



FUNDAMENTAL OF PROGRAMMING

LAB TASK 9

HAIDER NAWAZ

480239

Make 2D Array in C++ and print left diagonal and right diagonal sum of a 3x3 matrix

CODE:

```
#include <iostream>

using namespace std;

int main() {

    int size = 3;

    int matrix[size][size];

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

    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 << "Entered matrix:" << endl;

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

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

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

        }

        cout << endl;

    }

    int leftDiagonalSum = 0;

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

```

        leftDiagonalSum += matrix[i][i];
    }

    int rightDiagonalSum = 0;

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

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

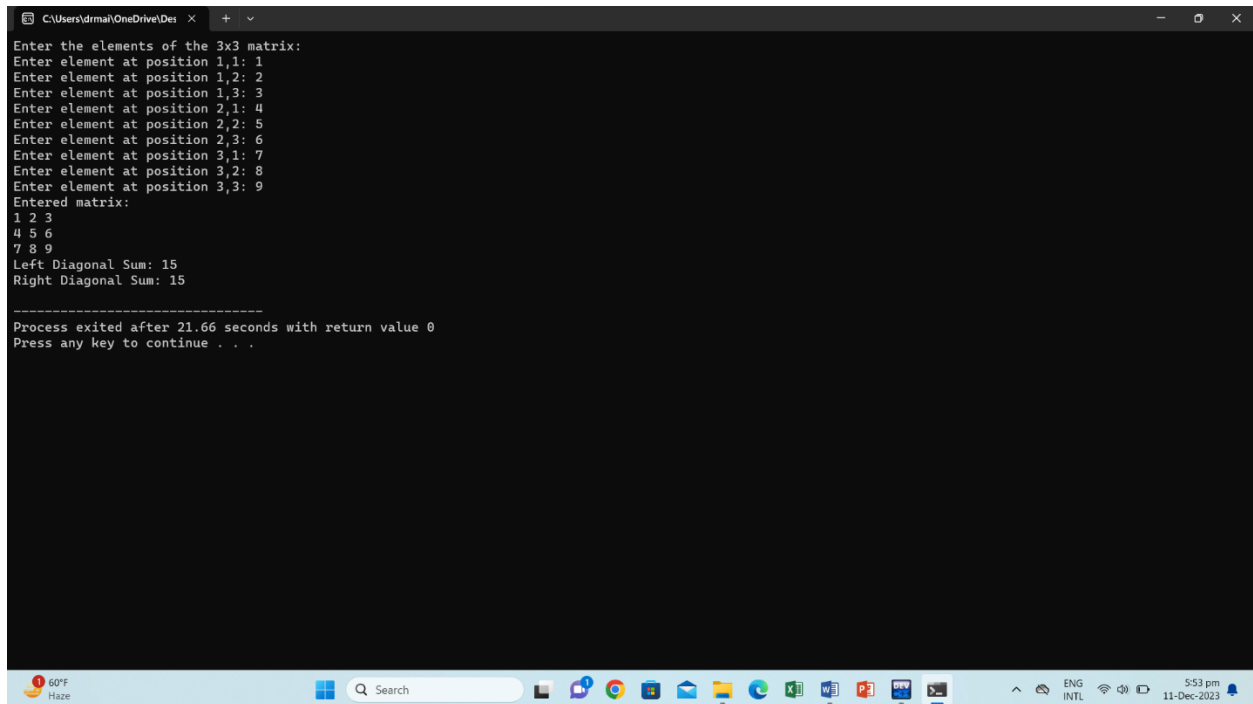
    cout << "Left Diagonal Sum: " << leftDiagonalSum << endl;

    cout << "Right Diagonal Sum: " << rightDiagonalSum << endl;

    return 0;
}

```

OUTPUT:



```

C:\Users\drmal\OneDrive\Des
Enter the elements of the 3x3 matrix:
Enter element at position 1,1: 1
Enter element at position 1,2: 2
Enter element at position 1,3: 3
Enter element at position 2,1: 4
Enter element at position 2,2: 5
Enter element at position 2,3: 6
Enter element at position 3,1: 7
Enter element at position 3,2: 8
Enter element at position 3,3: 9
Entered matrix:
1 2 3
4 5 6
7 8 9
Left Diagonal Sum: 15
Right Diagonal Sum: 15

