```
using namespace std;
#include <iostream>
void Entering_the_dimentions_of_matrix_A(int& row, int& col);
void  Entering_the_dimentions_of_matrix_B(int& row, int& col);
void  Entering_the_values_of_matrix(long double mat[12][12], int r, int c);
void  Addion_of_two_matrix(long double mat_A[12][12], long double mat_B[12]
  [12], int r_mat_A, int c_mat_A);
void  Subtraction_of_two_matrix(long double mat_A[12][12],long double mat_B
  [12][12], int r_mat_A, int c_mat_A);
void  Mutiplication_of_two_matrix(long double mat_A[12][12], long double
  mat_B[12][12], int r_mat_A, int c_mat_A, int c_mat_B);
long double get_det(long double matrix[12][12], int size_of_matrix);
long double get_sub_det(long double matrix[12][12], int size_of_matrix);
void arrangment_of_function(long double matrix[12][12], int row, int col,
  int x, int y);
void invers_matrix(long double matrix[12][12], int size);

int main()
{
    /*               Declaration of variables                */
    int r_mat_A, c_mat_A, r_mat_B, c_mat_B , kind_of_operation;
    string num;
    long double  mat_A[12][12], mat_B[12][12], valus_mat_A, values_mat_B;
    long long int result_mat[10][10];

    //       Entering_the_dimentions_of_matrix_A AND B


    Entering_the_dimentions_of_matrix_A(r_mat_A, c_mat_A);
    Entering_the_dimentions_of_matrix_B(r_mat_B, c_mat_B);


    // Entering the values of matrix A

    cout << "Please enter values of Matrix A:" << endl;

    Entering_the_values_of_matrix(mat_A, r_mat_A, c_mat_A);

    // Entering the values of matrix B

    cout << "Please enter values of Matrix B:" << endl;

    Entering_the_values_of_matrix(mat_B, r_mat_B, c_mat_B);




    while (1)
    {
        // choose the kind of operation
        cout << "Please choose operation type(1: A+B, 2: A-B, 3: AxB, 4:
          A*inverse(B), 5: |A|, 6: |B|, 7: quit):" << endl;
```

```
        cin >> num ;
        kind_of_operation = atoi(num.c_str());

        if (kind_of_operation == 1)  // Addion mat_A and mat_B

        {
            if ((r_mat_A != r_mat_B) || (c_mat_A != c_mat_B))
            {
                cout << "The operation you chose is invalid for the given
                  matrices." << endl;
            }
            else
            {
                Addion_of_two_matrix(mat_A, mat_B, r_mat_A, c_mat_A);

            }

        }

        else if (kind_of_operation == 2)   // subtraction of mat_B from
          mat_A
        {
            if ((r_mat_A != r_mat_B) || (c_mat_A != c_mat_B))
            {
                cout << "The operation you chose is invalid for the given
                  matrices." << endl;
            }
            else
            {
                Subtraction_of_two_matrix(mat_A, mat_B, r_mat_A, c_mat_A);

            }
        }

        else if (kind_of_operation == 3)    // MULTIBLICATION
        {
            if (c_mat_A == r_mat_B)
            {
                Mutiplication_of_two_matrix(mat_A, mat_B, r_mat_A, c_mat_A,
                  c_mat_B);
            }
            else
            {
                cout << "The operation you chose is invalid for the given
                  matrices." << endl;
            }
        }



        else if (kind_of_operation == 4)    //  DIVITION
        {
            double j = get_det(mat_B, r_mat_B);
```

```
            if ((c_mat_A != r_mat_B) || j == 0 || r_mat_B != c_mat_B)
            {
                cout << "The operation you chose is invalid for the given
                  matrices." << endl;
            }
            else if ((c_mat_A == r_mat_B) && j != 0 && r_mat_B == c_mat_B)
            {
                double rest_mat[12][12];
                for (int row = 0;row < r_mat_B; row++)
                {
                    for (int col = 0;col < c_mat_B;col++)
                    {
                        rest_mat[row][col] = mat_B[row][col];
                    }

                }

                invers_matrix(mat_B, r_mat_B);
                Mutiplication_of_two_matrix(mat_A, mat_B, r_mat_A, c_mat_A,
                  c_mat_B);

                for (int row = 0;row < r_mat_B; row++)
                {
                    for (int col = 0;col < c_mat_B;col++)
                    {
                        mat_B[row][col] = rest_mat[row][col];
                    }

                }

            }
        }

        else if (kind_of_operation == 5) // Determinent of matrix A
        {
            if (r_mat_A != c_mat_A)
            {
                cout << "The operation you chose is invalid for the given
                  matrices." << endl;
            }

            else
            {
                if (get_det(mat_A, r_mat_A) >= 0)
                    cout << long long int(get_det(mat_A, r_mat_A) + .5) <<
                   endl;
                else
                {
                    cout << long long int(get_det(mat_A, r_mat_A) - .5) <<
                   endl;
                }
```

```
            }



        }


        else if (kind_of_operation == 6) // Determinent of matrix B
        {
            if (r_mat_B != c_mat_B)
            {
                cout << "The operation you chose is invalid for the given
                  matrices." << endl;
            }

            else
            {
                if (get_det(mat_B, r_mat_B) >= 0)
