

**NUST**

**School of Mechanical & Manufacturing Engineering**

**FOP LAB MANUAL:09**

**Name:** Muhammad Muzammil Riaz

**Qalam:** 467817

**Batch:** ME-15

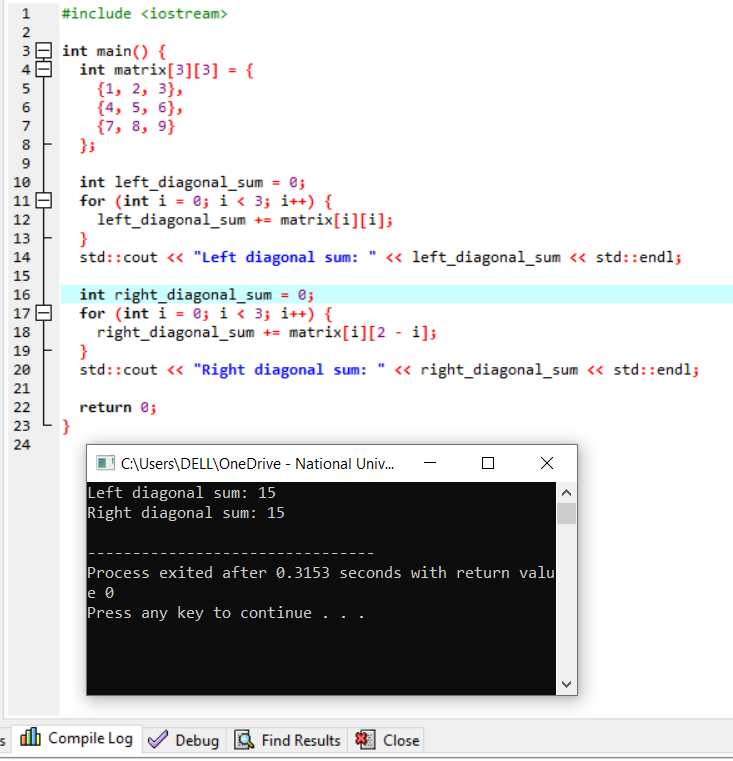
**Section:** A

**Course Instructor:** Dr. Jawad

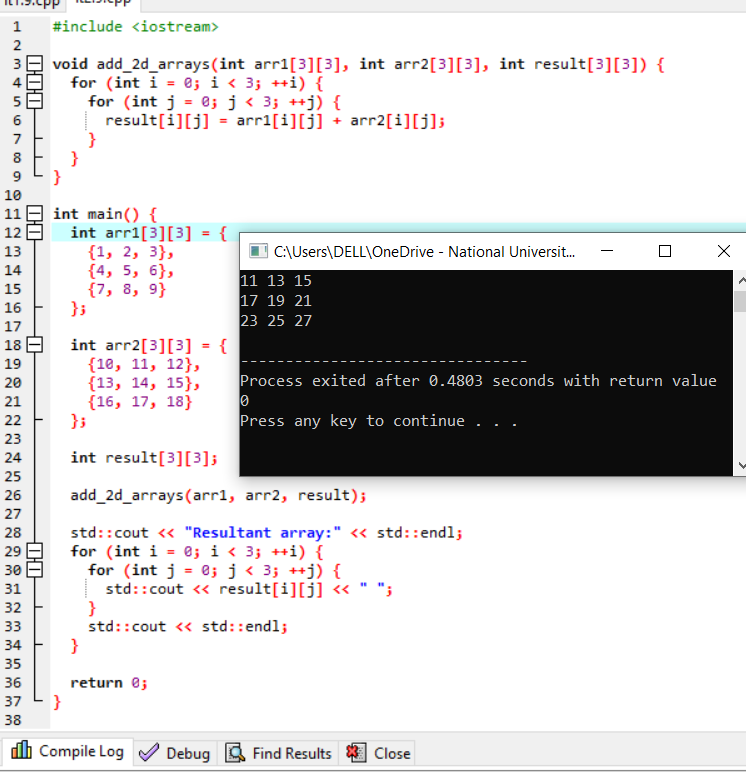
**Lab instructor:** Sir. Saqib

LAB TASK:

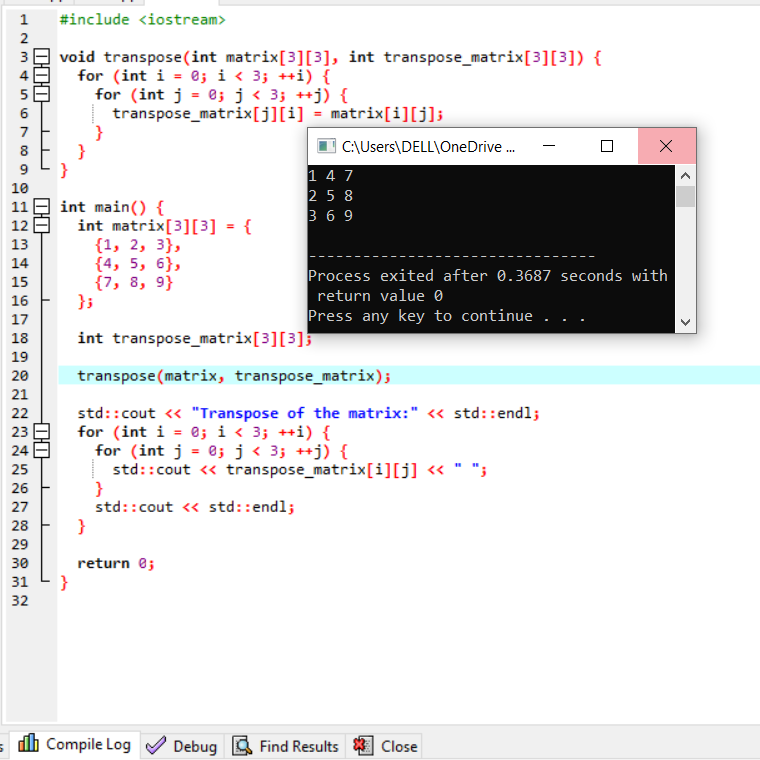
Qno1.



