Лабораторная работа № 4

**Тема:** программирование задач с матрицами.

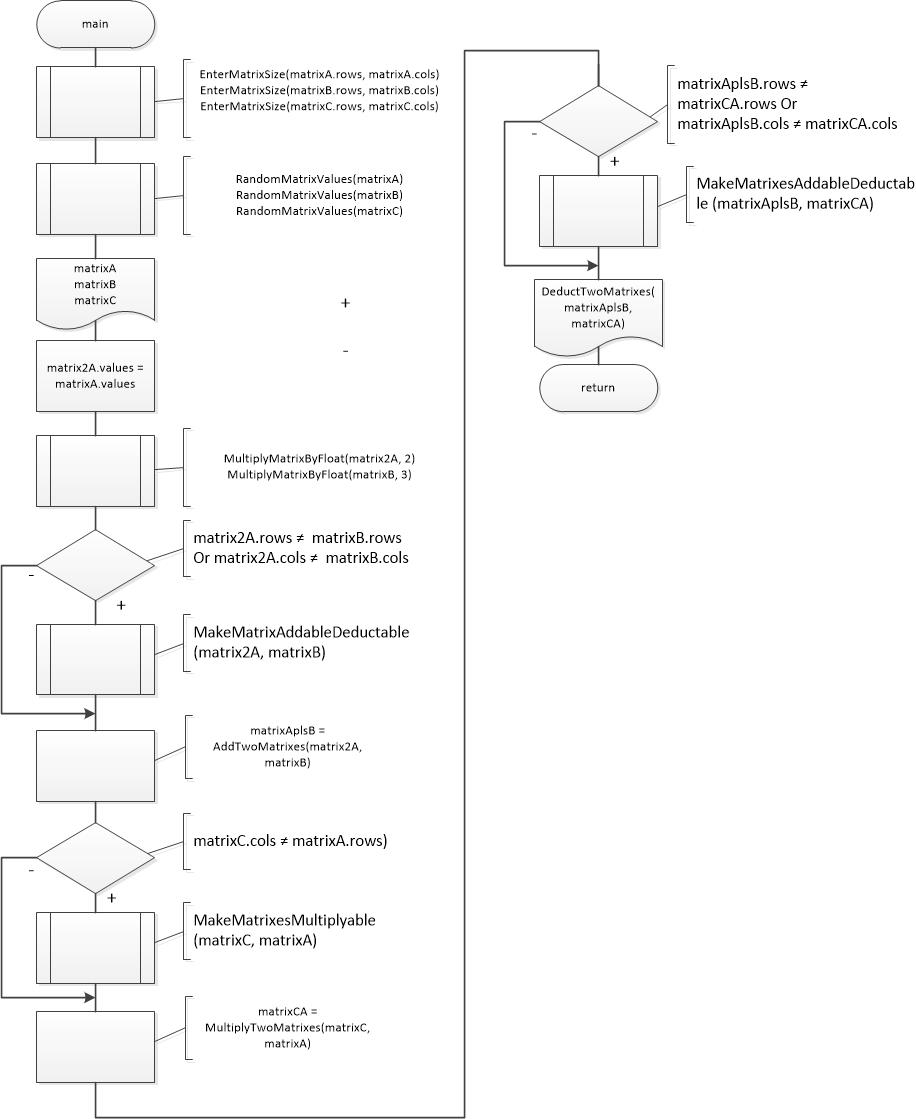
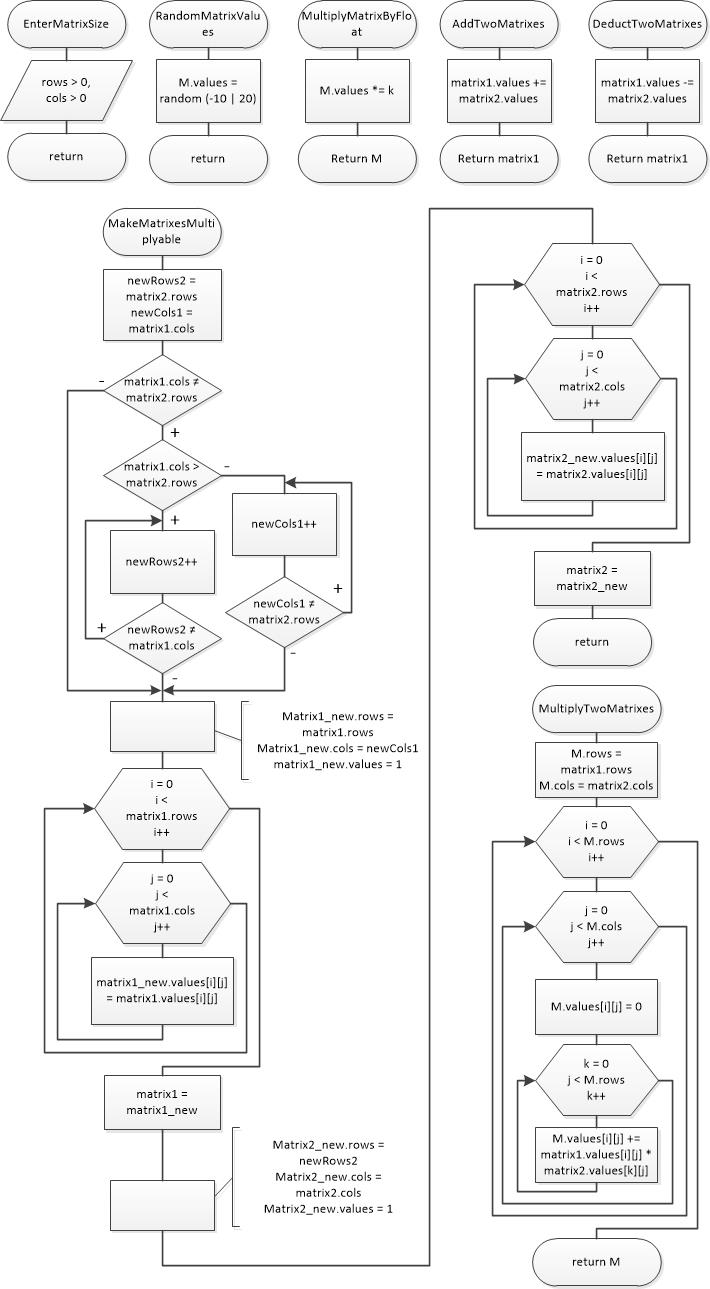
**Цель работы:** получить практические навыки программирования задач с матрицами.

**Задача:** найти значение выражения, где A, B и C - матрицы, V - вектор строка и W - вектор столбец. Размерность матриц и векторов должна вводить с клавиатуры (любые значения). В программе должна быть реализована проверка на возможность проведения необходимых операций, если размерность операндов не позволяет провести заданную операцию, то матрицы или вектора нужно увеличить, добавив соответствующее количество единиц. Выводить на печать промежуточные и конечные результаты.

**Индивидуальное задание:**

|  |  |
| --- | --- |
| № Варианта | Выражение |
| 10 | 2A + 3B - CA |

**Блок-схема:**



**Листинг программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

struct matrix { // матрица

int rows;

int cols;

float\*\* values;

};

float\*\* AllocMemoryForMatrix(int rows, int cols) // выделение памяти под матрицу

{

float\*\* matrix = new float\* [rows];

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

matrix[i] = new float[cols];

return matrix;

}

void EnterMatrixSize(int\* rows, int\* cols, char matrixName) // ввод размеров матрицы

{

do {

printf(" Enter the size of matrix '%c': ", matrixName);

scanf("%i%i", rows, cols);

} while (\*rows < 1 && \*cols < 1);

}

void FreeMatrixMemory(matrix\* M) // очистка памяти матрицы

{

for (int i = 0; i < M->rows; i++)

delete[] M->values[i];

delete[] M->values;

}

void RandomMatrixValues(matrix\* M) // рандом значений матрицы

{

for (int i = 0; i < M->rows; i++)

for (int j = 0; j < M->cols; j++)

M->values[i][j] = -10.0 + 20.0 \* rand() / RAND\_MAX;

}

void PrintMatrix(matrix M) // печать матрицы

{

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

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

printf("%8.1f", M.values[i][j]);

printf("\n");

}

}

void MakeMatrixesMultiplyable(matrix\* matrix1, matrix\* matrix2) // сделать матрицы умножаемыми

{

int newRows2 = matrix2->rows, newCols1 = matrix1->cols; // будущие новые размеры матриц

if (matrix1->cols != matrix2->rows) // подгоняем размеры матриц

if (matrix1->cols > matrix2->rows)

do {

newRows2++;

} while (newRows2 != matrix1->cols);

else

do {

newCols1++;

} while (newCols1 != matrix2->rows);

//----------------------------------------------------------------------------------------------

matrix matrix1\_new; // создаём дублёр 1й матрицы

matrix1\_new.rows = matrix1->rows;

matrix1\_new.cols = newCols1;

matrix1\_new.values = AllocMemoryForMatrix(matrix1\_new.rows, matrix1\_new.cols);

for (int i = 0; i < matrix1\_new.rows; i++) // приводим новую матрицу к единицам

for (int j = 0; j < matrix1\_new.cols; j++)

matrix1\_new.values[i][j] = 1;

for (int i = 0; i < matrix1->rows; i++) // сохраняем значения из старой матрицы

for (int j = 0; j < matrix1->cols; j++)

matrix1\_new.values[i][j] = matrix1->values[i][j];

FreeMatrixMemory(matrix1); // перевыделяем память под исходную матрицу

matrix1->values = AllocMemoryForMatrix(matrix1\_new.rows, matrix1\_new.cols);

matrix1->rows = matrix1\_new.rows;

matrix1->cols = matrix1\_new.cols;

for (int i = 0; i < matrix1->rows; i++) // переносим сохраненные значения в исходную матрицу

for (int j = 0; j < matrix1->cols; j++)

matrix1->values[i][j] = matrix1\_new.values[i][j];

FreeMatrixMemory(&matrix1\_new);

// ------------------------------------------------------------

matrix matrix2\_new; // создаём дублёр 2й матрицы

matrix2\_new.rows = newRows2;

matrix2\_new.cols = matrix2->cols;

matrix2\_new.values = AllocMemoryForMatrix(matrix2\_new.rows, matrix2\_new.cols);

for (int i = 0; i < matrix2\_new.rows; i++) // приводим новую матрицу к единицам

for (int j = 0; j < matrix2\_new.cols; j++)

matrix2\_new.values[i][j] = 1;

for (int i = 0; i < matrix2->rows; i++) // сохраняем значения из старой матрицы

for (int j = 0; j < matrix2->cols; j++)

matrix2\_new.values[i][j] = matrix2->values[i][j];

FreeMatrixMemory(matrix2); // перевыделяем память под исходную матрицу

matrix2->values = AllocMemoryForMatrix(matrix2\_new.rows, matrix2\_new.cols);

matrix2->rows = matrix2\_new.rows;

matrix2->cols = matrix2\_new.cols;

for (int i = 0; i < matrix2->rows; i++) // переносим сохраненные значения в исходную матрицу

for (int j = 0; j < matrix2->cols; j++)

matrix2->values[i][j] = matrix2\_new.values[i][j];

FreeMatrixMemory(&matrix2\_new);

}

matrix MultiplyTwoMatrixes(matrix matrix1, matrix matrix2)

{

matrix M = {};

M.rows = matrix1.rows;

M.cols = matrix2.cols;

M.values = AllocMemoryForMatrix(M.rows, M.cols);

for (int i = 0; i < M.rows; i++)

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

M.values[i][j] = 0;

for (int k = 0; k < M.rows; k++)

