**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**Кафедра інтелектуальних технологій**

Лабораторна робота №10  
*(вид роботи: лабораторна робота, індивідуальне завдання, курсова робота тощо)*

з дисципліни «Алгоритмізація та програмування»Тема роботи: «Багатовимірні масиви даних бібліотеки STL»  
**Варіант № 1**

Виконав(-ла) студент(-ка)  
групи АнД - 11  
Яковкін Микола Андрійович

Перевірив(-ла):  
ПІП викладача

Київ – 2022

**Завдання 1**

**Задана матриця розміром КхР (К та Р вводить користувач). Видалити з**

**матриці: 1) рядок, що містить максимальний елемент матриці; 2) стовбець,**

**який містить значення, що було найближчим до максимального.**

**Подання математичної постановки задачі у вигляді таблиці:**

**1.1 Тестування програми**

1. Для тестування заданий масив:

(1, 2, 3, 4

5, 0, 6, 7

11, 12, 23, 54,

-2, -9, 8, 7

45, 13, 21, 33)

Маємо отримати:

(1, 2, 3, 4

5, 0, 6, 7

-2, -9, 8, 7

45, 13, 21, 33)

( 2, 3, 4

0, 6, 7

-9, 8, 7

13, 21, 33)

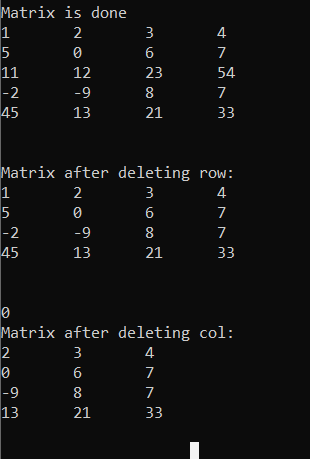


Рисунок 1.1 – Тестування програми.

2. Для тестування заданий масив:

(1, 2, 3, 4

5, 0, 6, 7

11, 12, 23, -9)

Маємо отримати:

(1, 2, 3, 4

5, 0, 6, 7)

( 1, 2, 3,

5, 0, 6)

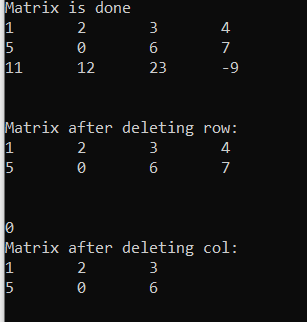


Рисунок 1.2 – Тестування програми.

3. Для тестування заданий масив:

(11, 22, 15, 17, 18

23, 10, -8, 7, 6,

45, 29, 100, 12, 1,

99, 84, 120, 11, 5)

Маємо отримати:

(11, 22, 15, 17, 18

23, 10, -8, 7, 6,

45, 29, 100, 12, 1,)

(11, 22, 17, 18

23, 10, -7, 6,

45, 29, 12, 1,)

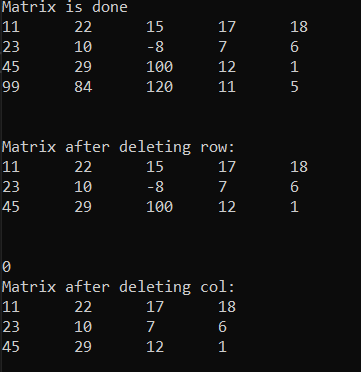


Рисунок 1.3 – Тестування програми.

**1.2 Текст програмною мовою С++.**

#**include** <iostream>

#**include** <vector>

#**include** <cstdlib>

#**include**<ctime>

**using** **namespace** std;

**vector**<**vector**<**int**>>initializeMatrix(**vector**<**vector**<**int**>>&, **int**);

**vector**<**vector**<**int**>>autoInitializeMatrix(**vector**<**vector**<**int**>>&, **int**);

**void** displayMatrix(**vector**<**vector**<**int**>>&);

**int** main()

{

**int** rows, cols;

**vector**<**vector**<**int**>> matrix;

**int** maxValue = 0, id = 0;

cout << "Input rows: " << endl;

cin >> rows;

matrix.**resize**(rows);

cout << "Input cols: " << endl;

cin >> cols;

**initializeMatrix**(matrix, cols);

//**autoInitializeMatrix**(matrix, cols);

cout << "Matrix is done" << endl;

**displayMatrix**(matrix);

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

{

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

{

**if** (matrix[i][j] > maxValue)

{

id = i;

maxValue = matrix[i][j];

}

}

}

matrix.**erase**(matrix.begin() + id);

rows--;

maxValue = 0;

cout << "Matrix after deleting row:" << endl;

**displayMatrix**(matrix);

cout << maxValue << endl;

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

{

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

{

**if** (matrix[i][j] > maxValue)

{

id = j;

maxValue = matrix[i][j];

}

}

}

**for** (**int** i = 0; i < matrix.**size**(); i++)

matrix[i].**erase**(matrix[i].**begin**() + id);

cout << "Matrix after deleting col:" << endl;

**displayMatrix**(matrix);

**return** 0;

}

**vector**<**vector**<**int**>>autoInitializeMatrix(**vector**<**vector**<**int**>>& vec, **int** cols)

{

**srand**(**time**(0));

**for** (**int** i = 0; i < vec.**size**(); i++) { vec[i].**resize**(cols); }

**for** (int i = 0; i < vec.**size**(); i++)

{

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

{

**int** element = 1 + **rand**() % (9 + 1);

vec[i].**push**\_**back**(element);

cout << element << '\t';

}

cout << endl;

}

cout << endl << endl;

**return** vec;

}

**vector**<**vector**<**int**>>initializeMatrix(**vector**<**vector**<**int**>>& vec, int cols)

{

**for** (**int** i = 0; i < vec.**size**(); i++) { vec[i].**resize**(cols); }

**for** (**int** i = 0; i < vec.**size**(); i++)

{

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

{

cout << "Enter " << j + 1 << "th element: " << endl;

cin >> vec[i][j];

}

}

**return** vec;

}

**void** displayMatrix(**vector**<**vector**<**int**>>& vec)

{

**for** (**int** i = 0; i < vec.**size**(); i++)

{

**for** (**int** j = 0; j < vec[i].**size**(); j++)

{

cout << vec[i][j] << '\t';

}

cout << endl;

}

cout << endl << endl;

}

**Завдання 2**

**Задана матриця розміром КхР (К та Р вводить користувач). 1) Після рядка, що містить всі додатні значення вставити рядок з нулями. 2) Перед стовпцем, елементи якого утворюють спадну послідовність вставити стовпець з одиницями.**

**2.1 Тестування програми**

1. Для тестування заданий масив:

(1, 2, 8, 4,

-2, -3, 5, 6

1, 2, 3, 4)

Маємо отримати:

(1, 2, 1, 8, 4,

0, 0, 1, 0, 0

-2, -3, 1, 5, 6

1, 2, 1, 3, 4

0, 0, 1, 0, 0)

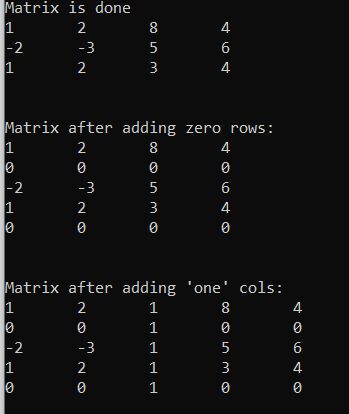
****

Рисунок 2.1 – Тестування програми.

2. Для тестування заданий масив:

(1, 2, 8, 4,

-2, -3, 5, 6

1, -2, 3, 4)

Маємо отримати:

(1, 2, 1, 8, 4,

0, 0, 1, 0, 0

-2, -3, 1, 5, 6

1, -2, 1, 3, 4)

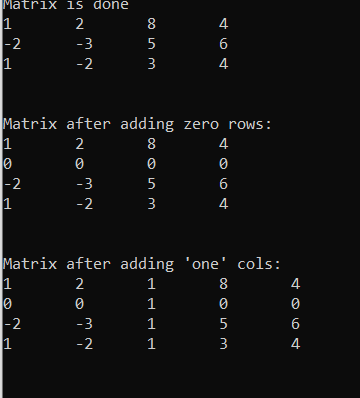
****

Рисунок 2.2 – Тестування програми.

3. Для тестування заданий масив:

(12, 21, 68, 4,

-2, -3, 5, 0

-10, -4, 3, -1)

Маємо отримати:

(1, 12, 1, 21, 1, 68, 1, 4,

1, 0, 1, 0, 1, 0, 1, 0

1, -2, 1, -3, 1, 5, 1, 0

1, -10, 1, -4, 1, 3, 1, -1)

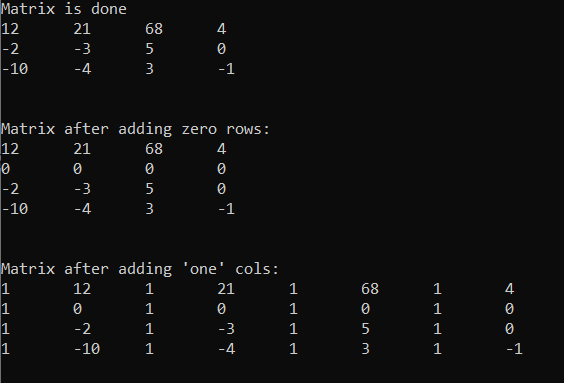
****

Рисунок 2.3 – Тестування програми.

