**FUNDAMENTALS OF PROGRAMMING**

**LAB MANUAL # 09**

**ABDUL MOIZ AKRAM**

**4559878**

**ME-15-C**

**LAB TASK**

**1 ;**

#include <iostream>

using namespace std;

int main() {

const int size = 3;

int matrix[size][size];

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

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

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

cout << "Enter element at position (" << i + 1 << ", " << j + 1 << "): ";

cin >> matrix[i][j];

}

}

cout << "\nThe matrix is:\n";

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

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

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

}

cout << "\n";

}

int leftDiagonalSum = 0;

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

leftDiagonalSum += matrix[i][i];

}

cout << "\nLeft Diagonal Sum: " << leftDiagonalSum << "\n";

int rightDiagonalSum = 0;

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

rightDiagonalSum += matrix[i][size - 1 - i];

}

cout << "Right Diagonal Sum: " << rightDiagonalSum << "\n";

return 0;

}

**2 ;**

Write a function to add two 2D arrays of size 3x3.

#include <iostream>

using namespace std;

void addMatrices(const int mat1[3][3], const int mat2[3][3], int result[3][3]) {

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

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

result[i][j] = mat1[i][j] + mat2[i][j];

}

}

}

int main() {

const int size = 3;

int matrix1[size][size] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int matrix2[size][size] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

int result[size][size];

addMatrices(matrix1, matrix2, result);

cout << "Matrix after addition:\n";

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

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

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

}

cout << "\n";

}

return 0;

}

**3;**

Using 2D arrays in C++, take transpose of a 3x3 matrix. Make a transpose function.

#include <iostream>

using namespace std;

void transposeMatrix(int matrix[3][3], int result[3][3]) {

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

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

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

}

}

}

int main() {

int originalMatrix[3][3] = {{1, 4, 3}, {4, 9, 6}, {0, 8, 9}};

int transposedMatrix[3][3];

transposeMatrix(originalMatrix, transposedMatrix);

cout << "Original Matrix:" << endl;

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

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

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

}

cout << endl;

}

cout << "\nTransposed Matrix:" << endl;

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

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

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

}

cout << endl;

}

return 0;

}

**4;**

Using 2D arrays in C++, implement 3x3 matrix multiplication. Make a function.

#include <iostream>

using namespace std;

void multiplyMatrices(int matrix1[3][3], int matrix2[3][3], int result[3][3]) {

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

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

result[i][j] = 0;

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

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

}

int main() {

int matrix\_a[3][3] = {{1, 7, 3}, {4, 8, 6}, {7, 5, 9}};

int matrix\_b[3][3] = {{9, 3, 7}, {6, 4, 4}, {3, 9, 1}};

int result\_matrix[3][3];

multiplyMatrices(matrix\_a, matrix\_b, result\_matrix);

cout << "Matrix A:" << endl;

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

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

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

}

cout << endl;

}

cout << "\nMatrix B:" << endl;

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

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

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

}

cout << endl;

}

cout << "\nResultant Matrix:" << endl;

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

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

cout << result\_matrix[i][j] << " ";

}

cout << endl;

}

return 0;

}

**5 ;**

Print the multiplication table of 15 using recursion.

#include <iostream>

using namespace std;

void printMultiplicationTable(int number, int multiplier) {

if (multiplier <= 10) {

cout << number << " x " << multiplier << " = " << number \* multiplier << endl;

printMultiplicationTable(number, multiplier + 1);

}

}

int main() {

int tableNumber = 15;

cout << "Multiplication Table of " << tableNumber << ":" << endl;

printMultiplicationTable(tableNumber, 1);

return 0;

}

**HOME TASK**

Write a C++ program to take inverse of a 3x3 matrix using its determinant and adjoint.

#include<iostream>

#include<cmath>

using namespace std;

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

return a \* d - b \* c;

}

float determinant3x3(float 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 adjointMatrix(float matrix[3][3], float adjoint[3][3]) {

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

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

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

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

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

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

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

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

adjoint[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 inverseMatrix(float matrix[3][3], float inverse[3][3]) {

float det = determinant3x3(matrix);

if (det == 0) {

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

return;

}

float adjoint[3][3];

adjointMatrix(matrix, adjoint);

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

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

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

}

// Function to display a matrix

void displayMatrix(float 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() {

float 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];

float inverse[3][3];

inverseMatrix(matrix, inverse);

cout << "Inverse matrix:" << endl;

displayMatrix(inverse);

return 0;

}