M.values[i][j] += matrix1.values[i][k] \* matrix2.values[k][j];

}

return M;

}

matrix\* MultiplyMatrixByFloat(matrix\* M, float k)

{

for (int i = 0; i < M->rows; i++)

for (int j = 0; j < M->cols; j++)

M->values[i][j] \*= k;

return M;

}

void MakeMatrixesAddableDeductable(matrix\* matrix1, matrix\* matrix2)

{

int newRows1 = matrix1->rows, newCols1 = matrix1->cols, newRows2 = matrix2->rows, newCols2 = matrix2->cols; // будущие новые размеры матриц

if (matrix1->rows != matrix2->rows) // увеличиваем строки

if (matrix1->rows > matrix2->rows)

do {

newRows2++;

} while (newRows2 != matrix1->rows);

else

do {

newRows1++;

} while (newRows1 != matrix2->rows);

if (matrix1->cols != matrix2->cols) // увеличиваем столбцы

if (matrix1->cols > matrix2->cols)

do {

newCols2++;

} while (newCols2 != matrix1->cols);

else

do {

newCols1++;

} while (newCols1 != matrix2->cols);

//----------------------------------------------------------------------------------------------------

matrix matrix1\_new; // создаём дублёр 1й матрицы

matrix1\_new.rows = newRows1;

matrix1\_new.cols = newCols1;

matrix1\_new.values = AllocMemoryForMatrix(matrix1\_new.rows, matrix1\_new.cols);

for (int i = 0; i < matrix1\_new.rows; i++) // приводим новую матрицу к единицам

for (int j = 0; j < matrix1\_new.cols; j++)

matrix1\_new.values[i][j] = 1;

for (int i = 0; i < matrix1->rows; i++) // сохраняем значения из старой матрицы

for (int j = 0; j < matrix1->cols; j++)

matrix1\_new.values[i][j] = matrix1->values[i][j];

FreeMatrixMemory(matrix1); // перевыделяем память под исходную матрицу

matrix1->values = AllocMemoryForMatrix(matrix1\_new.rows, matrix1\_new.cols);

matrix1->rows = matrix1\_new.rows;

matrix1->cols = matrix1\_new.cols;

for (int i = 0; i < matrix1->rows; i++) // переносим сохраненные значения в исходную матрицу

for (int j = 0; j < matrix1->cols; j++)

matrix1->values[i][j] = matrix1\_new.values[i][j];

FreeMatrixMemory(&matrix1\_new);

// ---------------------------------------------------------------------

matrix matrix2\_new; // создаём дублёр 2й матрицы

matrix2\_new.rows = newRows2;

matrix2\_new.cols = newCols2;

matrix2\_new.values = AllocMemoryForMatrix(matrix2\_new.rows, matrix2\_new.cols);

for (int i = 0; i < matrix2\_new.rows; i++) // приводим матрицу к единицам

for (int j = 0; j < matrix2\_new.cols; j++)

matrix2\_new.values[i][j] = 1;

for (int i = 0; i < matrix2->rows; i++) // переносим значения из старой матрицы

for (int j = 0; j < matrix2->cols; j++)

matrix2\_new.values[i][j] = matrix2->values[i][j];

FreeMatrixMemory(matrix2); // перевыделяем память под исходную матрицу

matrix2->values = AllocMemoryForMatrix(matrix2\_new.rows, matrix2\_new.cols);

matrix2->rows = matrix2\_new.rows;

matrix2->cols = matrix2\_new.cols;

for (int i = 0; i < matrix2->rows; i++) // переносим сохраненные значения в исходную матрицу

for (int j = 0; j < matrix2->cols; j++)

matrix2->values[i][j] = matrix2\_new.values[i][j];

FreeMatrixMemory(&matrix2\_new);

}

matrix AddTwoMatrixes(matrix matrix1, matrix matrix2)

{

for (int i = 0; i < matrix1.rows; i++)

for (int j = 0; j < matrix1.cols; j++)

matrix1.values[i][j] += matrix2.values[i][j];

return matrix1;