**2.2 Текст програмною мовою С++.**

#**include** <iostream>

#**include** <vector>

**using** **namespace** std;

**vector**<**vector**<**int**>>initializeMatrix(**vector**<**vector**<**int**>>&, **int**);

**void** displayMatrix(**vector**<**vector**<**int**>>&);

**int** main()

{

**int** rows, cols;

**vector**<**vector**<**int**>> matrix;

cout << "Input rows: " << endl;

cin >> rows;

matrix.**resize**(rows);

cout << "Input cols: " << endl;

cin >> cols;

**vector**<**int**> zeroRow(**cols**);

**for** (**int** i = 0; i < zeroRow.**size**(); i++) { zeroRow[i] = 0; }

**vector**<**int**> rowIds;

**vector**<**int**> colIds;

**initializeMatrix**(matrix, cols);

cout << "Matrix is done" << endl;

**displayMatrix**(matrix);

**for** (**int** i = 0; i < matrix.**size**(); i++)

{

**int** addZero = 1;

**for** (**int** j = 0; j < matrix[i].**size**(); j++)

{

**if** (matrix[i][j] < 0)

addZero = 0;

}

**if** (addZero)

rowIds.**push**\_**back**(i);

}

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

{

**int** addOne = 1;

**for** (**int** i = 0; i + 1 < rows; i++)

{

**if** (matrix[i][j] <= matrix[i + 1][j])

addOne = 0;

}

**if** (addOne)

colIds.**push**\_**back**(j);

}

**for** (**int** i = rowIds.**size**() - 1; i >= 0; i--)

{

matrix.**insert**(matrix.**begin**() + rowIds[i] + 1, zeroRow);

}

cout << "Matrix after adding zero rows: " << endl;

**displayMatrix**(matrix);

**for** (**int** i = colIds.**size**() - 1; i >= 0; i--)

{

**for** (**int** j = 0; j < matrix.**size**(); j++)

matrix[j].**insert**(matrix[j].**begin**() + colIds[i], 1);

}

cout << "Matrix after adding 'one' cols: " << endl;

**displayMatrix**(matrix);

}

**vector**<**vector**<**int**>>initializeMatrix(**vector**<**vector**<**int**>>& vec, **int** cols)

{

**for** (**int** i = 0; i < vec.**size**(); i++) { vec[i].**resize**(cols); }

**for** (**int** i = 0; i < vec.**size**(); i++)

{

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

{

cout << "Enter " << j + 1 << "th element: " << endl;

cin >> vec[i][j];

}

}

**return** vec;

}

**void** displayMatrix(**vector**<**vector**<**int**>>& vec)

{

**for** (**int** i = 0; i < vec.**size**(); i++)

{

**for** (**int** j = 0; j < vec[i].**size**(); j++)

{

cout << vec[i][j] << '\t';

}

cout << endl;

}

cout << endl << endl;

}

**Завдання 3**

**Задана матриця розміром КхР (К та Р вводить користувач). 1) Поміняти місцями рядок, що містить всі парні елементи з рядком, що містить всі непарні елементи (вважати, що такі рядки існують по одному). 2) Поміняти місцями стовпці, що містять мінімальний та максимальний елементи.**

**3.1 Тестування програми**

1. Для тестування заданий масив:

(4, 2, 8, 6,

-1, -3, 5, 7

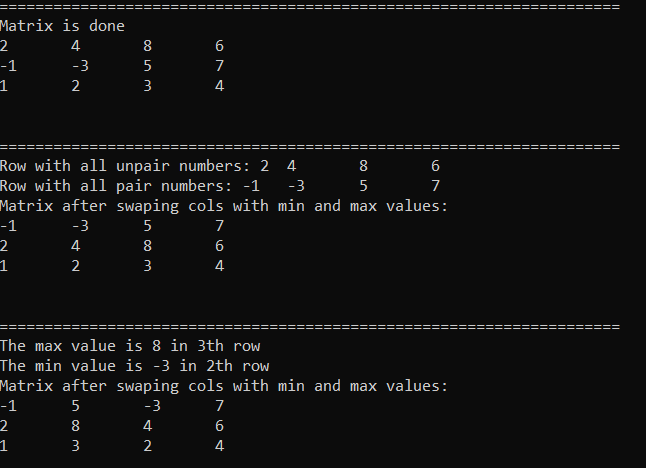
1, 2, 3, 4)

Маємо отримати:

( -1, 5, -3, 7

4, 8, 1, 6,

1, 3, 1, 4)

****Рисунок 3.1 – Тестування програми.

2. Для тестування заданий масив:

(9, 13, 7, 3,

-2, -4, 6, 8

11, 2, -6, 4)

Маємо отримати:

(-2, 6, -4, 8

9, 7, 13, 3,

11, -6, 2, 4)

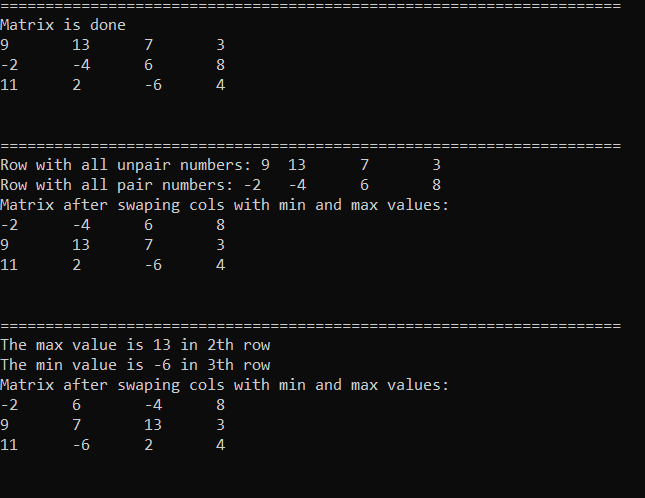


Рисунок 3.2 – Тестування програми

3. Для тестування заданий масив:

( 22, 11, 33, 44

-22, -44, -11, 33

0, 0, 0, 0,

1, 2, 1, 1)

Маємо отримати:

(0, 0, 0, 0,

-22, 33, -11, -44

22, 44, 33, 11

1, 1, 1, 2)

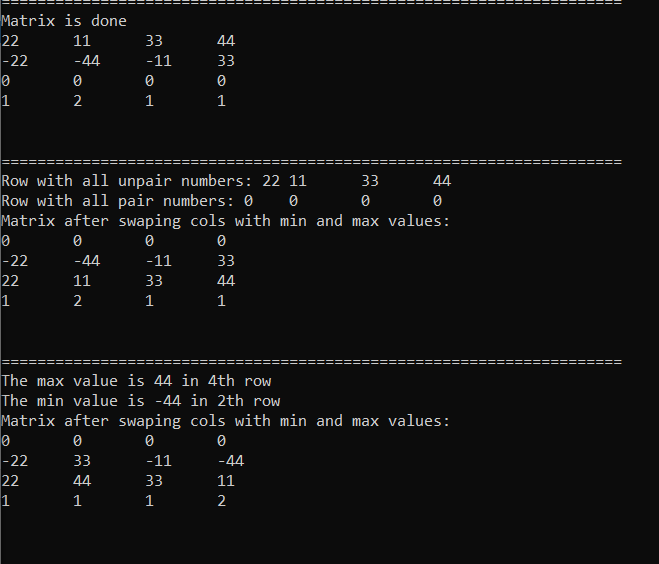


Рисунок 3.3 – Тестування програми.

**3.2 Текст програмною мовою С++.**

#**include** <iostream>

#**include** <vector>

**using** **namespace** std;

**vector**<**vector**<**int**>>initializeMatrix(vector<vector<int>>&, int);

**void** displayMatrix(vector<vector<int>>&);

**int** main()

{

**int** rows, cols;

**vector**<**vector**<**int**>> matrix;

**int** pairId = 0, unpairId = 0;

**int** maxColValue = 0,

minColValue = 10000000000000;

**int** maxId = 0,

minId = 0;

cout << "Input rows: " << endl;

cin >> rows;

matrix.**resize**(rows);

cout << "Input cols: " << endl;

cin >> cols;

**initializeMatrix**(matrix, cols);

cout << "=====================================================================" << endl;

cout << "Matrix is done" << endl;

**displayMatrix**(matrix);

**for** (**int** i = 0; i < matrix.**size**(); i++)

{

**int** pair = 1,

unpairId = 1;

**for** (**int** j = 0; j < matrix[i].**size**(); j++)

{

**if** (matrix[i][j]%2)

pair = 0;

else

unpair = 0;

}

**if** (pair)

pairId = i;

**if** (unpair)

unpairId = i;

}

cout << "=====================================================================" << endl;

cout << "Row with all unpair numbers: ";

**for** (**int** i = 0; i < cols; i++) { cout << matrix[unpairId][i] << '\t'; }

cout << "\nRow with all pair numbers: ";

**for** (**int** i = 0; i < cols; i++) { cout << matrix[pairId][i] << '\t'; }

matrix[pairId].swap(matrix[unpairId]);

cout << "\nMatrix after swaping cols with min and max values: " << endl;

**displayMatrix**(matrix);

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

{

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

{

**if** (matrix[i][j] > maxColValue)

{

maxColValue = matrix[i][j];

maxId = j;

}

**if** (matrix[i][j] < minColValue)

{

minColValue = matrix[i][j];

minId = j;

}

}

}

cout << "=====================================================================" << endl;

cout << "The max value is " << maxColValue << " in " << maxId + 1 << "th row" << endl;

cout << "The min value is " << minColValue << " in " << minId + 1 << "th row" << endl;

**for** (**int** i = 0; i < matrix.**size**(); i++)

**swap**(matrix[i][minId], matrix[i][maxId]);

cout << "Matrix after swaping cols with min and max values: " << endl;

**displayMatrix**(matrix);

**return** 0;

}

**vector**<**vector**<**int**>>initializeMatrix(**vector**<**vector**<**int**>>& vec, **int** cols)

{

**for** (**int** i = 0; i < vec.**size**(); i++) { vec[i].**resize**(cols); }

**for** (**int** i = 0; i < vec.**size**(); i++)

{

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

{

cout << "Enter " << j + 1 << "th element: " << endl;

cin >> vec[i][j];

