Metrix (2D Arrays ) in C++

**Objective:**

The objective of this experiment to get familiar with:

1. 2D Arrays (metrices) in C++.

2. Operations that can be performed on metrix in C++.

3. Implementation of OOP concepts (classes and functions).

Matrix multiplication is a fundamental operation in many areas, including computer graphics, physics simulations, and machine learning algorithms.

1. Write a program in c++ that performs binary search on a 2D matrix. Implement OOP concepts (make a class and define function for searching an element in an array).

Code:

#include <iostream>

#include <vector>

using namespace std;

class BinarySearch{

public:

    void search(vector<vector<int>>& matrix, int targetValue){

        int rows = matrix.size();

        int columns = matrix[0].size();

        int left = 0;

        int right = rows \* columns -1;

        while (left <= right) {

            int mid = left + (right - left)/2;

            int midValue = matrix[mid/rows][mid%rows];

            if (midValue == targetValue){

                cout << targetValue<< " is at "<<"Row: " << mid/rows + 1 << " Column: " << mid%rows + 1 << endl;

                return;

            }

            else if (midValue > targetValue) {

                right = mid - 1;

            }

            else {

                left = mid + 1;

            }

        }

        cout << targetValue <<" is not found in matrix." << endl;

    }

};

int main(){

    vector<vector<int>> Matrix = {{1, 2, 3},

                                  {4, 5, 6},

                                  {7, 8, 9}};

    BinarySearch m;

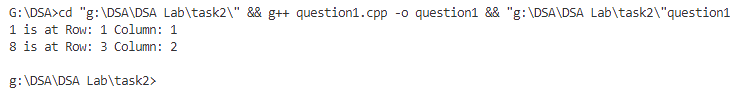
    m.search(Matrix, 1);

    m.search(Matrix, 8);

    return 0;

}

Output:



1. Write a program in C++ with a class Matrix that performs addition, subtraction, multiplication of two matrices. Multiplies a matrix with a constant and calculates transpose matrix.

Code:

#include <iostream>

using namespace std;

class Matrix {

private:

    int rows;

    int cols;

    int\*\* data;

public:

    Matrix(int r, int c) : rows(r), cols(c) {

        data = new int\*[rows];

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

            data[i] = new int[cols];

        }

    }

    ~Matrix() {

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

            delete[] data[i];

        }

        delete[] data;

    }

    void inputElements() {

        cout << "Enter elements of the matrix (" << rows << "x" << cols << "):" << endl;

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

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

                cin >> data[i][j];

            }

        }

    }

    void display() const {

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

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

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

            }

            cout << endl;

        }

    }

    Matrix add(const Matrix& B)  {

        Matrix result(rows, cols);

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

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

                result.data[i][j] = data[i][j] + B.data[i][j];

            }

        }

        return result;

    }

    Matrix subtract(const Matrix& B) {

        Matrix result(rows, cols);

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

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

                result.data[i][j] = data[i][j] - B.data[i][j];

            }

        }

        return result;

    }

    Matrix multiply(const Matrix& B) {

        Matrix result(rows, B.cols);

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

            for (int j = 0; j < B.cols; j++) {

                result.data[i][j] = 0;

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

                    result.data[i][j] += data[i][k] \* B.data[k][j];

                }

            }

        }

        return result;

    }

    Matrix multiplyByConstant(int constant) {

        Matrix result(rows, cols);

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

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

                result.data[i][j] = data[i][j] \* constant;

            }

        }

        return result;

    }

    Matrix transpose() {

        Matrix result(cols, rows);

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

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

                result.data[j][i] = data[i][j];

            }

        }

        return result;

    }

};

int main() {

    int rows, cols, constant;

    cout << "Enter the number of rows and columns for Matrix A: ";

    cin >> rows >> cols;

    Matrix A(rows, cols);

    A.inputElements();

    cout << "Enter the number of rows and columns for Matrix B: ";

    cin >> rows >> cols;

    Matrix B(rows, cols);

    B.inputElements();

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

    Matrix C = A.add(B);

    C.display();

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

    Matrix D = A.subtract(B);

    D.display();

    cout << "Matrix A \* Matrix B:" << endl;

    Matrix E = A.multiply(B);

    E.display();

    cout << "Enter a constant to multiply Matrix A: ";

    cin >> constant;

    cout << "Matrix A \* " << constant << ":" << endl;

    Matrix F = A.multiplyByConstant(constant);

    F.display();

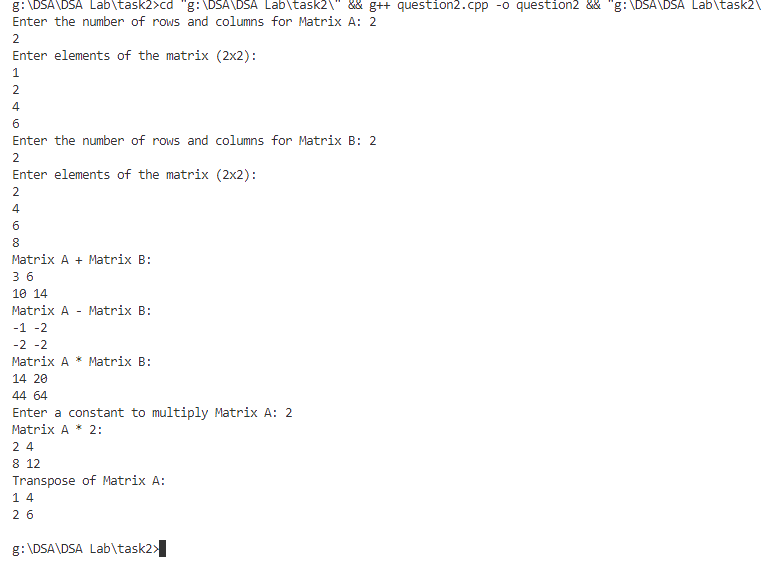
    cout << "Transpose of Matrix A:" << endl;

    Matrix G = A.transpose();

    G.display();

    return 0;

}

Output

1. Suppose A and B are n-elements vector array in memory and X and Y are scalars.

Write a program to find.

* + XA + YB
  + A . B

Test the program using A= (16, -6,7), B=(4,2,-3), X= 2, Y= -5

Code:

#include <iostream>

#include <vector>

using namespace std;

int scalarProduct(int a[], int b[], int size){

    int result;

    for (int i=0; i<size; i++){

        result += a[i] \* b[i];

    }

    return result;

}

void vectorAddition(int a[], int b[], int x, int y, int size){

    int result[size];

    for (int i = 0; i < size; i++){

        result[i] = x\*a[i] + y\*b[i];

    }

    cout << "XA + YB = (";

    for (int i=0; i<size; i++){

        cout << result[i];

        if (i != size - 1){

            cout << ", ";

        }

    }

    cout << ")" << endl;

}

int main(){

    int A[] = {16, -6, 7};

    int B[] = {4, 2, -3};

    int vectorSize =sizeof(A)/sizeof(A[0]) ;

    int X = 2;

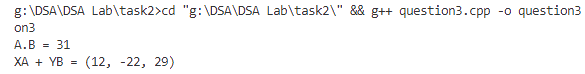
    int Y = -5;

    cout << "A.B = " << scalarProduct(A, B, vectorSize) << endl;

    vectorAddition(A, B, X, Y, vectorSize);

    return 0;}

Output:



|  |  |  |
| --- | --- | --- |
| **Lab 02 Evaluation** | | |
| **Student Name: Adil Javed Student ID:SESE-23025 Date: 08-Sep-2024** | | |
| **Rubric** | **Marks (25)** | **Remarks by teacher in accordance with the rubrics** |
| **R1** |  |  |
| **R2** |  |  |
| **R3** |  |  |
| **R4** |  |  |
| **R5** |  |  |