}

matrix DeductTwoMatrixes(matrix matrix1, matrix matrix2)

{

for (int i = 0; i < matrix1.rows; i++)

for (int j = 0; j < matrix1.cols; j++)

matrix1.values[i][j] -= matrix2.values[i][j];

return matrix1;

}

int main()

{

matrix matrixA, matrixB, matrixC, matrixAplsB, matrixCA; // объявление матриц

EnterMatrixSize(&matrixA.rows, &matrixA.cols, 'A'); // ввод размеров матриц

EnterMatrixSize(&matrixB.rows, &matrixB.cols, 'B');

EnterMatrixSize(&matrixC.rows, &matrixC.cols, 'C');

matrixA.values = AllocMemoryForMatrix(matrixA.rows, matrixA.cols), // выделение памяти под матрицы

matrixB.values = AllocMemoryForMatrix(matrixB.rows, matrixB.cols),

matrixC.values = AllocMemoryForMatrix(matrixC.rows, matrixC.cols);

RandomMatrixValues(&matrixA); // рандомим элементы матриц

RandomMatrixValues(&matrixB);

RandomMatrixValues(&matrixC);

printf("\nMatrix A: \n");

PrintMatrix(matrixA); // выводим матрицы на экран

printf("\nMatrix B: \n");

PrintMatrix(matrixB);

printf("\nMatrix C: \n");

PrintMatrix(matrixC);

matrix matrix2A; // создаём 2А

matrix2A.rows = matrixA.rows;

matrix2A.cols = matrixA.cols;

matrix2A.values = AllocMemoryForMatrix(matrix2A.rows, matrix2A.cols);

for (int i = 0; i < matrixA.rows; i++) // копируем матрицу А

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

matrix2A.values[i][j] = matrixA.values[i][j];

MultiplyMatrixByFloat(&matrix2A, 2.0); // 2A

printf("\nMatrix 2A: \n");

PrintMatrix(matrix2A);

MultiplyMatrixByFloat(&matrixB, 3.0); // 3B

printf("\nMatrix 3B: \n");

PrintMatrix(matrixB);

if (matrix2A.rows != matrixB.rows || matrix2A.cols != matrixB.cols) {

MakeMatrixesAddableDeductable(&matrix2A, &matrixB); // prepare for 2A + 3B

printf("\n Matrixes 2A & 3B are now addable. \n");

printf("\nMatrix 2A: \n");

PrintMatrix(matrix2A);

printf("\nMatrix 3B: \n");

PrintMatrix(matrixB);

}

matrixAplsB = AddTwoMatrixes(matrix2A, matrixB); // 2A + 3B;

printf("\nMatrix 2A + 3B: \n");

PrintMatrix(matrixAplsB);

if (matrixC.cols != matrixA.rows) {

MakeMatrixesMultiplyable(&matrixC, &matrixA); // prepare for CA

printf("\n Matrixes C & A are now multiplyable. \n");

printf("\nMatrix new C: \n");

PrintMatrix(matrixC);

printf("\nMatrix new A: \n");

PrintMatrix(matrixA);

}

matrixCA = MultiplyTwoMatrixes(matrixC, matrixA); // CA;

printf("\nMatrix CA: \n");

PrintMatrix(matrixCA);

if (matrixAplsB.rows != matrixCA.rows || matrixAplsB.cols != matrixCA.cols) {

MakeMatrixesAddableDeductable(&matrixAplsB, &matrixCA); // prepare for 2A + 3B - CA

printf("\n Matrixes 2A + 3B & CA are now deductable \n");

printf("\nMatrix new 2A + 3B: \n");

PrintMatrix(matrixAplsB);

printf("\nMatrix new CA: \n");

PrintMatrix(matrixCA);

}

printf("\nMatrix 2A + 3B - CA: \n");

PrintMatrix(DeductTwoMatrixes(matrixAplsB, matrixCA));