}

}

**return** vec;

}

**void** displayMatrix(**vector**<**vector**<**int**>>& vec)

{

**for** (**int** i = 0; i < vec.**size**(); i++)

{

**for** (**int** j = 0; j < vec[i].**size**(); j++)

{

cout << vec[i][j] << '\t';

}

cout << endl;

}

cout << endl << endl;

}

**Завдання 4**

**Задана квадранта матриця (розмір матриці може бути довільний). Вивести на екран елементи головної та побічної діагоналей. Обнулити елементи вище головної та побічної діагоналі (позначено на схемі).**

**4.1 Тестування програми**

1. Для тестування заданий масив 4х4:

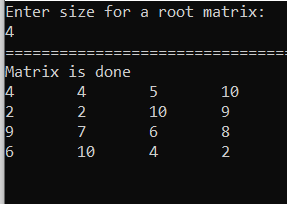


Рисунок 4.1 – Вхідні дані.

Маємо отримати:

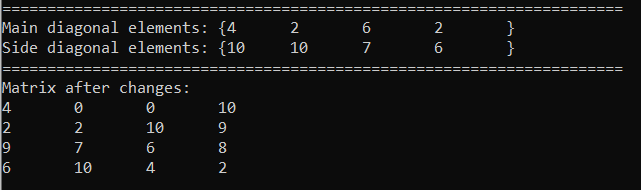


Рисунок 4.2 – Тестування програми.

2. Для тестування заданий масив 5х5:

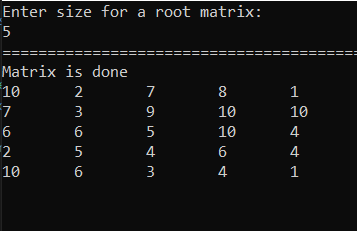


Рисунок 4.3 – Вхідні дані.

Маємо отримати:

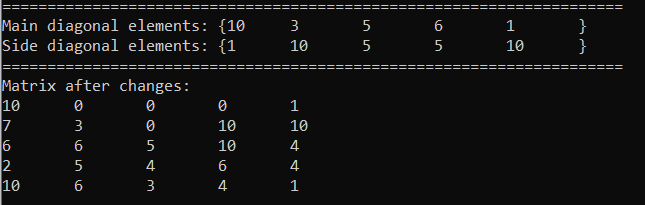


Рисунок 4.4 – Тестування програми.

3. Для тестування заданий масив 6х6:

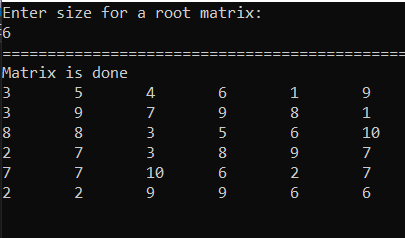


Рисунок 4.5 – Вхідні дані.

Маємо отримати:

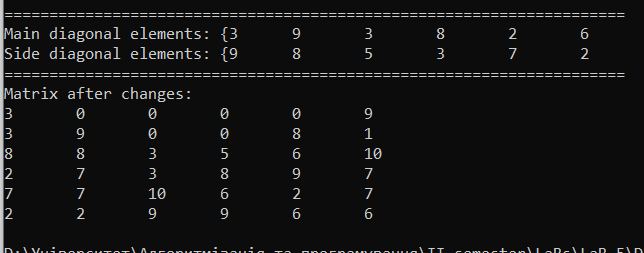


Рисунок 4.6 – Тестування програми.

**4.2 Текст програмною мовою С++.**

#**include** <iostream>

#**include** <vector>

**using** **namespace** std;

**vector**<**vector**<**int**>>autoInitializeMatrix(**vector**<**vector**<**int**>>&, **int**);

**void** displayMatrix(**vector**<**vector**<**int**>>&);

**int** main()

{

**int** size;

**vector**<**vector**<**int**>> matrix;

cout << "Enter size for a root matrix: " << endl;

cin >> size;

matrix.**resize**(size);

cout << "=====================================================================" << endl;

cout << "Matrix is done" << endl;

**autoInitializeMatrix**(matrix, size);

cout << "=====================================================================" << endl;

cout << "Main diagonal elements: {";

**for** (**int** i = 0; i < size; i++) { cout << matrix[i][i] << '\t'; }

cout << "}" << endl;

cout << "Side diagonal elements: {";

**for** (**int** i = 0; i < size; i++) { cout << matrix[i][size - (i + 1)] << '\t'; }

cout << "}" << endl;

cout << "=====================================================================" << endl;

cout << "Matrix after changes: " << endl;

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

{

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

{

**if** (i < j && i < size - (j + 1))

{

matrix[i][j] = 0;

}

cout << matrix[i][j] << '\t';

}

cout << endl;

}

**return** 0;

}

**vector**<**vector**<**int**>>initializeMatrix(**vector**<**vector**<**int**>>& vec, **int** cols)

{

**for** (**int** i = 0; i < vec.**size**(); i++) { vec[i].**resize**(cols); }

**for** (**int** i = 0; i < vec.**size**(); i++)

{

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

{

//cout << "Enter " << j + 1 << "th element: " << endl;

