

**Name:** Muhammad Mubbashir Khan

**CMS:** 465077

**Section: B**

**Lab manual: 9**

**Task No 1:**

#include <iostream>

using namespace std;

int main()

{

int matrix[3][3];

cout << "Enter the elements of the 3x3 matrix:\n";

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

{

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

{

cin >> matrix[i][j];

}

}

int left\_sum = 0;

int right\_sum = 0;

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

{

left\_sum += matrix[i][i];

right\_sum += matrix[i][3 - i - 1];

}

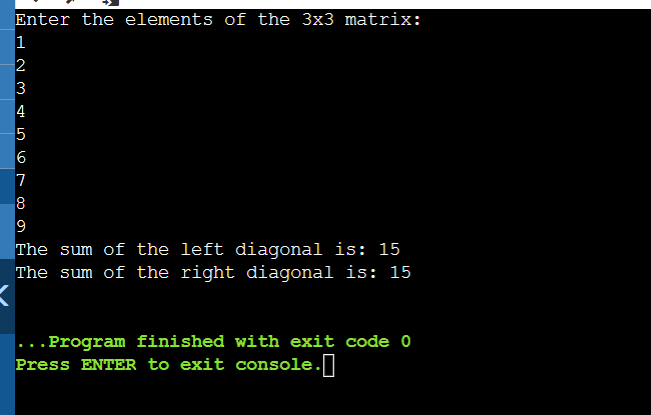
cout << "The sum of the left diagonal is: " << left\_sum << "\n";

cout << "The sum of the right diagonal is: " << right\_sum << "\n";

return 0;

}

**Output:**



**Task No 2:**

#include <iostream>

using namespace std;

void addArrays(int arr1[3][3], int arr2[3][3], int arr3[3][3]) {

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

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

arr3[i][j] = arr1[i][j] + arr2[i][j];

}

}

}

void printArray(int arr[3][3]) {

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

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

cout << arr[i][j] << " ";

}

cout << endl;

}

}

int main() {

3

int arr1[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int arr2[3][3] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

int arr3[3][3];

addArrays(arr1, arr2, arr3);

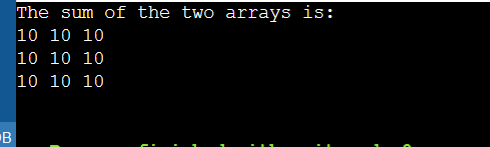
cout << "The sum of the two arrays is:" << endl;

printArray(arr3);

return 0;

}

Output:



**Task No 3:**

#include <iostream>

using namespace std;

// A function to transpose a 3x3 matrix

void transpose(int matrix[3][3]) {

// Loop through the rows and columns of the matrix

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

for (int j = i + 1; j < 3; j++) {

// Swap the elements at (i, j) and (j, i)

int temp = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = temp;

}

}

}

// A function to print a 3x3 matrix

void print\_matrix(int matrix[3][3]) {

// Loop through the rows and columns of the matrix

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

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

// Print the element at (i, j) with a space

cout << matrix[i][j] << " ";

}

// Print a new line after each row

cout << endl;

}

}

int main() {

// A sample 3x3 matrix

int matrix[3][3] = {{1, 2, 3},

{4, 5, 6},

{7, 8, 9}};

// Print the original matrix

cout << "Original matrix:" << endl;

print\_matrix(matrix);

// Transpose the matrix

transpose(matrix);

// Print the transposed matrix

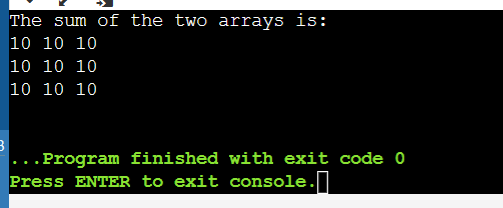
cout << "Transposed matrix:" << endl;

print\_matrix(matrix);

return 0;

}

Output:



**Task No 4:**

#include <iostream>

using namespace std;

// A function to multiply two 3x3 matrices

void multiply(int A[3][3], int B[3][3], int C[3][3]) {

// Loop through the rows of A

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

// Loop through the columns of B

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

// Initialize the element of C at (i,j) to zero

C[i][j] = 0;

// Loop through the common dimension of A and B

for (int k = 0; k < 3; k++) {

// Add the product of A[i][k] and B[k][j] to C[i][j]

C[i][j] += A[i][k] \* B[k][j];

}

}

}

}

// A function to print a 3x3 matrix

void print(int M[3][3]) {

// Loop through the rows of M

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

// Loop through the columns of M

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

// Print the element of M at (i,j) with a space

cout << M[i][j] << " ";

}

// Print a new line after each row

cout << endl;

}

}

int main() {

// Declare and initialize two 3x3 matrices

int A[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int B[3][3] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

// Declare a 3x3 matrix to store the result

int C[3][3];

// Call the multiply function to compute C = A \* B

multiply(A, B, C);

// Print the result

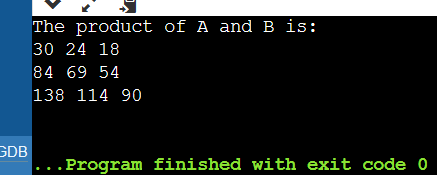
cout << "The product of A and B is:" << endl;

print(C);

return 0;

}

Output:



**Task No 5:**

#include <iostream>

using namespace std;

// A recursive function to print the multiplication table of 15

void print\_table(int n, int i) {

// Base case: when i reaches 11, stop the recursion

if (i == 11) {

return;

}

// Recursive case: print the product of 15 and i, and call the function again with i+1

cout << "15 x " << i << " = " << 15 \* i << endl;

print\_table(n, i + 1);

}

int main() {

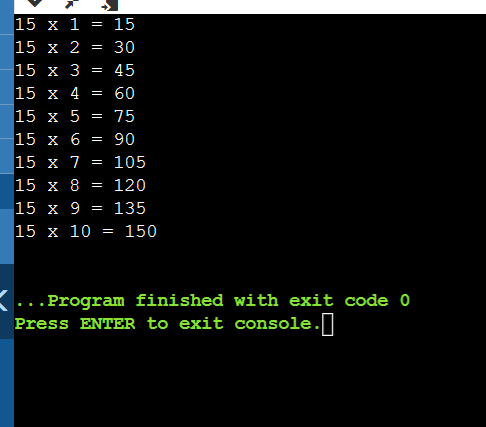
// Call the function with n = 15 and i = 1

print\_table(15, 1);

return 0;

}

Output:



**Home Task**

**Task:**

#include <iostream>

using namespace std;

// Function to calculate the determinant of a 2x2 matrix

double determinant2x2(double a, double b, double c, double d) {

return a \* d - b \* c;

}

// Function to calculate the determinant of a 3x3 matrix

double determinant3x3(double matrix[3][3]) {

return matrix[0][0] \* determinant2x2(matrix[1][1], matrix[1][2], matrix[2][1], matrix[2][2]) -

matrix[0][1] \* determinant2x2(matrix[1][0], matrix[1][2], matrix[2][0], matrix[2][2]) +

matrix[0][2] \* determinant2x2(matrix[1][0], matrix[1][1], matrix[2][0], matrix[2][1]);

}

// Function to calculate the adjoint of a 3x3 matrix

void adjoint3x3(double matrix[3][3], double adj[3][3]) {

adj[0][0] = determinant2x2(matrix[1][1], matrix[1][2], matrix[2][1], matrix[2][2]);

adj[0][1] = -determinant2x2(matrix[0][1], matrix[0][2], matrix[2][1], matrix[2][2]);

adj[0][2] = determinant2x2(matrix[0][1], matrix[0][2], matrix[1][1], matrix[1][2]);

adj[1][0] = -determinant2x2(matrix[1][0], matrix[1][2], matrix[2][0], matrix[2][2]);

adj[1][1] = determinant2x2(matrix[0][0], matrix[0][2], matrix[2][0], matrix[2][2]);

adj[1][2] = -determinant2x2(matrix[0][0], matrix[0][2], matrix[1][0], matrix[1][2]);

adj[2][0] = determinant2x2(matrix[1][0], matrix[1][1], matrix[2][0], matrix[2][1]);

adj[2][1] = -determinant2x2(matrix[0][0], matrix[0][1], matrix[2][0], matrix[2][1]);

adj[2][2] = determinant2x2(matrix[0][0], matrix[0][1], matrix[1][0], matrix[1][1]);

}

// Function to calculate the inverse of a 3x3 matrix

void inverse3x3(double matrix[3][3], double inverse[3][3]) {

double det = determinant3x3(matrix);

if (det == 0) {

cout << "Inverse does not exist as the determinant is zero." << endl;

return;

}

double adj[3][3];

adjoint3x3(matrix, adj);

// Calculate the inverse by dividing each element of the adjoint by the determinant

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

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

inverse[i][j] = adj[i][j] / det;

}

}

}

// Function to display a 3x3 matrix

void displayMatrix(double matrix[3][3]) {

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

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

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

int main() {

double matrix[3][3];

cout << "Enter the elements of the 3x3 matrix:" << endl;

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

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

cin >> matrix[i][j];

}

}

double inverse[3][3];

inverse3x3(matrix, inverse);

cout << "Inverse of the matrix is:" << endl;

displayMatrix(inverse);

return 0;

}

Output:

