Object oriented programming in c++

(CIC-257)

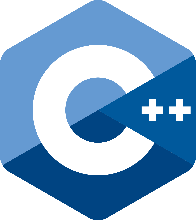
Saksham Gupta

CSE 3

12415002721

Submitted To:

Mr. Vikrant Shokeen



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No.** | **Name Of Experiment** | **Date** | **Pg. No.** | **Remarks** |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No.** | **Name Of Experiment** | **Date** | **Pg. No.** | **Remarks** |
|  |  |  |  |  |

**Program No 1: Multiplication of two matrices using OOP**

**THEORY**

**MULTIDIMENSIONAL ARRAYS**

A multi-dimensional array can be termed as an array of arrays that stores homogeneous data in tabular form. Data in multidimensional arrays are stored in row-major order.

The***general form of declaring N-dimensional arrays*** is:

data\_type array\_name[size1][size2]....[sizeN];

* **data\_type**: Type of data to be stored in the array.
* **array\_name**: Name of the array
* **size1, size2,… ,sizeN**: Sizes of the dimension

**TWO DIMENSIONAL ARRAYS (MATRIX)**

Two – dimensional array is the simplest form of a multidimensional array. We can see a two – dimensional array as an array of one-dimensional array for easier understanding.

The basic form of declaring a two-dimensional array of size x, y:   
**Syntax:**

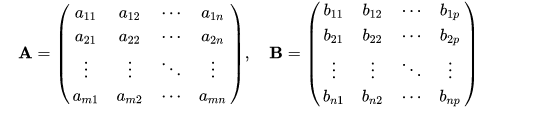
**data\_type array\_name[x][y];**

We can declare a two-dimensional integer array say ‘x’ of size 10,20 as:

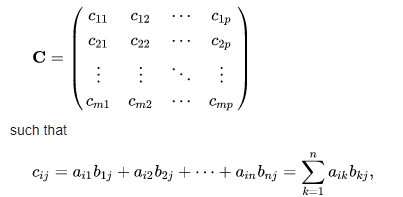
int x[10][20];

**MATRIX MULTIPLICATION**

If **A** is an *m* × *n* matrix and **B** is an *n* × *p* matrix,



the *matrix product* **C** = **AB** (denoted without multiplication signs or dots) is defined to be the *m* × *p* matrix



for *i* = 1, ..., *m* and *j* = 1, ..., *p*.

**SOURCE CODE**

// Header Files

#include <iostream>

// Prototyping the functions

int \*input2D\_matrix1D(int \*, int \*);

int \*AxB(int \*, int \*, int, int, int);

void print2Dmatrix(int \*, int, int);

int main()

{

    // Getting the two matrices A and B inputted

    // A1D namins is used becuase its a 2D array A but in 1D form

    int ARows, ACols, BRows, BCols;

    std::cout << "Input data for Matrix A:" << std::endl;

    int \*A1D = input2D\_matrix1D(&ARows, &ACols);

    std::cout << "\nInput data for Matrix B:" << std::endl;

    int \*B1D = input2D\_matrix1D(&BRows, &BCols);

    // Checking if the matrices are valid for being multiplied

    if (ACols != BRows)

    {

        std::cout << "Inputted matrix cant be multiplied.";

        return 0;

    }

    // Multiplication using function

    int \*C1D = AxB(A1D, B1D, ARows, ACols, BCols);

    // Printing using function

    print2Dmatrix(C1D, ARows, BCols);

    return 0;

}

int \*AxB(int \*A, int \*B, int ARows, int ACols, int BCols)

{

    // Creating a 2D array with size ARows X BCols

    int(\*C)[ARows] = (int(\*)[ARows]) new int[ARows \* BCols];

    // Matrix multiplication logic

    for (int i = 0; i < ARows; i++)

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

        {

            C[i][j] = 0;

            for (int k = 0; k < ACols; k++)

                C[i][j] += \*(A + ACols \* i + k) \* \*(B + ACols \* k + j);

        }

    return (int \*)C;

}

int \*input2D\_matrix1D(int \*rows, int \*columns)

{

    // This function takes input from the user and creates an array from that

    printf("Enter the number of Rows: ");

    scanf("%d", rows); // It also returns the value back because pointer

    printf("Enter the number of Columns: ");

    scanf("%d", columns);

    // Creates an array with length as if the 2D matrix was converted to 1D row wise

    // This is done since you cannot pass in function and return a variable length multidimensional array

    int \*matrix = new int[(\*rows) \* (\*columns)];

    for (int i = 0; i < \*rows; i++)

        for (int j = 0; j < \*columns; j++)

        {

            printf("Enter the number at position (%d,%d): ", i, j);

            scanf("%d", matrix + i \* (\*columns) + j);

        }

    return matrix;

}

void print2Dmatrix(int \*A1D, int rows, int columns)

{

    // Converting 1D array into 2D array using type casting

    int(\*A)[rows] = (int(\*)[rows])A1D;

    std::cout << "\nMatrix C = AxB:\n";

    for (int i = 0; i < rows; i++)

    {

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

            std::cout << "\t" << A[i][j];

        std::cout << "\n";

    }

}

**OUTPUT**

* Test case 1: Where matrices can be multiplied, ie: Columns of A = Rows of B

PS C:\Users\admin\Documents\Saksham Gupta\MSIT> ./a.exe

Input data for Matrix A:

Enter the number of Rows: 2

Enter the number of Columns: 3

Enter the number at position (0,0): 2

Enter the number at position (0,1): 4

Enter the number at position (0,2): 1

Enter the number at position (1,0): 2

Enter the number at position (1,1): 5

Enter the number at position (1,2): 3

Input data for Matrix B:

Enter the number of Rows: 3

Enter the number of Columns: 1

Enter the number at position (0,0): 5

Enter the number at position (1,0): 6

Enter the number at position (2,0): 3

Matrix C = AxB:

20

70

* Test case 2: Where matrices can’t be multiplied, ie: Columns of A != Rows of B

PS C:\Users\admin\Documents\Saksham Gupta\MSIT> ./a.exe

Input data for Matrix A:

Enter the number of Rows: 1

Enter the number of Columns: 2

Enter the number at position (0,0): 4

Enter the number at position (0,1): 5

Input data for Matrix B:

Enter the number of Rows: 4

Enter the number of Columns: 1

Enter the number at position (0,0): 2

Enter the number at position (1,0): 3

Enter the number at position (2,0): 4

Enter the number at position (3,0): 5

Inputted matrix cant be multiplied.