-----
Process exited after 21.66 seconds with return value 0
Press any key to continue . . .

```

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

CODE:

```
#include <iostream>

using namespace std;

const int size = 3;

void addMatrices(int mat1[size][size], int mat2[size][size], int result[size][size]) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            result[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
}

int main() {
    int matrix1[size][size], matrix2[size][size], resultMatrix[size][size];

    cout << "Enter elements for the first 3x3 matrix:" << endl;
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            cout << "Enter element at position " << i + 1 << ", " << j + 1 << ": ";
            cin >> matrix1[i][j];
        }
    }

    cout << "Enter elements for the second 3x3 matrix:" << endl;
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
```

```

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

        cin >> matrix2[i][j];

    }

}

addMatrices(matrix1, matrix2, resultMatrix);

cout << "Resultant matrix after addition:" << endl;

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

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

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

    }

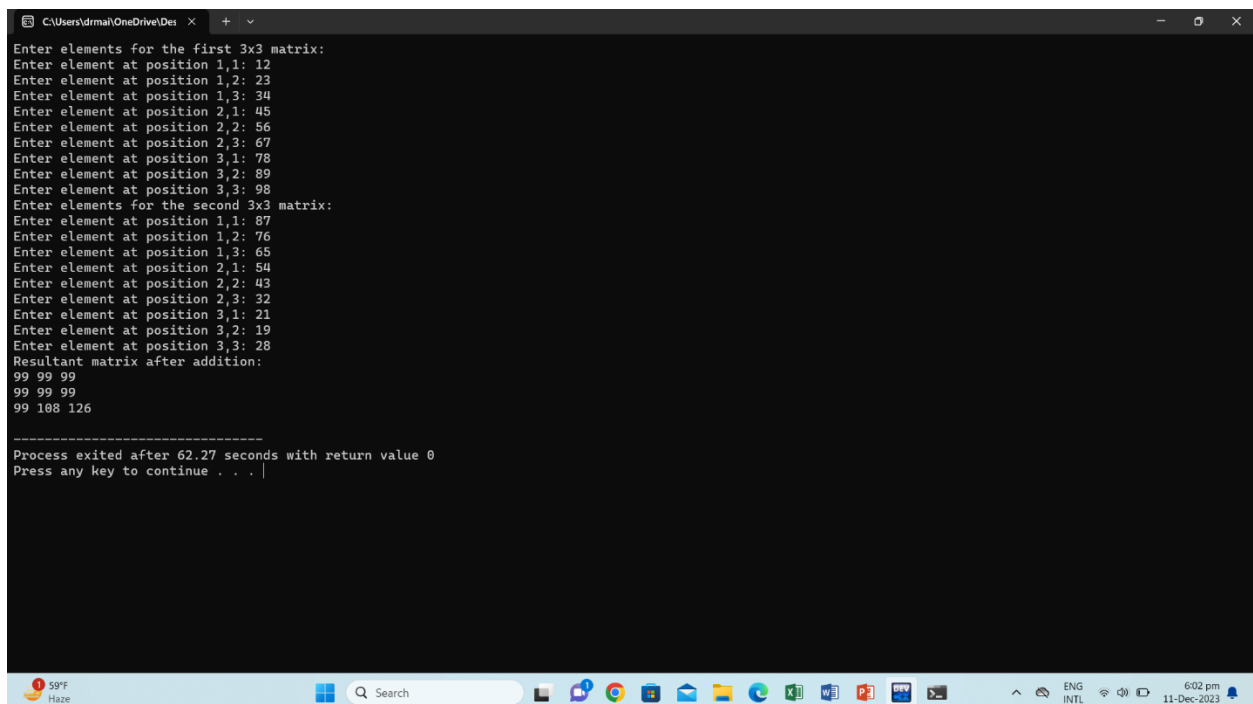
    cout << endl; }

return 0;

}

```

OUTPUT:



```

C:\Users\drrmal\OneDrive\Des
Enter elements for the first 3x3 matrix:
Enter element at position 1,1: 12
Enter element at position 1,2: 23
Enter element at position 1,3: 34
Enter element at position 2,1: 45
Enter element at position 2,2: 56
Enter element at position 2,3: 67
Enter element at position 3,1: 78
Enter element at position 3,2: 89
Enter element at position 3,3: 98
Enter elements for the second 3x3 matrix:
Enter element at position 1,1: 87
Enter element at position 1,2: 76
Enter element at position 1,3: 65
Enter element at position 2,1: 54
Enter element at position 2,2: 43
Enter element at position 2,3: 32
Enter element at position 3,1: 21
Enter element at position 3,2: 19
Enter element at position 3,3: 28
Resultant matrix after addition:
99 99 99
99 99 99
99 108 126

-----
Process exited after 62.27 seconds with return value 0
Press any key to continue . . .

```

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

CODE:

```
#include <iostream>

using namespace std;

const int size = 3;

void transposeMatrix(int original[size][size], int transposed[size][size]) {
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            transposed[j][i] = original[i][j];
        }
    }
}

int main() {
    int matrix[size][size], transposedMatrix[size][size];
    cout << "Enter elements for the 3x3 matrix:" << endl;
    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];
        }
    }

    transposeMatrix(matrix, transposedMatrix);
    cout << "Original matrix:" << endl;
```

```

for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
        cout << matrix[i][j] << " ";
    }
    cout << endl;
}

cout << "Transposed matrix:" << endl;
for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
        cout << transposedMatrix[i][j] << " ";
    }
    cout << endl;
}

return 0;
}

```

OUTPUT:

```

C:\Users\idmal\OneDrive\Desktop>
Enter elements for the 3x3 matrix:
Enter element at position 1,1: 1
Enter element at position 1,2: 2
Enter element at position 1,3: 3
Enter element at position 2,1: 4
Enter element at position 2,2: 5
Enter element at position 2,3: 6
Enter element at position 3,1: 7
Enter element at position 3,2: 8
Enter element at position 3,3: 9
Original matrix:
1 2 3
4 5 6
7 8 9
Transposed matrix:
1 4 7
2 5 8
3 6 9

-----
Process exited after 20.66 seconds with return value 0
Press any key to continue . . .

```

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

CODE:

```
#include <iostream>

using namespace std;

const int size = 3;

void multiplyMatrices(int mat1[size][size], int mat2[size][size], int
result[size][size]) {
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            result[i][j] = 0;
            for (int k = 0; k < size; ++k) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}

int main() {
    int matrix1[size][size], matrix2[size][size], resultMatrix[size][size];
    cout << "Enter elements for the first 3x3 matrix:" << endl;
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            cout << "Enter element at position " << i + 1 << ", " << j + 1 << ": ";
            cin >> matrix1[i][j]; }    }
```



