МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп'ютерні інформаційні технології»

**Лабораторна робота №6**

**з дисципліни «Основи програмної інженерії»**

**на тему: «**Розробка програм із графічним інтерфейсом користувача**»**

Виконав:

студент гр.ПЗ1911

Сафонов Д.Є.

Прийняла:

Куроп'ятник О. С.

Дніпро, 2020

**Тема.** Розробка програм із графічним інтерфейсом користувача.

**Мета.** Отримати практичні навички розробки програм із графічним інтерфейсом користувача на основі механізму Windows Forms.

# **Постановка завдання.**

Для задачі лабораторної роботі №1 виконайте:

− побудуйте сценарій діалогу програми;

− створити програму з графічним інтерфейсом згідно розробленого сценарію.

Інтерфейс користувача має включати зазначені елементи та виконувати такі дії:

− поля для введення розмірності матриці (окремо для введення кількості рядків і стовбців, окрім випадків, коли обробка матриці пов’язана зі значеннями елементів діагоналі). Виконати перевірку коректності введених значень. При некоректному введені видати повідомлення користувачу з порадою щодо подальших дій для уникнення некоректних значень;

− таблицю, для введення та виведення матриці. Таблиця має дозволяти вводити та редагувати значення елементів. Передбачити перевірку коректних значень та виведення відповідних повідомлень для користувача;

− поле (поля) для виведення результатів обробки матриці;

− передбачати можливість багаторазової ініціалізації матриці (включаючи зміну розмірності) та обробки матриці;

− обробники подій елементів форми не мають містити коду виконання логіки програми, а лише виклики відповідних функцій.

17. Відсортуйте в матриці стовпці, що містять елементи, в запису яких є однакові цифри. Інші стовпці переверніть.

# **Сценарій діалогу програми.**

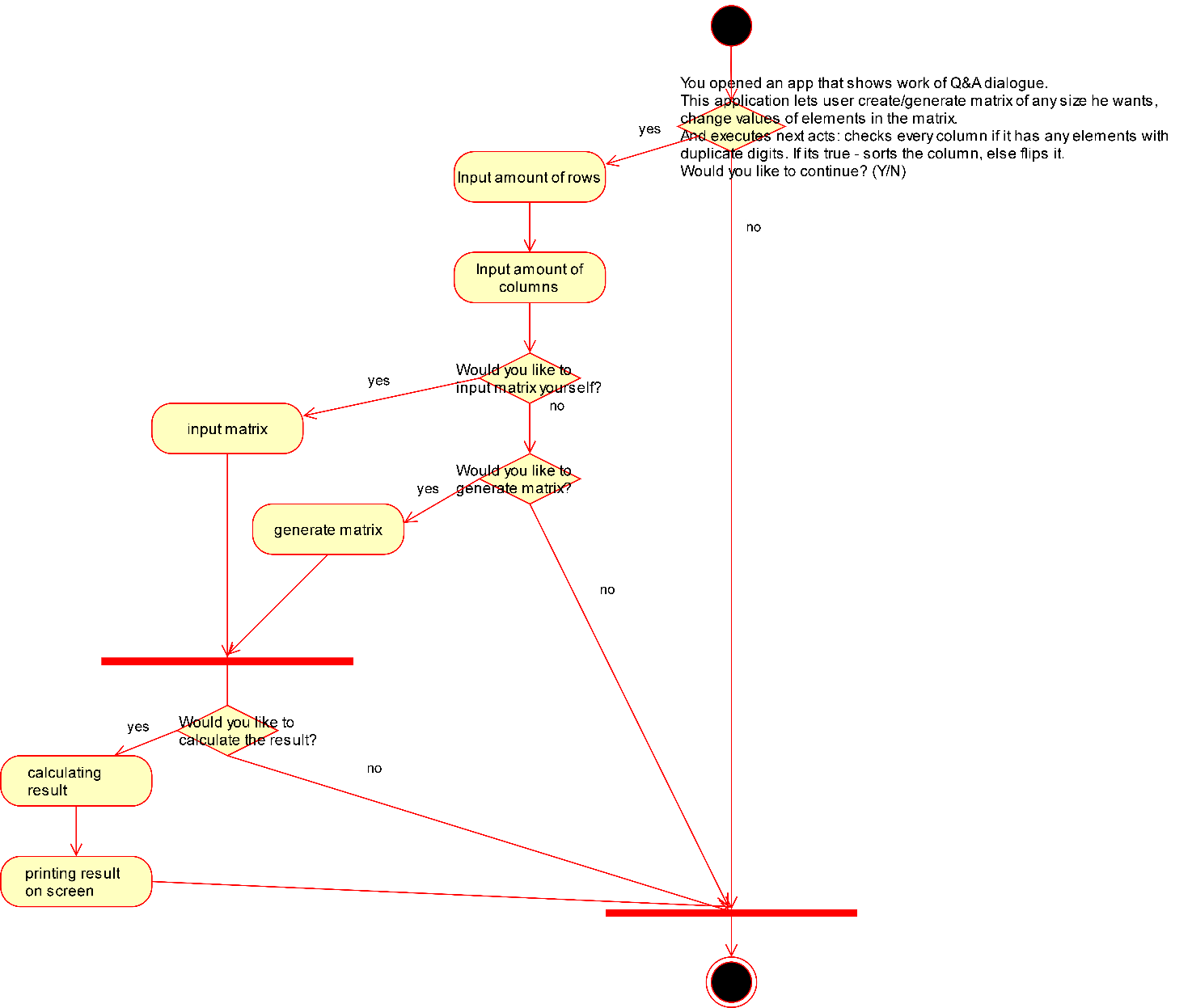


Рисунок 1

# **Набори тестів для перевірки працездатності програми.**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва | Вхідні дані | Очікуваний результат |
| 1 | Некоректні вхідні дані про кількість рядків/стовпців | “ff” | З'являється повідомлення про некоректний формат вводу |
| 2 | Некоректні вхідні дані значення клітки матриці | “ff” | З'являється повідомлення про некоректний формат вводу |
| 3 | Натискання кнопки встановлення розмірності матриці | - | На полі для вхідної та вихідної матриці з’являється відповідна кількість клітинок заповнених нулями, блокуються елементи встановлення розмірності, розблоковуються елементи розрахунка результату та генерації матриці. |
| 4 | Натискання кнопки генерації матриці | - | Вхідна матриця заповнюється випадковими значеннями |
| 5 | Натискання кнопки розрахунок результату | - | Вихідна матриця заповнюється відповідними значеннями. |
| 6 | Натискання кнопки перезапуску програми | - | Усі елементи повертаються до початкового стану. |

# **Код програми.**

“MyForm.h”

#pragma once

namespace opi6 {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Label^ greetingLabel;

protected:

private: System::Windows::Forms::Label^ rowsAmountLabel;

private: System::Windows::Forms::Label^ columnsAmountLabel;

private: System::Windows::Forms::Label^ matrixInLabel;

private: System::Windows::Forms::Label^ matrixOutLabel;

private: System::Windows::Forms::DataGridView^ matrixInDGV;

private: System::Windows::Forms::DataGridView^ matrixOutDGV;

private: System::Windows::Forms::TextBox^ columnsAmountTextBox;

private: System::Windows::Forms::TextBox^ rowsAmountTextBox;

private: System::Windows::Forms::Button^ setRowsColumnsButton;

private: System::Windows::Forms::Button^ genRandomMatrixInButton;

private: System::Windows::Forms::Button^ performExcerciseButton;

private: System::Windows::Forms::Label^ badRowsLabel;

private: System::Windows::Forms::Label^ badColumnsLabel;

private: System::Windows::Forms::Button^ restartButton;

private: System::Windows::Forms::Label^ badCellLabel;

protected:

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->greetingLabel = (gcnew System::Windows::Forms::Label());

this->rowsAmountLabel = (gcnew System::Windows::Forms::Label());

this->columnsAmountLabel = (gcnew System::Windows::Forms::Label());

this->matrixInLabel = (gcnew System::Windows::Forms::Label());

this->matrixOutLabel = (gcnew System::Windows::Forms::Label());

this->matrixInDGV = (gcnew System::Windows::Forms::DataGridView());

this->matrixOutDGV = (gcnew System::Windows::Forms::DataGridView());

this->columnsAmountTextBox = (gcnew System::Windows::Forms::TextBox());

this->rowsAmountTextBox = (gcnew System::Windows::Forms::TextBox());

this->setRowsColumnsButton = (gcnew System::Windows::Forms::Button());

this->genRandomMatrixInButton = (gcnew System::Windows::Forms::Button());

this->performExcerciseButton = (gcnew System::Windows::Forms::Button());

this->badRowsLabel = (gcnew System::Windows::Forms::Label());

this->badColumnsLabel = (gcnew System::Windows::Forms::Label());

this->restartButton = (gcnew System::Windows::Forms::Button());

this->badCellLabel = (gcnew System::Windows::Forms::Label());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->matrixInDGV))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->matrixOutDGV))->BeginInit();

this->SuspendLayout();

//

// greetingLabel

//

this->greetingLabel->AutoSize = true;

this->greetingLabel->Location = System::Drawing::Point(10, 10);

this->greetingLabel->Name = L"greetingLabel";

this->greetingLabel->Size = System::Drawing::Size(720, 34);

this->greetingLabel->TabIndex = 0;

this->greetingLabel->Text = L"This app lets you create/generate matrix.\nAnd performs next actions: check every "

L"column for elements with duplicate digits.If it has any - sorts it, else - flips"

L".";

//

// rowsAmountLabel

//

this->rowsAmountLabel->AutoSize = true;

this->rowsAmountLabel->Location = System::Drawing::Point(10, 57);

this->rowsAmountLabel->Name = L"rowsAmountLabel";

this->rowsAmountLabel->Size = System::Drawing::Size(101, 17);

this->rowsAmountLabel->TabIndex = 1;

this->rowsAmountLabel->Text = L"rowsAmount = ";

//

// columnsAmountLabel

//

this->columnsAmountLabel->AutoSize = true;

this->columnsAmountLabel->Location = System::Drawing::Point(10, 117);

this->columnsAmountLabel->Name = L"columnsAmountLabel";

this->columnsAmountLabel->Size = System::Drawing::Size(124, 17);

this->columnsAmountLabel->TabIndex = 2;

this->columnsAmountLabel->Text = L"columnsAmount = ";

//

// matrixInLabel

//

this->matrixInLabel->AutoSize = true;

this->matrixInLabel->Location = System::Drawing::Point(256, 316);

this->matrixInLabel->Name = L"matrixInLabel";

this->matrixInLabel->Size = System::Drawing::Size(56, 17);

this->matrixInLabel->TabIndex = 3;

this->matrixInLabel->Text = L"matrixIn";

//

// matrixOutLabel

//

this->matrixOutLabel->AutoSize = true;

this->matrixOutLabel->Location = System::Drawing::Point(522, 316);

this->matrixOutLabel->Name = L"matrixOutLabel";

this->matrixOutLabel->Size = System::Drawing::Size(68, 17);

this->matrixOutLabel->TabIndex = 4;

this->matrixOutLabel->Text = L"matrixOut";

//

// matrixInDGV

//

this->matrixInDGV->AllowUserToAddRows = false;

this->matrixInDGV->AllowUserToDeleteRows = false;

this->matrixInDGV->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->matrixInDGV->ColumnHeadersVisible = false;

this->matrixInDGV->Location = System::Drawing::Point(256, 57);

this->matrixInDGV->Name = L"matrixInDGV";

this->matrixInDGV->RowHeadersVisible = false;

this->matrixInDGV->RowHeadersWidth = 51;

this->matrixInDGV->RowTemplate->Height = 24;

this->matrixInDGV->Size = System::Drawing::Size(256, 256);

this->matrixInDGV->TabIndex = 5;

//

// matrixOutDGV

//

this->matrixOutDGV->AllowUserToAddRows = false;

this->matrixOutDGV->AllowUserToDeleteRows = false;

this->matrixOutDGV->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->matrixOutDGV->ColumnHeadersVisible = false;

this->matrixOutDGV->Location = System::Drawing::Point(522, 57);

this->matrixOutDGV->Name = L"matrixOutDGV";

this->matrixOutDGV->ReadOnly = true;

this->matrixOutDGV->RowHeadersVisible = false;

this->matrixOutDGV->RowHeadersWidth = 51;

this->matrixOutDGV->RowTemplate->Height = 24;

this->matrixOutDGV->Size = System::Drawing::Size(256, 256);

this->matrixOutDGV->TabIndex = 6;

