0; w < 3; w++)

{

for (int u = 0; u < 3; u++)

{

inv[u][w] = adj[w][u];

}

}

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

Result[i][j] = inv[i][j] / x;

}

}

cout << "["; /\*The output of the Transpose Matrix in its form\*/

for (int w = 0; w < 3; w++)

{

for (int u = 0; u < 3; u++)

{

if (u < r - 1)

{

if (Result[w][u] <= 1 && Result[w][u] >= -1)

{

cout << setprecision(2) << Result[w][u] << " ";

}

else

cout << setprecision(3) << Result[w][u] << " ";

}

else

{

if (Result[w][u] <= 1 && Result[w][u] >= -1)

{

cout << setprecision(2) << Result[w][u];

}

else

cout << setprecision(3) << Result[w][u];

}

}

if (w < c - 1)

{

cout << "," << " ";

}

else

{

cout << "]" << endl;

}

}

}

else if (r == 4) /\*For Inverse Matrix 4x4\*/

{

adj[0][0] = matrix[1][1] \* (matrix[3][3] \* matrix[2][2] - matrix[3][2] \* matrix[2][3])

- matrix[2][1] \* (matrix[3][3] \* matrix[1][2] - matrix[3][2] \* matrix[1][3])

+ matrix[3][1] \* (matrix[2][3] \* matrix[1][2] - matrix[2][2] \* matrix[1][3]);

adj[0][1] = -matrix[1][0] \* (matrix[3][3] \* matrix[2][2] - matrix[3][2] \* matrix[2][3])

+ matrix[2][0] \* (matrix[3][3] \* matrix[1][2] - matrix[3][2] \* matrix[1][3])

- matrix[3][0] \* (matrix[2][3] \* matrix[1][2] - matrix[2][2] \* matrix[1][3]);

adj[0][2] = matrix[1][0] \* (matrix[3][3] \* matrix[2][1] - matrix[3][1] \* matrix[2][3])

- matrix[2][0] \* (matrix[3][3] \* matrix[1][1] - matrix[3][1] \* matrix[1][3])

+ matrix[3][0] \* (matrix[2][3] \* matrix[1][1] - matrix[2][1] \* matrix[1][3]);

adj[0][3] = -matrix[1][0] \* (matrix[3][2] \* matrix[2][1] - matrix[3][1] \* matrix[2][2])

+ matrix[2][0] \* (matrix[3][2] \* matrix[1][1] - matrix[3][1] \* matrix[1][2])

- matrix[3][0] \* (matrix[2][2] \* matrix[1][1] - matrix[2][1] \* matrix[1][2]);

adj[1][0] = -matrix[0][1] \* (matrix[3][3] \* matrix[2][2] - matrix[3][2] \* matrix[2][3])

+ matrix[0][2] \* (matrix[2][1] \* matrix[3][3] - matrix[3][1] \* matrix[2][3])

- matrix[0][3] \* (matrix[2][1] \* matrix[3][2] - matrix[2][2] \* matrix[3][1]);

adj[1][1] = matrix[0][0] \* (matrix[3][3] \* matrix[2][2] - matrix[3][2] \* matrix[2][3])

- matrix[0][2] \* (matrix[2][0] \* matrix[3][3] - matrix[3][0] \* matrix[2][3])

+ matrix[0][3] \* (matrix[2][0] \* matrix[3][2] - matrix[2][2] \* matrix[3][0]);

adj[1][2] = -matrix[0][0] \* (matrix[3][3] \* matrix[2][1] - matrix[3][1] \* matrix[2][3])

+ matrix[0][1] \* (matrix[2][0] \* matrix[3][3] - matrix[3][0] \* matrix[2][3])

- matrix[0][3] \* (matrix[2][0] \* matrix[3][1] - matrix[2][1] \* matrix[3][0]);

adj[1][3] = matrix[0][0] \* (matrix[3][2] \* matrix[2][1] - matrix[3][1] \* matrix[2][2])

- matrix[0][1] \* (matrix[3][2] \* matrix[2][0] - matrix[3][0] \* matrix[2][2])

+ matrix[0][2] \* (matrix[2][0] \* matrix[3][1] - matrix[3][0] \* matrix[2][1]);

adj[2][0] = matrix[0][1] \* (matrix[3][3] \* matrix[1][2] - matrix[3][2] \* matrix[1][3])

- matrix[0][2] \* (matrix[3][3] \* matrix[1][1] - matrix[3][1] \* matrix[1][3])