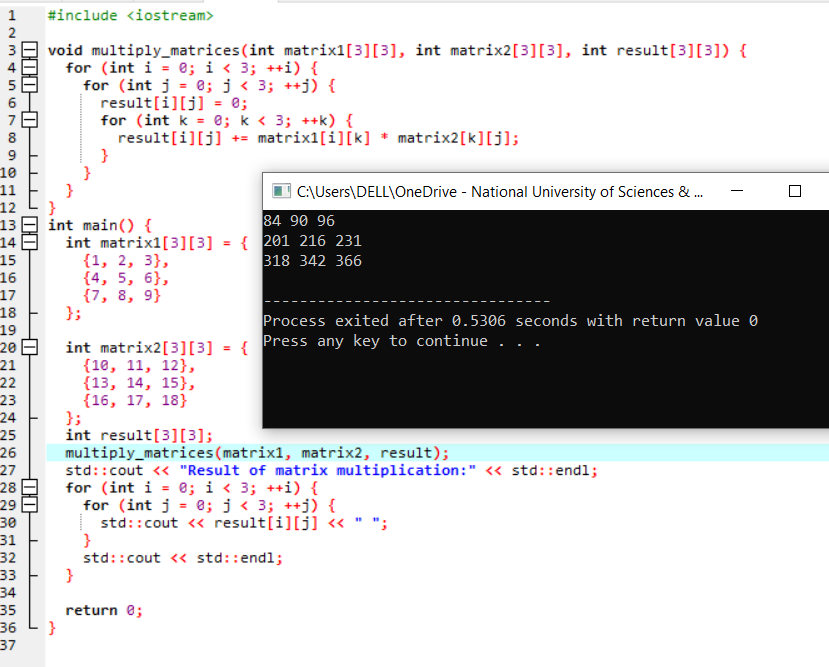
Qno2.



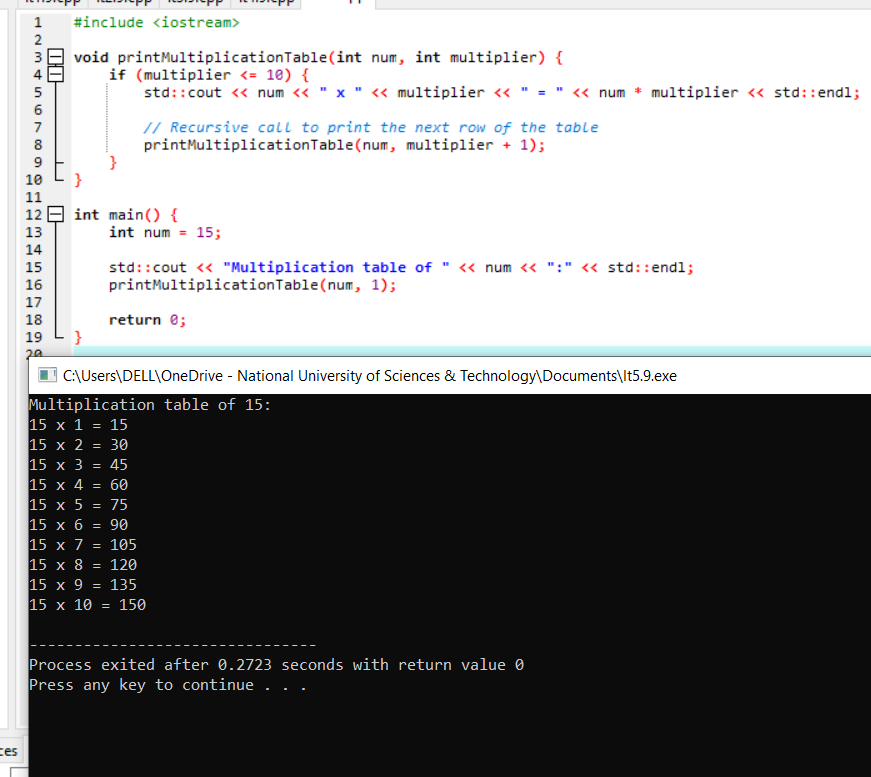
Qno3.



Qno4.



Qno5.



# HOME TASK:

Qno1.

#include <iostream>

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

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

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

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

}

std::cout << std::endl;

}

}

double calculateDeterminant(const double matrix[3][3]) {

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

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

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

}

void calculateAdjugate(const double matrix[3][3], double adjugate[3][3]) {

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

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

int sign = ((i + j) % 2 == 0) ? 1 : -1;

double temp[2][2];

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

int ti = 0;

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

if (m != i && n != j) {

temp[ti][n - ti] = matrix[m][n];

if (n == 2) {

ti++;

}

}

}

}

adjugate[i][j] = sign \* calculateDeterminant(temp);

}

}

}

void calculateInverse(const double matrix[3][3], double inverse[3][3]) {

double det = calculateDeterminant(matrix);

if (det == 0) {

std::cout << "The matrix is singular, and its inverse does not exist." << std::endl;

return;

}

calculateAdjugate(matrix, inverse);

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

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

inverse[i][j] /= det;

}

}

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

displayMatrix(inverse);

}

int main() {

double matrix[3][3];

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

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

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

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

std::cin >> matrix[i][j];

}

}

calculateInverse(matrix, matrix); // Use the same matrix to store the result

return 0;

}