                    cout <<long long  int(get_det(mat_B, r_mat_B) + .5) <<
                   endl;
                else
                {
                    cout << long long int(get_det(mat_B, r_mat_B) - .5) <<
                   endl;
                }

            }



        }



        else if (kind_of_operation == 7)
        {
            cout << "Thank you!" << endl;
            break;
        }
        else if (kind_of_operation > 7 || kind_of_operation < 1)    // un
          valid input
        {
            cout << "The operation you chose is invalid for the given
              matrices." << endl;
        }

    }
}



void Entering_the_dimentions_of_matrix_A(int& row, int& col)
{
```

```
    while (1)
    {
        string  r_mat_a, c_mat_a;
        cout << "Please enter dimensions of Matrix A:" << endl;
        cin >> r_mat_a >> c_mat_a;
        int r_mat = atoi(r_mat_a.c_str());
        int c_mat = atoi(c_mat_a.c_str());
        if (r_mat > 0 && c_mat > 0 && r_mat <= 12 && c_mat <= 12)
        {
            row = r_mat;     col = c_mat;
            break;
        }
        else
        {
            cout << "invalid inputs" << endl;
            continue;
        }

    }   //cout << endl;
}

void Entering_the_dimentions_of_matrix_B(int& row, int& col)
{
    while (1)
    {
        string  r_mat_b, c_mat_b;
        cout << "Please enter dimensions of Matrix B:" << endl;
        cin >> r_mat_b >> c_mat_b;
        int r_mat = atoi(r_mat_b.c_str());
        int c_mat = atoi(c_mat_b.c_str());
        if (r_mat > 0 && c_mat > 0 && r_mat <= 12 && c_mat <= 12)
        {
            row = r_mat;     col = c_mat;
            break;
        }
        else
        {
            cout << "invalid inputs" << endl;
            continue;
        }
    }
}
////////////////////////////////////////////////////////////////////////////
  ////////////////////////////////////////

                          /*Get inputs(values of each item in two
                matrixes ) from the user */

void Entering_the_values_of_matrix(long double mat[12][12], int r, int c)
{
    string input;
    for (int row = 0;row < r; row++)
    {
```

```
        for (int col = 0;col < c;col++)
        {
            cin >> input;

            mat[row][col] = atoi(input.c_str());

        }

    }
}
////////////////////////////////////////////////////////////////////////////////
  ////////////////////////////////////////////

                                        /*  Make addtion of mat_a and mat_b
                  */

void Addion_of_two_matrix(long double mat_A[12][12], long double mat_B[12]
  [12], int r_mat_A, int c_mat_A)
{
    long long int result_mat[12][12];
    for (int row = 0; row < r_mat_A;row++)
    {
        for (int col = 0; col < c_mat_A;col++)
        {
            long double x = mat_A[row][col] + mat_B[row][col];
            if (x >= 0)     // in case or the result is positive
            {
                result_mat[row][col] = x + .5;  // approxemation to the
                  nearst positive number
            }
            else   //in case of negative number
            {
                result_mat[row][col] = x - .5;   //  approxemation to the
                  nearst negative number
            }
            cout << result_mat[row][col] << " ";
        }
        cout << endl;
    }
}




void Subtraction_of_two_matrix(long double mat_A[12][12], long double mat_B
  [12][12], int r_mat_A, int c_mat_A)
{
    long long int result_mat[12][12];
    for (int row = 0; row < r_mat_A;row++)
    {
        for (int col = 0; col < c_mat_A;col++)
```

```
        {
            long double x = mat_A[row][col] - mat_B[row][col];
            if (x >= 0)     // in case or the result is positive
            {
                result_mat[row][col] = x + .5;  // approxemation to the
                  nearst positive number
            }
            else   //in case of negative number
            {
                result_mat[row][col] = x - .5;   //  approxemation to the
                  nearst negative number
            }
            cout << result_mat[row][col] << " ";
        }
        cout << endl;
    }
}
```

```
void Mutiplication_of_two_matrix(long double mat_A[12][12], long double
  mat_B[12][12], int r_mat_A, int c_mat_A, int c_mat_B)
{
    long long int result_mat[12][12];  long double x;


    for (int row = 0; row < r_mat_A;row++)
    {

        for (int col = 0; col < c_mat_B;col++)
        {
            x = 0;


            for (int i = 0;i < c_mat_A;i++)
            {

                x = x + mat_A[row][i] * mat_B[i][col];

            }
            if (x >= 0)     // in case or the result is positive
            {
                result_mat[row][col] = x + .5;  // approxemation to the
                  nearst positive number
            }
            else   //in case of negative number
            {
                result_mat[row][col] = x - .5;   //  approxemation to the
                  nearst negative number
            }
```

```
            cout << result_mat[row][col] << " ";

        }

        cout << endl;
    }

}

long double get_det(long double matrix[12][12], int size_of_matrix)
{
    long double det = 0;
    if (size_of_matrix == 1)
    {
        //cout << matrix[0][0];
        return matrix[0][0];
    }
    else if (size_of_matrix == 2)
    {
        return(matrix[0][0] * matrix[1][1]) - (matrix[1][0] * matrix[0][1]);
    }
    else
    {
        long double internal_matrix[12][12];

        for (int i = 0; i < size_of_matrix;i++)
        {
            int internal_row = 0;
            for (int row = 1; row < size_of_matrix; row++)
            {
                int internal_col = 0;
                for (int col = 0;col < size_of_matrix; col++)
                {
                    if (col == i)  continue;
                    internal_matrix[internal_row][internal_col] = matrix
                   [row][col];
                    internal_col++;
                }
                internal_row++;
            }
            det = det + (pow(-1, i) * matrix[0][i] * get_det
              (internal_matrix, size_of_matrix - 1));









        }

        return det;
    }
```

```
}



//                ***( in this function we try to get sub matrix and get its
   determinant to get finally frind matrix)***

long double get_sub_det(long double matrix[12][12], int size_of_matrix)
{
    long double det = 0;

    int size_of_sub_matrix = size_of_matrix - 1;


    if (size_of_sub_matrix == 1)
    {
        return matrix[1][1];
    }
    else if (size_of_sub_matrix == 2)
    {
        return(matrix[1][1] * matrix[2][2]) - (matrix[1][2] * matrix[2][1]);
    }
    else
    {
        long double internal_matrix1[12][12];

        int internal_row1 = 0;

        for (int row = 1; row < size_of_matrix; row++)
        {
            int internal_col1 = 0;
            for (int col = 1;col < size_of_matrix; col++)
            {
                //if (col == i)  continue;
                internal_matrix1[internal_row1][internal_col1] = matrix[row]
                  [col];
                internal_col1++;
            }
            internal_row1++;

        }



        return get_det(internal_matrix1, size_of_sub_matrix);


    }
}
```

```
// arrangment of matrix

//             ***(in this matrix we make arrangment of function for each item to get sub determinant of each item in the matrix)***


void arrangment_of_function(long double matrix[12][12], int row, int col, int x, int y)
{

    long double row_0[12], row_required[12], col_0[12], col_required[12];

    for (int r = 0;r < row; r++)
    {
        for (int c = 0;c < col;c++)
        {
            if (r == 0)
            {
                row_0[c] = matrix[r][c];

            }
            if (r == x)
            {
                row_required[c] = matrix[r][c];
            }

        }


    }


    for (int c = 0;c < col; c++)
    {
        matrix[0][c] = row_required[c];

        matrix[x][c] = row_0[c];

    }


    for (int r = 0; r < row; r++)
    {
        for (int c = 0; c < col; c++)
```

```
        {
            if (c == 0)
            {
                col_0[r] = matrix[r][c];

            }
            if (c == y)
            {
                col_required[r] = matrix[r][c];
            }

        }

    }


    for (int r = 0; r < col; r++)
    {
        matrix[r][0] = col_required[r];

        matrix[r][y] = col_0[r];

    }


}

void invers_matrix(long double matrix[12][12], int size)
{
    long double det = get_det(matrix, size);
    long double restore_mat[12][12], temp_mat[12][12];


    if (size == 1)
    {
        matrix[0][0] = 1 / matrix[0][0];

    }
    else if (size == 2)
    {
        int x = matrix[1][1];
        matrix[1][1] = matrix[0][0];
        matrix[0][1] = -matrix[0][1];
        matrix[1][0] = -matrix[1][0];
        matrix[0][0] = x;
        for (int row = 0;row < size; row++)
        {
            for (int col = 0;col < size;col++)
            {
                matrix[row][col] = matrix[row][col] * (1 / det);

            }
```

```
        }
    }

    else
    {
        for (int row = 0;row < size; row++)
        {
            for (int col = 0;col < size;col++)
            {
                restore_mat[row][col] = matrix[row][col];
            }

        }

        for (int r = 0;r < size;r++)
        {
            for (int c = 0; c < size; c++)
            {
                arrangment_of_function(matrix, size, size, r, c);
                //double item = matrix[0][0];


                if (((r == 0) && (c != 0)) || ((c == 0) && (r != 0)))
                {
                    temp_mat[r][c] = -get_sub_det(matrix, size);
                }
                else
                {
                    temp_mat[r][c] = get_sub_det(matrix, size);
                }

                for (int row = 0;row < size; row++)
                {
                    for (int col = 0;col < size;col++)
                    {
                        matrix[row][col] = restore_mat[row][col];
                    }

                }

            }

        }

        for (int row = 0;row < size; row++)
        {
            for (int col = 0;col < size;col++)
            {
                matrix[col][row] = temp_mat[row][col];

            }

        }
```

```
        for (int row = 0;row < size; row++)
        {
            for (int col = 0;col < size;col++)
            {
                matrix[row][col] = matrix[row][col] * (1 / det);

            }

        }
    }

}
```