+ matrix[0][3] \* (matrix[3][2] \* matrix[1][1] - matrix[3][1] \* matrix[1][2]);

adj[2][1] = -matrix[0][0] \* (matrix[3][3] \* matrix[1][2] - matrix[3][2] \* matrix[1][3])

+ matrix[0][2] \* (matrix[3][3] \* matrix[1][0] - matrix[3][0] \* matrix[1][3])

- matrix[0][3] \* (matrix[3][2] \* matrix[1][0] - matrix[1][2] \* matrix[3][0]);

adj[2][2] = matrix[0][0] \* (matrix[3][3] \* matrix[1][1] - matrix[3][1] \* matrix[1][3])

- matrix[0][1] \* (matrix[3][3] \* matrix[1][0] - matrix[3][0] \* matrix[1][3])

+ matrix[0][3] \* (matrix[3][1] \* matrix[1][0] - matrix[3][0] \* matrix[1][1]);

adj[2][3] = -matrix[0][0] \* (matrix[3][2] \* matrix[1][1] - matrix[3][1] \* matrix[1][2])

+ matrix[0][1] \* (matrix[3][2] \* matrix[1][0] - matrix[3][0] \* matrix[1][2])

- matrix[0][2] \* (matrix[3][1] \* matrix[1][0] - matrix[3][0] \* matrix[1][1]);

adj[3][0] = -matrix[0][1] \* (matrix[2][3] \* matrix[1][2] - matrix[2][2] \* matrix[1][3])

+ matrix[0][2] \* (matrix[2][3] \* matrix[1][1] - matrix[2][1] \* matrix[1][3])

- matrix[0][3] \* (matrix[2][2] \* matrix[1][1] - matrix[2][1] \* matrix[1][2]);

adj[3][1] = matrix[0][0] \* (matrix[2][3] \* matrix[1][2] - matrix[2][2] \* matrix[1][3])

- matrix[0][2] \* (matrix[2][3] \* matrix[1][0] - matrix[2][0] \* matrix[1][3])

+ matrix[0][3] \* (matrix[2][2] \* matrix[1][0] - matrix[2][0] \* matrix[1][2]);

adj[3][2] = -matrix[0][0] \* (matrix[2][3] \* matrix[1][1] - matrix[2][1] \* matrix[1][3])

+ matrix[0][1] \* (matrix[2][3] \* matrix[1][0] - matrix[2][0] \* matrix[1][3])

- matrix[0][3] \* (matrix[2][1] \* matrix[1][0] - matrix[2][0] \* matrix[1][1]);

adj[3][3] = matrix[0][0] \* (matrix[2][2] \* matrix[1][1] - matrix[2][1] \* matrix[1][2])

- matrix[0][1] \* (matrix[2][2] \* matrix[1][0] - matrix[2][0] \* matrix[1][2])

+ matrix[0][2] \* (matrix[1][0] \* matrix[2][1] - matrix[2][0] \* matrix[1][1]);

for (int w = 0; w < 4; w++)

{

for (int u = 0; u < 4; u++)

{

inv[u][w] = adj[w][u];

}

}

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

Result[i][j] = inv[i][j] / x;

}

}

cout << "["; /\*The output of the Transpose Matrix in its form\*/

for (int w = 0; w < 4; w++)

{

for (int u = 0; u < 4; u++)

{

if (u < r - 1)

{

if (Result[w][u] == -0)

{

cout << 0 << " ";

}

else if (Result[w][u] <= 1 && Result[w][u] >= -1)

{

cout << setprecision(2) << Result[w][u] << " ";

}

else

cout << setprecision(3) << Result[w][u] << " ";

}

else

{

if (Result[w][u] == -0)

{

cout << 0;

}

else if (Result[w][u] <= 1 && Result[w][u] >= -1)

{

cout << setprecision(2) << Result[w][u];

}

else

cout << setprecision(3) << Result[w][u];

}

}

if (w < c - 1)

{

cout << "," << " ";

}

else

{

cout << "]" << endl;

}

}

}

}

}

else

{

cout << "ERROR!" << endl;

}

}

else

{

cout << "ERROR!" << endl;

}

}

else if (v == 'D') /\*Determinant of Matrix\*/

{

if (r == c)

{

x = determinant(matrix, r);

cout << x << endl;

}

else

{

cout << "ERROR!" << endl;

}

}

else

{

cout << "ERROR!" << endl;

}

}