```

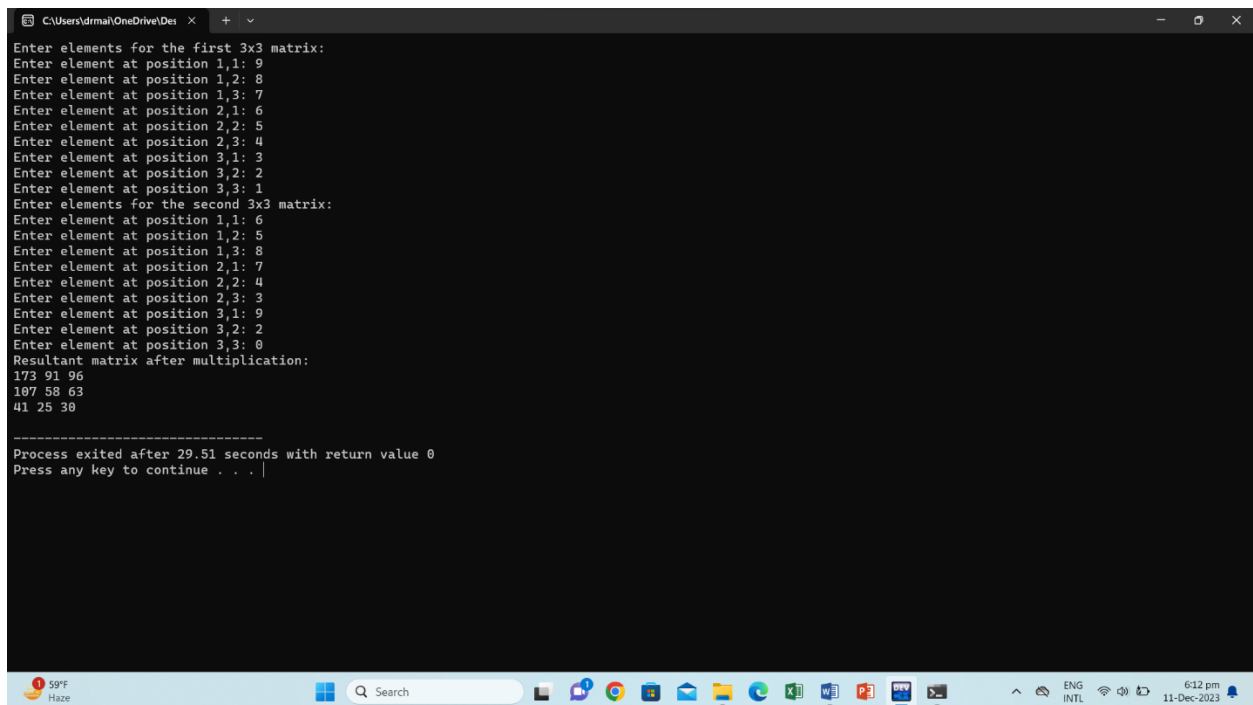
cout << "Enter elements for the second 3x3 matrix:" << endl;

for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
        cout << "Enter element at position " << i + 1 << ", " << j + 1 << ": ";
        cin >> matrix2[i][j];}    }
multiplyMatrices(matrix1, matrix2, resultMatrix);
cout << "Resultant matrix after multiplication:" << endl;
for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
        cout << resultMatrix[i][j] << " ";
    }
    cout << endl;}

return 0;}

```

OUTPUT:



```

C:\Users\dimal\OneDrive\Des  x  +  v
Enter elements for the first 3x3 matrix:
Enter element at position 1,1: 9
Enter element at position 1,2: 8
Enter element at position 1,3: 7
Enter element at position 2,1: 6
Enter element at position 2,2: 5
Enter element at position 2,3: 4
Enter element at position 3,1: 3
Enter element at position 3,2: 2
Enter element at position 3,3: 1
Enter elements for the second 3x3 matrix:
Enter element at position 1,1: 6
Enter element at position 1,2: 5
Enter element at position 1,3: 8
Enter element at position 2,1: 7
Enter element at position 2,2: 4
Enter element at position 2,3: 3
Enter element at position 3,1: 9
Enter element at position 3,2: 2
Enter element at position 3,3: 0
Resultant matrix after multiplication:
173 91 96
107 58 63
41 25 30

-----
Process exited after 29.51 seconds with return value 0
Press any key to continue . . .

```

Print the multiplication table of 15 using recursion

CODE:

```
#include <iostream>

using namespace std;