//cin >> vec[i][j];

vec[i][j] = 1;

}

}

**return** vec;

}

**vector**<**vector**<**int**>>autoInitializeMatrix(**vector**<**vector**<**int**>>& vec, **int** size)

{

**srand**(**time**(0));

**for** (**int** i = 0; i < size; i++) { vec[i].**resize**(size); }

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

{

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

{

vec[i][j] = 1 + **rand**() % (9 + 1);

cout << vec[i][j] << '\t';

}

cout << endl;

}

cout << endl << endl;

**return** vec;

}

**void** displayMatrix(**vector**<**vector**<**int**>>& vec)

{

**for** (**int** i = 0; i < vec.**size**(); i++)

{

**for** (**int** j = 0; j < vec[i].**size**(); j++)

{

cout << vec[i][j] << '\t';

}

cout << endl;

}

cout << endl << endl;

}

**Завдання 5**

**Задана прямокутна матриця розміром КхР ( P >= K ). 1) Знайти суми елементів головної діагоналі та для кожної діагоналей, що їй паралельні. 2) Знайти добутки елементів побічної діагоналі та для кожної діагоналей, що їй паралельні.**

**5.1 Тестування програми**

1. Для тестування заданий масив 4х5:

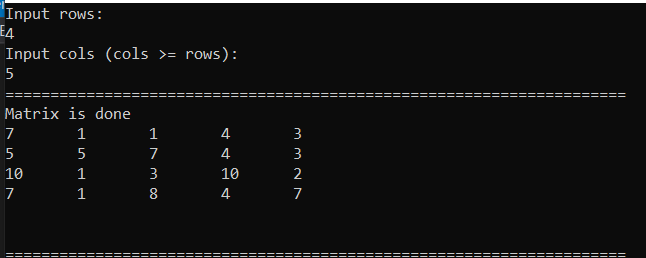


Рисунок 5.1 – Вхідні дані.

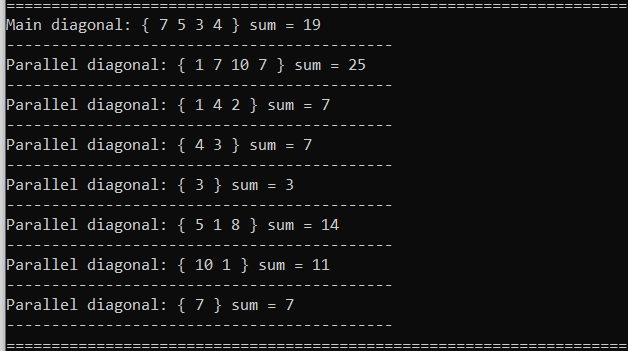
Маємо отримати:

Рисунок 5.2 – Тестування програми.

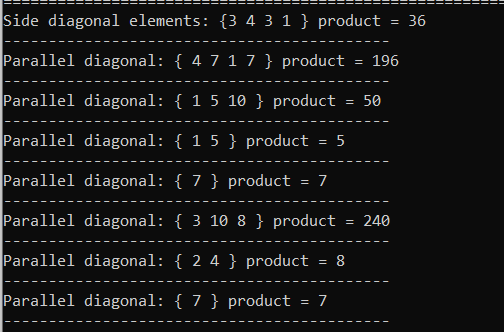


Рисунок 5.3 – Тестування програми.

2. Для тестування заданий масив 6х7:

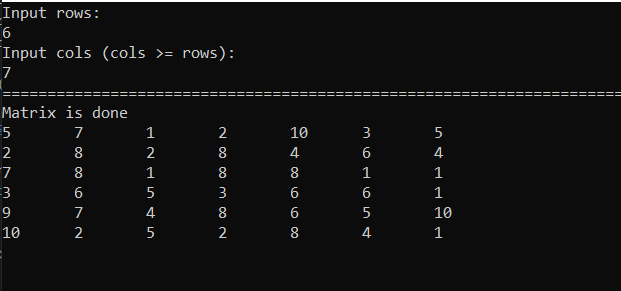


Рисунок 5.4 – Вхідні дані.

Маємо отримати:

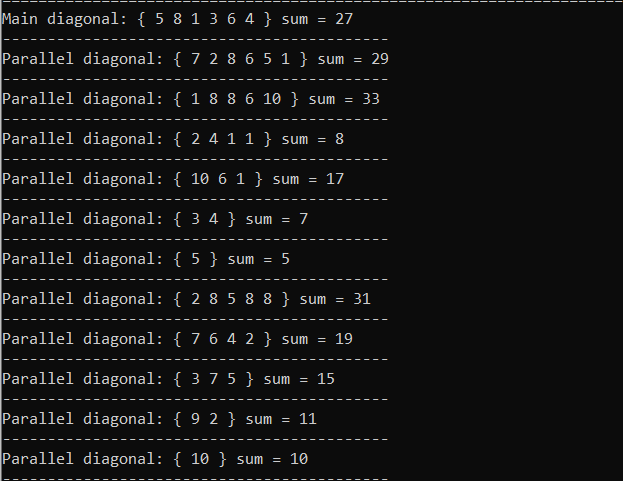


Рисунок 5.5 – Тестування програми.