FreeMatrixMemory(&matrixA); // удаляем матрицы

FreeMatrixMemory(&matrixB);

FreeMatrixMemory(&matrixC);

printf("\n\n");

system("pause");

return 0;

}

**Результат:**

Enter the size of matrix 'A': 5 5

Enter the size of matrix 'B': 6 6

Enter the size of matrix 'C': 7 7

Matrix A:

-10.0 1.3 -6.1 6.2 1.7

-0.4 -3.0 7.9 6.5 4.9

-6.5 7.2 4.2 0.3 -3.9

-9.7 -8.2 -2.7 -7.1 -6.7

9.8 -1.1 -7.6 -9.9 -9.8

Matrix B:

-2.4 0.6 1.4 2.0 2.1 -6.7

3.3 -1.0 -3.0 -8.9 2.2 5.7

6.1 0.4 -4.0 7.5 4.5 9.1

8.5 0.8 -7.2 -0.8 -5.3 7.2

-5.8 5.6 6.9 9.9 10.0 2.2

-2.2 -4.7 -4.1 6.8 -9.5 -2.5

Matrix C:

-8.1 3.5 -8.9 -9.8 8.4 -4.5 -4.5

1.8 3.8 6.8 4.5 -0.3 -5.9 4.9

-0.6 -0.8 9.0 4.9 -7.8 2.0 -2.3

4.7 2.2 1.4 -2.8 -7.0 -5.5 -1.5

6.1 0.3 9.8 5.0 -3.1 -6.6 3.1

-0.2 -8.7 4.0 0.1 -7.1 9.0 -7.2

8.1 3.9 -3.9 -1.5 -8.6 9.3 3.7

Matrix 2A:

-19.9 2.5 -12.3 12.3 3.4

-0.8 -6.0 15.8 12.9 9.9

-13.0 14.4 8.4 0.5 -7.8

-19.4 -16.3 -5.4 -14.1 -13.4

19.5 -2.2 -15.2 -19.8 -19.6

Matrix 3B:

-7.3 1.9 4.3 6.1 6.4 -20.0

9.8 -3.0 -8.9 -26.6 6.5 17.0

18.2 1.2 -11.9 22.6 13.6 27.4

25.5 2.4 -21.5 -2.3 -15.9 21.7

-17.4 16.8 20.6 29.8 30.0 6.7

-6.5 -14.0 -12.2 20.4 -28.6 -7.4

Matrixes 2A & 3B are now addable.

Matrix 2A:

-19.9 2.5 -12.3 12.3 3.4 1.0

-0.8 -6.0 15.8 12.9 9.9 1.0

-13.0 14.4 8.4 0.5 -7.8 1.0

-19.4 -16.3 -5.4 -14.1 -13.4 1.0

19.5 -2.2 -15.2 -19.8 -19.6 1.0

1.0 1.0 1.0 1.0 1.0 1.0

Matrix 3B:

-7.3 1.9 4.3 6.1 6.4 -20.0

9.8 -3.0 -8.9 -26.6 6.5 17.0

18.2 1.2 -11.9 22.6 13.6 27.4

25.5 2.4 -21.5 -2.3 -15.9 21.7

-17.4 16.8 20.6 29.8 30.0 6.7

-6.5 -14.0 -12.2 20.4 -28.6 -7.4

Matrix 2A + 3B:

-27.3 4.4 -8.0 18.5 9.8 -19.0

9.0 -8.9 7.0 -13.7 16.3 18.0

5.1 15.6 -3.5 23.1 5.8 28.4

6.1 -14.0 -26.9 -16.4 -29.2 22.7

2.1 14.6 5.4 10.0 10.3 7.7

-5.5 -13.0 -11.2 21.4 -27.6 -6.4

Matrixes C & A are now multiplyable.