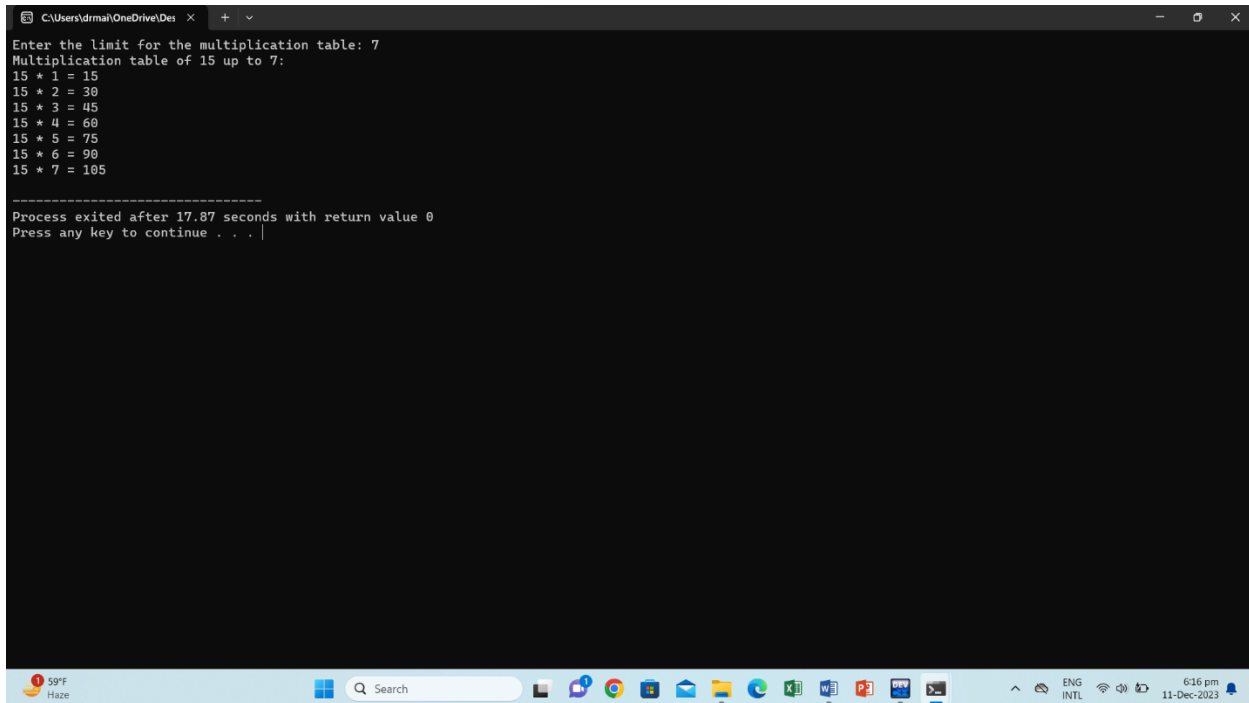
void printMultiplicationTable(int number, int limit, int current) {
    if (current > limit) {
        return;
    }
    cout << number << " * " << current << " = " << number * current << endl;
    printMultiplicationTable(number, limit, current + 1);
}

int main() {
    const int tableOf = 15;
    int limit;

    cout << "Enter the limit for the multiplication table: ";
    cin >> limit;

    cout << "Multiplication table of " << tableOf << " up to " << limit << ":" << endl;
    printMultiplicationTable(tableOf, limit, 1);
    return 0;
}
```

OUTPUT:



```
C:\Users\dramal\OneDrive\Desktop>
Enter the limit for the multiplication table: 7
Multiplication table of 15 up to 7:
15 * 1 = 15
15 * 2 = 30
15 * 3 = 45
15 * 4 = 60
15 * 5 = 75
15 * 6 = 90
15 * 7 = 105

-----
Process exited after 17.87 seconds with return value 0
Press any key to continue . . .
```

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

CODE:

```
#include <iostream>

using namespace std;

const int size = 3;

int determinant(int mat[size][size]) {
    return mat[0][0] * (mat[1][1] * mat[2][2] - mat[2][1] * mat[1][2]) -
           mat[0][1] * (mat[1][0] * mat[2][2] - mat[2][0] * mat[1][2]) +
           mat[0][2] * (mat[1][0] * mat[2][1] - mat[2][0] * mat[1][1]);
}

void transposeMatrix(int original[size][size], int transposed[size][size]) {
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
```

```

        transposed[j][i] = original[i][j];} } }
void cofactorMatrix(int mat[size][size], int cofactor[size][size]) {
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            int sign = ((i + j) % 2 == 0) ? 1 : -1;
            cofactor[i][j] = sign * determinant(mat[i][j]);} } }
void inverseMatrix(int mat[size][size], int inverse[size][size]) {
    int det = determinant(mat);
    if (det == 0) {
        cout << "The matrix is singular, and its inverse does not exist." << endl;
        return 0;}
    int cofactorMatrix[size][size];
    cofactorMatrix(mat, cofactorMatrix);
    int adjoint[size][size];
    transposeMatrix(cofactorMatrix, adjoint);
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            inverse[i][j] = adjoint[i][j] / det;} } }
void printMatrix(int mat[size][size]) {
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            cout << mat[i][j] << " ";
            cout << endl;}
    }
}
int main() {

```