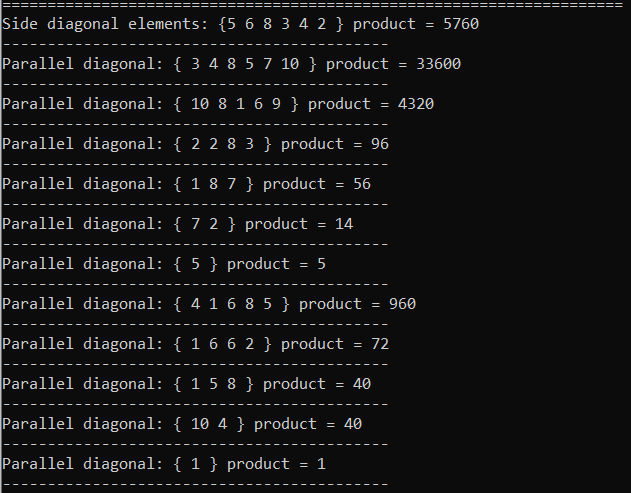


Рисунок 5.6 – Тестування програми.

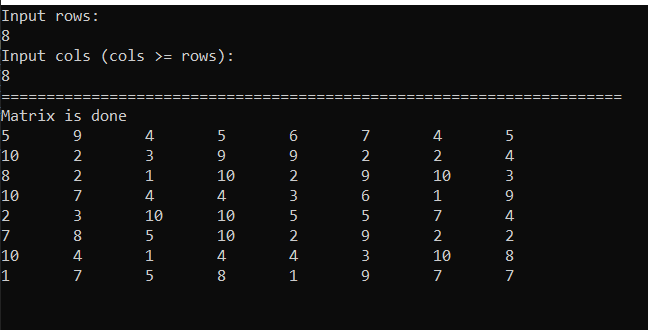
3. Для тестування заданий масив 8х8:

Рисунок 5.7 – Тестування програми.

Маємо отримати:

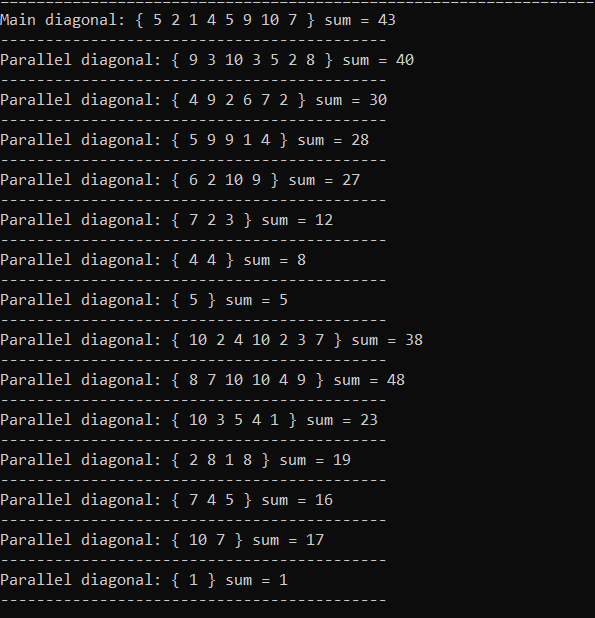


Рисунок 5.8 – Тестування програми.

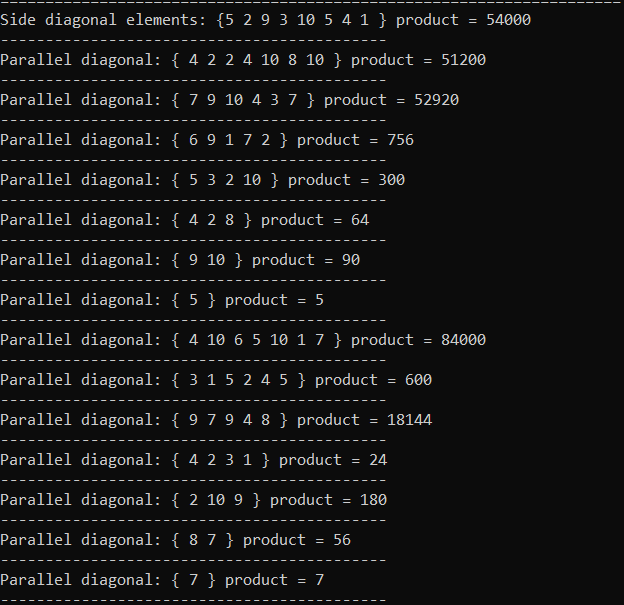


Рисунок 5.9 – Тестування програми.