Matrix new C:

-8.1 3.5 -8.9 -9.8 8.4 -4.5 -4.5

1.8 3.8 6.8 4.5 -0.3 -5.9 4.9

-0.6 -0.8 9.0 4.9 -7.8 2.0 -2.3

4.7 2.2 1.4 -2.8 -7.0 -5.5 -1.5

6.1 0.3 9.8 5.0 -3.1 -6.6 3.1

-0.2 -8.7 4.0 0.1 -7.1 9.0 -7.2

8.1 3.9 -3.9 -1.5 -8.6 9.3 3.7

Matrix new A:

-10.0 1.3 -6.1 6.2 1.7

-0.4 -3.0 7.9 6.5 4.9

-6.5 7.2 4.2 0.3 -3.9

-9.7 -8.2 -2.7 -7.1 -6.7

9.8 -1.1 -7.6 -9.9 -9.8

1.0 1.0 1.0 1.0 1.0

1.0 1.0 1.0 1.0 1.0

Matrix CA:

305.8 -22.5 -5.5 -52.5 12.8

-111.0 1.6 36.9 7.4 -32.9

-176.2 34.4 81.1 35.9 3.5

-105.4 33.1 48.1 125.1 93.0

-206.9 35.8 13.2 33.9 -33.2

-88.9 63.3 4.0 14.7 11.4

-113.4 4.8 46.7 182.4 155.4

Matrixes 2A + 3B & CA are now deductable

Matrix new 2A + 3B:

-27.3 4.4 -8.0 18.5 9.8 -19.0

9.0 -8.9 7.0 -13.7 16.3 18.0

5.1 15.6 -3.5 23.1 5.8 28.4

6.1 -14.0 -26.9 -16.4 -29.2 22.7

2.1 14.6 5.4 10.0 10.3 7.7

-5.5 -13.0 -11.2 21.4 -27.6 -6.4

1.0 1.0 1.0 1.0 1.0 1.0

Matrix new CA:

305.8 -22.5 -5.5 -52.5 12.8 1.0

-111.0 1.6 36.9 7.4 -32.9 1.0

-176.2 34.4 81.1 35.9 3.5 1.0

-105.4 33.1 48.1 125.1 93.0 1.0

-206.9 35.8 13.2 33.9 -33.2 1.0

-88.9 63.3 4.0 14.7 11.4 1.0

-113.4 4.8 46.7 182.4 155.4 1.0

Matrix 2A + 3B - CA:

-333.1 27.0 -2.5 71.0 -2.9 -20.0

120.0 -10.5 -30.0 -21.1 49.3 17.0

181.3 -18.9 -84.6 -12.8 2.2 27.4

111.5 -47.1 -75.0 -141.5 -122.3 21.7

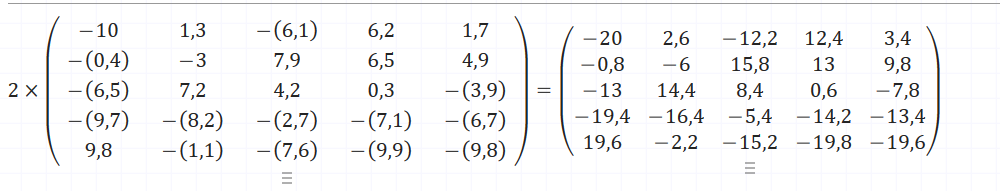
209.0 -21.2 -7.8 -23.9 43.5 6.7

83.4 -76.3 -15.1 6.7 -39.0 -7.4

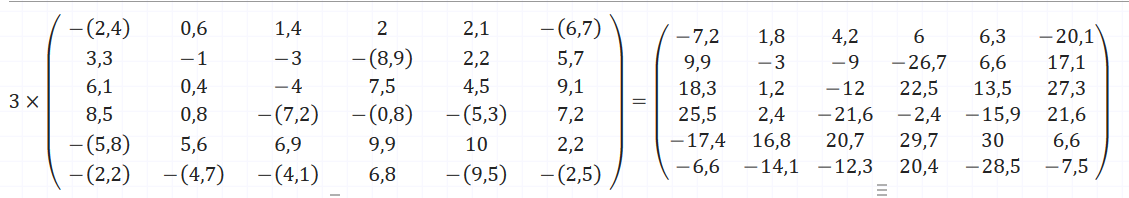
114.4 -3.8 -45.7 -181.4 -154.4 0.0

**Вычисления:**

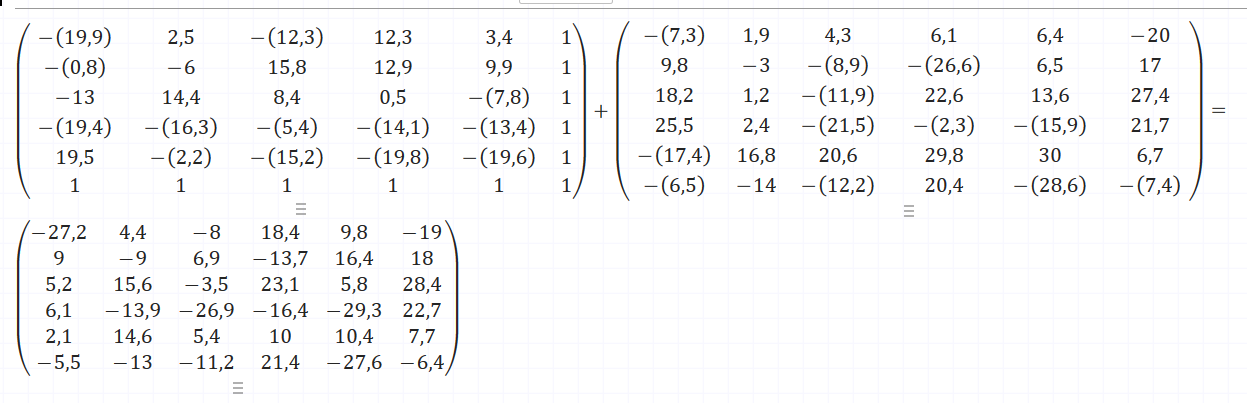
2А



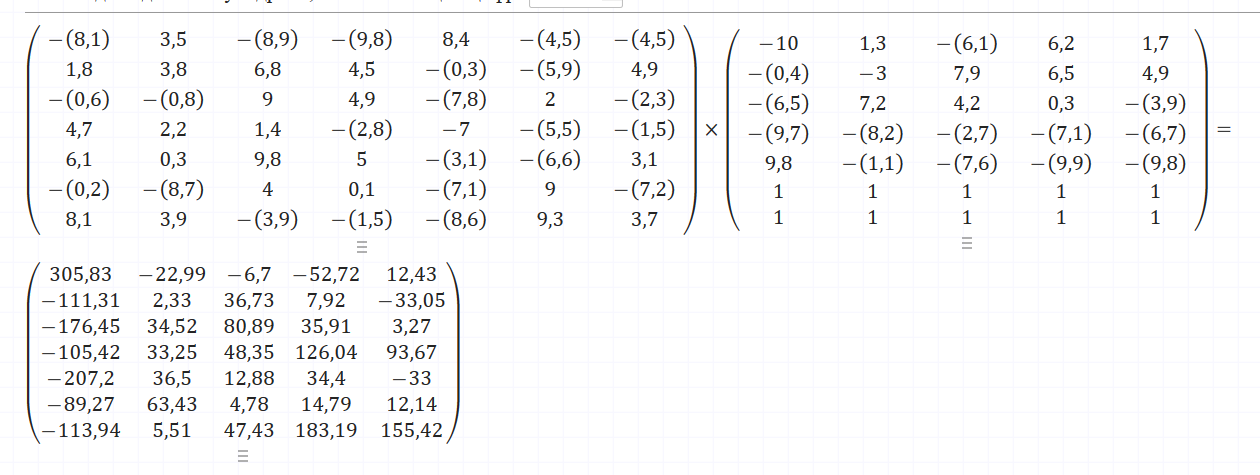
3В



2А + 3В



CA



2A + 3B - CA