```

int matrix[size][size], inverseMatrixResult[size][size];

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

    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];}    }

inverseMatrix(matrix, inverseMatrixResult);

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

printMatrix(matrix);

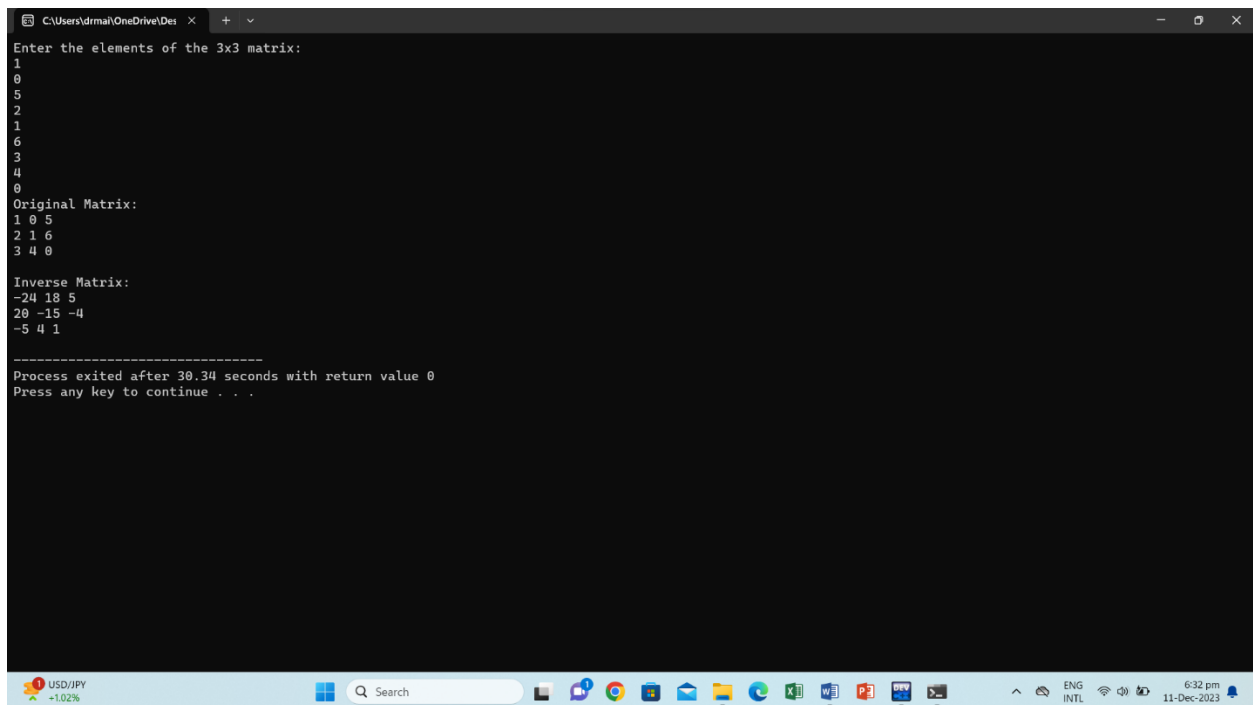
cout << "\nInverse matrix:" << endl;

printMatrix(inverseMatrixResult);

return 0;}

```

OUTPUT:



```

C:\Users\dimal\OneDrive\Des >
Enter the elements of the 3x3 matrix:
1
0
5
2
1
6
3
4
0
Original Matrix:
1 0 5
2 1 6
3 4 0
Inverse Matrix:
-24 18 5
20 -15 -4
-5 4 1
-----
Process exited after 30.34 seconds with return value 0
Press any key to continue . . .

```