**5.2 Текст програмною мовою С++.**

#**include** <iostream>

#**include** <vector>

**std**::**vector**<**std**::**vector**<**int**>>autoInitializeMatrix(**std**::**vector**<**std**::**vector**<**int**>>&, **int**);

**void** findSum(**std::vector**<**std**::**vector**<**int**>>&, **int**, **int**);

**void** displayMatrix(**std::vector**<**std::vector**<**int**>>&);

**void** findProduct(**std::vector**< **std::vector**<**int**>>&, **int**, **int**);

**int** main()

{

using namespace std;

**int** rows, cols;

**vector**<**vector**<**int**>> matrix;

cout << "Input rows: " << endl;

cin >> rows;

matrix.**resize**(rows);

cout << "Input cols (cols >= rows): " << endl;

cin >> cols;

cout << "=====================================================================" << endl;

cout << "Matrix is done" << endl;

**autoInitializeMatrix**(matrix, cols);

cout << "=====================================================================" << endl;

**for** (**int** i = 0; i < cols; i++)

{

**findSum**(matrix, 0, i);

cout << "------------------------------------------- " << endl;

}

**for** (**int** j = 1; j < rows; j++)

{

**findSum**(matrix, j, 0);

cout << "------------------------------------------- " << endl;

}

cout << "=====================================================================" << endl;

**for** (**int** i = 0; i < cols; i++)

{

**findProduct**(matrix, 0, i);

cout << "------------------------------------------- " << endl;

}

**for** (**int** j = 1; j < rows; j++)

{

**findProduct**(matrix, j, 0);

cout << "------------------------------------------- " << endl;

}

**return** 0;

}

**std::vector**<**std::vector**<**int**>>initializeMatrix(**std::vector**<**std::vector**<**int**>>& vec, **int** cols)

{

**using namespace std;**

**for** (**int** i = 0; i < vec.**size**(); i++) { vec[i].**resize**(cols); }

**for** (**int** i = 0; i < vec.**size**(); i++)

{

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

{

//cout << "Enter " << j + 1 << "th element: " << endl;

//cin >> vec[i][j];

vec[i][j] = 1;

}

}

**return** vec;

}

**std**::**vector**<**std**::**vector**<**int**>>autoInitializeMatrix(**std**::**vector**<**std::vector**<**int**>>& vec, **int** size)

{

**using** **namespace** std;

**srand**(time(0));

**for** (**int** i = 0; i < vec.**size**(); i++) { vec[i].**resize**(size); }

**for** (**int** i = 0; i < vec.**size**(); i++)

{

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

{

vec[i][j] = 1 + rand() % (9 + 1);

cout << vec[i][j] << '\t';

}

cout << endl;

}

cout << endl << endl;

**return** vec;

}

**void** displayMatrix(**std**::**vector**<**std**::**vector**<**int**>>& vec)

{

**using** **namespace** std;

**for** (**int** i = 0; i < vec.**size**(); i++)

{

**for** (**int** j = 0; j < vec[i].**size**(); j++)

{

cout << vec[i][j] << '\t';

}

cout << endl;

}

cout << endl << endl;

}

**void** findSum(**std**::**vector**<**std**::**vector**<**int**>>& **vec**, **int** R, **int** C)

{

**using namespace std;**

**int** sum = 0;

**if** (!R && !C)

{

cout << "Main diagonal: { ";

**for** (**int** i = 0; i < vec.**size**(); i++)

{

cout << vec[i][i] << " ";

sum += vec[i][i];

}

}

**if** (R)

{

cout << "Parallel diagonal: { ";

**for** (**int** i = 0; i < vec.**size**()- R; i++)

{

cout << vec[i + R][i] << " ";

sum += vec[i + R][i];

}

}

**if**(C)

{

cout << "Parallel diagonal: { ";

**for** (**int** i = 0; i < vec[0].**size**() - C; i++)

{

cout << vec[i][i + C] << " ";

sum += vec[i][i + C];

}

}

cout << "} sum = " << sum << endl;

}

**void** findProduct(**std**::**vector**<**std**::**vector**<**int**>>& vec, **int** R, **int** C)

{

**using** **namespace** **std**;

**int** product = 1;

**if** (!R && !C)

{

cout << "Side diagonal elements: {";

**for** (**int** i = 0; i < vec.**size**(); i++)

{

cout << vec[i][vec[0].**size**() - (i + 1)] << " ";

product \*= vec[i][vec[0].**size**() - (i + 1)];

}

}

**if** (C)

{

cout << "Parallel diagonal: { ";

for (**int** i = 0; i < vec[0].**size**() - C; i++)

{

cout << vec[i][vec[0].**size**() - (i + C + 1)] << " ";

product \*= vec[i][vec[0].**size**() - (i + C + 1)];

}

}

**if**(R)

{

cout << "Parallel diagonal: { ";

**for** (**int** i = 0; i < vec.size() - R; i++)

{

cout << vec[i + R][vec[0].**size**() - i - 1] << " ";

product \*= vec[i + R][vec[0].**size**() - i - 1];

}

}

cout << "} product = " << product << endl;

}