INTERNATIONAL MULTIDISCIPLINARY JOURNAL FOR RESEARCH & DEVELOPMENT
SJIF 2019: 5.222 2020: 5.552 2021: 5.637 2022:5.479 2023:6.563 2024: 7,805
eISSN :2394-6334 https://www.ijmrd.in/index.php/imjrd Volume 12, issue 05 (2025)

WORKING WITH MATRICES IN THE C++ PROGRAMMING LANGUAGE ENVIRONMENT .

Boritov Muzaffar Mansurovich
Kokand University, Digital Technologies and Mathematics department teacher
botirovmuzaffarmansurov@gmail.com

Abstract . The C++ programming language uses two-dimensional or multi-dimensional arrays to work with matrices. Matrix elements can be entered manually or by the user, displayed on the screen, and operations such as addition, multiplication, and transposition can be performed. Matrices are mainly used in scientific calculations, graphics, and image analysis. This thesis discusses the advantages of using matrices.
Keywords: addition, multiplication, transposition, symmetric, column, row .

A matrix is a rectangular table of numbers, algebraic expressions, or other elements, arranged in rows and columns. Each element is identified by its row and column indices. A matrix is usually written in square brackets or parentheses. For example, a 2×3 matrix has two rows and three columns. Matrices are widely used in mathematics, physics, computer science, and other sciences to model various processes, solve equations, process graphs, and create algorithms. Operations such as **addition** , **multiplication** , and **transposition** can be performed on a matrix.
Arrays are used to work with matrices in the C++ programming language. A matrix is a two-dimensional array consisting of rows and columns. The following are the basic methods for working with matrices and performing operations on them.
Creating a Matrix A matrix is created by declaring a two-dimensional array.

```
#include <iostream>
using namespace std;
int main() {
const int rows = 3; // Number of rows
const int cols = 3; // Number of columns
int matrix[rows][cols] = {
{1, 2, 3},
{4, 5, 6},
{7, 8, 9}
};
// Display the matrix
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
cout << matrix[i][j] << " ";
}
cout << endl;
}
return 0;
}
```

Ask the user to enter matrix elements and display it on the screen.

```
#include <iostream>
using namespace std;
int main() {
```

```
const int rows = 2;
const int cols = 3;
int matrix[rows][cols];
// Get matrix elements from the user
cout << "Enter the matrix elements:" << endl;
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         cout << "Element [" << i << "][" << j << "]: ";
         cin >> matrix[i][j];
      }
   }
   // Matritsani ekranga chiqarish
   cout << "Kiritilgan matritsa:" << endl;
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         cout << matrix[i][j] << " ";
      }
      cout << endl;
   }
   return 0;
}
```

To add matrices, they must be of the same size (have the same number of rows and columns). The addition process involves adding matching elements, that is, each element of the first matrix is added to the element of the second matrix at the same position. For example, if matrices A and B are given, then A + B = C, where C(i, j) = A(i, j) + B(i, j). If the matrices are not of the same size, the addition will not be performed. Adding matrices results in a new matrix whose size is equal to the size of the original matrices.

Adding the elements of two matrices of the same size.

```
#include <iostream>
using namespace std;
int main() {
const int rows = 2, cols = 2;
int matrix1[rows][cols] = {{1, 2}, {3, 4}};
   int matrix2[rows][cols] = {{5, 6}, {7, 8}};
   int result[rows][cols];
   // Qo‘shish
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         result[i][j] = matrix1[i][j] + matrix2[i][j];
      }
   }
   // Natijaviy matritsani chiqarish
   cout << "Natija (qo‘shilgan matritsa):" << endl;
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         cout << result[i][j] << " ";
      }
      cout << endl;
```

```
    }
    return 0;
}
```

To multiply matrices, the number of columns of the first matrix must be equal to the number of rows of the second matrix. If matrix A is m×n and matrix B is n×p , their product is matrix C of size m×p . Each element C(i,j) is found by multiplying the elements of row i of the first matrix by the elements of column j of the second matrix, respectively, and then calculating the sum.

$$C(i,j) = \sum_{k=1}^{n} A(i,k) * B(k,j).$$

Matrices multiplication is not commutative, i.e. A×B≠B×A
Multiplication according to the rules of matrix multiplication

```
#include <iostream>
using namespace std;
int main() {
    const int rows1 = 2, cols1 = 3, rows2 = 3, cols2 = 2;
    int matrix1[rows1][cols1] = {{1, 2, 3}, {4, 5, 6}};
    int matrix2[rows2][cols2] = {{7, 8}, {9, 10}, {11, 12}};
    int result[rows1][cols2] = {0};
    // Matritsalarni ko‘paytirish
    for (int i = 0; i < rows1; i++) {
        for (int j = 0; j < cols2; j++) {
            for (int k = 0; k < cols1; k++) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
    // Natijani chiqarish
    cout << "Ko‘paytirilgan matritsa:" << endl;
    for (int i = 0; i < rows1; i++) {
        for (int j = 0; j < cols2; j++) {
            cout << result[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

The transpose of a matrix is the process of converting the rows of a matrix into columns and the columns into rows. If the matrix A is of dimension m×n , then its transpose is an n×m matrix, which is A^T is denoted by . In the transposition process, each element of the matrix is replaced as follows: $A^T(i,j)=A(j,i)$, that is, the i -row goes to the j -column, and the j -column goes to the i -row. For example,

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ for $A^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$. Transposition is symmetric and plays an important role in many mathematical calculations.

Convert the rows of a matrix to columns.

```
#include <iostream>
```

```
using namespace std;
int main() {
const int rows = 2, cols = 3;
int matrix[rows][cols] = {{1, 2, 3}, {4, 5, 6}};
int transpose[cols][rows];
// Calculate the transposition
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
transpose[j][i] = matrix[i][j];
}
}
// Output the result
cout << "Transposed matrix:" << endl;
for (int i = 0; i < cols; i++) {
for (int j = 0; j < rows; j++) {
cout << transpose[i][j] << " ";
}
cout << endl;
}
return 0;
}
```

The above codes show different ways to work with matrices.
Test each code yourself and change the dimensions as needed.

References.

1. "C++ Primer" (Stanley B. Lippman, Josée Lajoie, Barbara E. Moo)
2. "Programming: Principles and Practice Using C++" (Bjarne Stroustrup)
3. "Data Structures and Algorithm Analysis in C++" (Mark Allen Weiss)
4. "Introduction to Algorithms" (Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein)