Tema laborator 3 AF

Vasile Mihai Teodor – 10LF204

Ex1

#include <iostream>

using namespace std;

void Input\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cin >> matrix[counter\_line][counter\_column];

}

void Output\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

void Maximum\_main\_diagonal(int size\_matrix, int matrix[][101], unsigned& position)

{

int max\_value = 0;

for (int counter = 0; counter < size\_matrix; counter++)

{

if (max\_value < matrix[counter][counter])

{

max\_value = matrix[counter][counter];

position = counter;

}

}

}

void Minimum\_secondary\_diagonal(int size\_matrix, int matrix[][101], unsigned& position)

{

int min\_value = 1001;

for (int counter = 0; counter < size\_matrix; counter++)

{

if (min\_value > matrix[counter][size\_matrix-counter-1])

{

min\_value = matrix[counter][size\_matrix - counter - 1];

position = counter;

}

}

}

int main()

{

unsigned n,position\_min,position\_max;

int matrix[101][101];

cin >> n;

Input\_quadratic\_matrix(n, matrix);

Maximum\_main\_diagonal(n, matrix, position\_max);

Minimum\_secondary\_diagonal(n, matrix, position\_min);

swap(matrix[position\_max][position\_max], matrix[position\_min][n - position\_min - 1]);

Output\_quadratic\_matrix(n,matrix);

}

Ex2

#include <iostream>

using namespace std;

void Input\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cin >> matrix[counter\_line][counter\_column];

}

void Output\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

int Sum\_upper\_triangle(int size\_matrix, int matrix[][101])

{

int sum = 0;

if (size\_matrix % 2 == 0)

{

for (int counter\_line = 0; counter\_line < size\_matrix / 2 - 1; counter\_line++)

for (int counter\_column = counter\_line + 1; counter\_column < size\_matrix - counter\_line - 1; counter\_column++)

sum = sum + matrix[counter\_line][counter\_column];

return sum;

}

else

{

for (int counter\_line = 0; counter\_line < size\_matrix / 2; counter\_line++)

for (int counter\_column = counter\_line + 1; counter\_column < size\_matrix - counter\_line - 1; counter\_column++)

sum = sum + matrix[counter\_line][counter\_column];

return sum;

}

}

int Sum\_lower\_triangle(int size\_matrix, int matrix[][101])

{

int sum = 0;

for (int counter\_line = size\_matrix / 2 + 1; counter\_line < size\_matrix; counter\_line++)

for (int counter\_column = size\_matrix - counter\_line; counter\_column < counter\_line; counter\_column++)

sum = sum + matrix[counter\_line][counter\_column];

return sum;

}

int Sum\_left\_triangle(int size\_matrix, int matrix[][101])

{

int sum = 0;

for (int counter\_line = 1; counter\_line < size\_matrix - 1; counter\_line++)

{

if (counter\_line < size\_matrix / 2)

for (int counter\_column = 0; counter\_column < counter\_line; counter\_column++)

sum = sum + matrix[counter\_line][counter\_column];

else

for (int counter\_column = 0; counter\_column < size\_matrix - counter\_line - 1; counter\_column++)

sum = sum + matrix[counter\_line][counter\_column];

}

return sum;

}

int Sum\_right\_triangle(int size\_matrix, int matrix[][101])

{

int sum = 0;

for (int counter\_line = 1; counter\_line < size\_matrix - 1; counter\_line++)

{

if (counter\_line < size\_matrix / 2)

for (int counter\_column = size\_matrix - 1; counter\_column >= size\_matrix - counter\_line; counter\_column--)

sum = sum + matrix[counter\_line][counter\_column];

else

for (int counter\_column = size\_matrix - 1; counter\_column > counter\_line; counter\_column--)

sum = sum + matrix[counter\_line][counter\_column];

}

return sum;

}

int main()

{

int matrix[101][101];

unsigned n;

cin >> n;

Input\_quadratic\_matrix(n, matrix);

cout << "Suma din triunghiul de deasupra celor doua diagonale este: " << Sum\_upper\_triangle(n, matrix)<<endl;

cout << "Suma din triunghiul de dedesubtul celor doua diagonale este: " << Sum\_lower\_triangle(n, matrix)<<endl;

cout << "Suma din triunghiul din stanga celor doua diagonale este: " << Sum\_left\_triangle(n, matrix)<<endl;

cout << "Suma din triunghiul din dreapta celor doua diagonale este: " << Sum\_right\_triangle(n, matrix);

}

Ex 3

#include <iostream>

using namespace std;

int counting\_distinct\_elements(int size\_column, int matrix[][101],int line)

{

int counter\_distinct = 0,array[101];

for (int counter = 0; counter < size\_column; counter++)

array[counter] = matrix[line][counter];

for (int counter = 0; counter < size\_column; counter++)

{

int check\_condition = 1;

for (int seconday\_counter =0; seconday\_counter < size\_column && check\_condition == 1; seconday\_counter++)

if(seconday\_counter!=counter)

if (array[counter] == array[seconday\_counter])

check\_condition = 0;

if (check\_condition)

counter\_distinct++;

}

return counter\_distinct;

}

int Maximum(int size\_column, int matrix[][101])

{

int max = 0;

for (int counter = 0; counter < size\_column; counter++)

if (counting\_distinct\_elements(size\_column, matrix, counter) > max)

max = counting\_distinct\_elements(size\_column, matrix, counter);

return max;

}

void Eliminating\_lines(int size\_column, int& size\_lines, int matrix[][101],int position)

{

for (int counter = position; counter < size\_lines; counter++)

for (int secondary\_counter = 0; secondary\_counter < size\_column; secondary\_counter++)

matrix[counter][secondary\_counter] = matrix[counter + 1][secondary\_counter];

size\_lines--;

}

void Input\_matrix(int size\_lines,int size\_columns, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_lines; counter\_line++)

for (int counter\_column = 0; counter\_column < size\_columns; counter\_column++)

cin >> matrix[counter\_line][counter\_column];

}

void Output\_matrix(int size\_lines,int size\_columns, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_lines; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_columns; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

int main()

{

int n, m, max\_distinct\_elements = 0;

int matrix[101][101];

cin >> n >> m;

Input\_matrix(n, m, matrix);

cout << endl;

cout << "Liniile cu numar maxim de elemente distincte sunt: " << endl;

max\_distinct\_elements = Maximum(m, matrix);

for (int counter = 0; counter < n; counter++)

{

if (counting\_distinct\_elements(m, matrix, counter) == max\_distinct\_elements)

{

for (int secondary\_counter = 0; secondary\_counter < m; secondary\_counter++)

cout << matrix[counter][secondary\_counter] << " ";

cout << endl;

Eliminating\_lines(m, n, matrix, counter);

counter--;

}

}

cout << endl;

cout << "Matricea ramasa in urma eliminarilor este:" << endl;

Output\_matrix(n, m, matrix);

}

Ex4

#include <iostream>

using namespace std;

void Input\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cin >> matrix[counter\_line][counter\_column];

}

void Output\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

void Sorting\_in\_ascending\_order\_the\_main\_diagonal(int size\_matrix, int matrix[][101])

{

for (int counter = 0; counter < size\_matrix-1; counter++)

for (int secondary\_counter = counter + 1; secondary\_counter < size\_matrix; secondary\_counter++)

if (matrix[counter][counter] > matrix[secondary\_counter][secondary\_counter])

{

for (int sorting\_counter = 0; sorting\_counter < size\_matrix; sorting\_counter++)

{

int aux = matrix[counter][sorting\_counter];

matrix[counter][sorting\_counter] = matrix[secondary\_counter][sorting\_counter];

matrix[secondary\_counter][sorting\_counter] = aux;

}

for (int sorting\_counter = 0; sorting\_counter <= size\_matrix; sorting\_counter++)

{

int aux = matrix[sorting\_counter][counter];

matrix[sorting\_counter][counter] = matrix[sorting\_counter][secondary\_counter];

matrix[sorting\_counter][secondary\_counter] = aux;

}

}

}

int main()

{

int matrix[101][101];

unsigned n;

cin >> n ;

Input\_quadratic\_matrix(n, matrix);

Sorting\_in\_ascending\_order\_the\_main\_diagonal(n, matrix);

Output\_quadratic\_matrix(n, matrix);

}

Ex5

#include <iostream>

using namespace std;

void Input\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cin >> matrix[counter\_line][counter\_column];

}

void Output\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

void matrix\_rotation(int size\_matrix, int initial\_matrix[][101], int rotated\_matrix[][101])

{

for (int counter = 0; counter < size\_matrix; counter++)

for (int secondary\_counter = 0; secondary\_counter < size\_matrix; secondary\_counter++)

rotated\_matrix[counter][secondary\_counter] = initial\_matrix[size\_matrix - secondary\_counter - 1][counter];

}

void copying\_matrix(int size\_matrix, int new\_matrix[][101],int copied\_matrix[][101])

{

for (int counter = 0; counter < size\_matrix; counter++)

for (int secondary\_counter = 0; secondary\_counter < size\_matrix; secondary\_counter++)

new\_matrix[counter][secondary\_counter] = copied\_matrix[counter][secondary\_counter];

}

int main()

{

int matrix[101][101], aux\_matrix[101][101];

int n;

cin >> n;

Input\_quadratic\_matrix(n, matrix);

matrix\_rotation(n, matrix, aux\_matrix);

copying\_matrix(n, matrix, aux\_matrix);

Output\_quadratic\_matrix(n, matrix);

}

Ex6

#include <iostream>

using namespace std;

void Input\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cin >> matrix[counter\_line][counter\_column];

}

void Output\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

void swapping\_same\_parity\_elements\_in\_matrices(int size\_matrix, int matrix[][101])

{

for (int counter = 1; counter < size\_matrix; counter++)

for (int secondary\_counter = 0; secondary\_counter < counter; secondary\_counter++)

if (matrix[counter][secondary\_counter] % 2 == matrix[secondary\_counter][counter] % 2)

swap(matrix[counter][secondary\_counter], matrix[secondary\_counter][counter]);

}

int main()

{

int n, matrix[101][101];

cin >> n;

Input\_quadratic\_matrix(n, matrix);

swapping\_same\_parity\_elements\_in\_matrices(n, matrix);

Output\_quadratic\_matrix(n, matrix);

}

Ex7

#include <iostream>

using namespace std;

void Quadratic\_matrix\_building(int size\_matrix, int matrix[][101])

{

for (int counter = 0; counter < size\_matrix; counter++)

matrix[counter][counter] = 1;

for (int counter = 1; counter < size\_matrix; counter++)

for (int secondary\_counter = 0; secondary\_counter < counter; secondary\_counter++)

matrix[counter][secondary\_counter] = matrix[counter - 1][secondary\_counter]+1;

for (int counter = 0; counter < size\_matrix - 1; counter++)

for (int secondary\_counter = counter + 1; secondary\_counter < size\_matrix; secondary\_counter++)

matrix[counter][secondary\_counter] = matrix[counter][secondary\_counter - 1] + 1;

}

void Output\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

int main()

{

unsigned n;

int matrix[101][101];

cin >> n;

Quadratic\_matrix\_building(n, matrix);

Output\_quadratic\_matrix(n, matrix);

}

Ex8

#include <iostream>

using namespace std;

void Input\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cin >> matrix[counter\_line][counter\_column];

}

void Outputing\_parallel\_secondary\_diagonal(int size\_matrix, int matrix[][101])

{

for (int counter = 0; counter < 2 \* size\_matrix; counter++)

for (int secondary\_counter = 0; secondary\_counter < counter; secondary\_counter++)

if (counter - 1 - secondary\_counter < size\_matrix && secondary\_counter < size\_matrix)

if (counter % 2 == 0)

cout << matrix[counter - 1 - secondary\_counter][secondary\_counter] << " ";

else

cout << matrix[secondary\_counter][counter - 1 - secondary\_counter] << " ";

}

int main()

{

int matrix[101][101];

unsigned n;

cin >> n;

Input\_quadratic\_matrix(n, matrix);

Outputing\_parallel\_secondary\_diagonal(n, matrix);

}

Ex9

#include <iostream>

#include <math.h>

using namespace std;

void Output\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

void buildMatrix(int& size\_matrix,int matrix[][101])

{

int count = 0;

int actual\_size = sqrt(size\_matrix);

while (count < actual\_size / 2)

{

int line = count - 1, row = count;

while (line < actual\_size - 1 - count)

{

line++;

cin >> matrix[line][row];

}

while (row < actual\_size - 1 - count)

{

row++;

cin >> matrix[line][row];

}

while (line > count)

{

line--;

cin >> matrix[line][row];

}

while (row > count + 1)

{

row--;

cin >> matrix[line][row];

}

count++;

}

if (actual\_size % 2 == 1)

cin >> matrix[actual\_size / 2][actual\_size / 2];

size\_matrix = actual\_size;

}

int main()

{

int matrix[101][101];

int n;

cin >> n;

buildMatrix(n, matrix);

Output\_quadratic\_matrix(n, matrix);

}

Ex10

#include <iostream>

using namespace std;

void Input\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cin >> matrix[counter\_line][counter\_column];

}

void Output\_quadratic\_matrix(int size\_matrix, int matrix[][101])

{

for (int counter\_line = 0; counter\_line < size\_matrix; counter\_line++)

{

for (int counter\_column = 0; counter\_column < size\_matrix; counter\_column++)

cout << matrix[counter\_line][counter\_column] << " ";

cout << endl;

}

}

bool Verifying\_element\_condition(int size\_matrix, int matrix[][101])

{

int array\_length = size\_matrix \* size\_matrix ;

int frequency\_array[100000]{};

for (int counter = 0; counter < size\_matrix; counter++)

for (int secondary\_counter = 0; secondary\_counter < size\_matrix; secondary\_counter++)

frequency\_array[matrix[counter][secondary\_counter]]++;

for (int counter = 0; counter < array\_length; counter++)

if (frequency\_array[counter] == 0)

return false;

return true;

}

void Finding\_position\_of\_0(int size\_matrix, int matrix[][101], int& line, int& row)

{

for (int counter = 0; counter < size\_matrix; counter++)

for (int secondary\_counter = 0; secondary\_counter < size\_matrix; secondary\_counter++)

if (matrix[counter][secondary\_counter] == 0)

{

line = counter;

row = secondary\_counter;

break;

}

}

void Swapping\_0(int size\_matrix, int matrix[][101], int line, int row)

{

if (line-1>=0 && line-1<size\_matrix)

{

swap(matrix[line - 1][row], matrix[line][row]);

cout << "0 mutat in N"<<endl;

Output\_quadratic\_matrix(size\_matrix, matrix);

swap(matrix[line - 1][row], matrix[line][row]);

}

cout << endl;

if (row-1>=0 && row-1<size\_matrix)

{

swap(matrix[line][row-1], matrix[line][row]);

cout << "0 mutat in V" << endl;

Output\_quadratic\_matrix(size\_matrix, matrix);

swap(matrix[line][row-1], matrix[line][row]);

}

cout << endl;

if (line+1>=0 && line+1<size\_matrix)

{

swap(matrix[line + 1][row], matrix[line][row]);

cout << "0 mutat in S" << endl;

Output\_quadratic\_matrix(size\_matrix, matrix);

swap(matrix[line + 1][row], matrix[line][row]);

}

cout << endl;

if (row+1>=0 && row+1<size\_matrix)

{

swap(matrix[line][row + 1], matrix[line][row]);

cout << "0 mutat in E" << endl;

Output\_quadratic\_matrix(size\_matrix, matrix);

swap(matrix[line][row + 1], matrix[line][row]);

}

}

int main()

{

int n, matrix[101][101]{};

cin >> n;

Input\_quadratic\_matrix(n, matrix);

if (Verifying\_element\_condition(n, matrix))

{

int row, line;

Finding\_position\_of\_0(n, matrix, line, row);

Swapping\_0(n, matrix, line, row);

}

}