//

// columnsAmountTextBox

//

this->columnsAmountTextBox->Location = System::Drawing::Point(142, 117);

this->columnsAmountTextBox->Name = L"columnsAmountTextBox";

this->columnsAmountTextBox->Size = System::Drawing::Size(100, 22);

this->columnsAmountTextBox->TabIndex = 7;

this->columnsAmountTextBox->TextChanged += gcnew System::EventHandler(this, &MyForm::columnsAmountTextBox\_TextChanged);

//

// rowsAmountTextBox

//

this->rowsAmountTextBox->Location = System::Drawing::Point(142, 57);

this->rowsAmountTextBox->Name = L"rowsAmountTextBox";

this->rowsAmountTextBox->Size = System::Drawing::Size(100, 22);

this->rowsAmountTextBox->TabIndex = 8;

this->rowsAmountTextBox->TextChanged += gcnew System::EventHandler(this, &MyForm::rowsAmountTextBox\_TextChanged);

//

// setRowsColumnsButton

//

this->setRowsColumnsButton->Enabled = false;

this->setRowsColumnsButton->Location = System::Drawing::Point(13, 194);

this->setRowsColumnsButton->Name = L"setRowsColumnsButton";

this->setRowsColumnsButton->Size = System::Drawing::Size(229, 40);

this->setRowsColumnsButton->TabIndex = 9;

this->setRowsColumnsButton->Text = L"set height and width";

this->setRowsColumnsButton->UseVisualStyleBackColor = true;

this->setRowsColumnsButton->Click += gcnew System::EventHandler(this, &MyForm::setRowsColumnsButton\_Click);

//

// genRandomMatrixInButton

//

this->genRandomMatrixInButton->Enabled = false;

this->genRandomMatrixInButton->Location = System::Drawing::Point(13, 262);

this->genRandomMatrixInButton->Name = L"genRandomMatrixInButton";

this->genRandomMatrixInButton->Size = System::Drawing::Size(229, 40);

this->genRandomMatrixInButton->TabIndex = 10;

this->genRandomMatrixInButton->Text = L"generate random MatrixIn";

this->genRandomMatrixInButton->UseVisualStyleBackColor = true;

this->genRandomMatrixInButton->Click += gcnew System::EventHandler(this, &MyForm::genRandomMatrixInButton\_Click);

//

// performExcerciseButton

//

this->performExcerciseButton->Enabled = false;

this->performExcerciseButton->Location = System::Drawing::Point(256, 378);

this->performExcerciseButton->Name = L"performExcerciseButton";

this->performExcerciseButton->Size = System::Drawing::Size(256, 47);

this->performExcerciseButton->TabIndex = 11;

this->performExcerciseButton->Text = L"compute MatrixOut";

this->performExcerciseButton->UseVisualStyleBackColor = true;

this->performExcerciseButton->Click += gcnew System::EventHandler(this, &MyForm::performExcerciseButton\_Click);

//

// badRowsLabel

//

this->badRowsLabel->AutoSize = true;

this->badRowsLabel->ForeColor = System::Drawing::Color::Red;

this->badRowsLabel->Location = System::Drawing::Point(81, 87);

this->badRowsLabel->Name = L"badRowsLabel";

this->badRowsLabel->Size = System::Drawing::Size(161, 17);

this->badRowsLabel->TabIndex = 12;

this->badRowsLabel->Text = L"must be positive integer!";

this->badRowsLabel->Visible = false;

//

// badColumnsLabel

//

this->badColumnsLabel->AutoSize = true;

this->badColumnsLabel->ForeColor = System::Drawing::Color::Red;

this->badColumnsLabel->Location = System::Drawing::Point(81, 147);

this->badColumnsLabel->Name = L"badColumnsLabel";

this->badColumnsLabel->Size = System::Drawing::Size(161, 17);

this->badColumnsLabel->TabIndex = 13;

this->badColumnsLabel->Text = L"must be positive integer!";

this->badColumnsLabel->Visible = false;

//

// restartButton

//

this->restartButton->Location = System::Drawing::Point(522, 378);

this->restartButton->Name = L"restartButton";

this->restartButton->Size = System::Drawing::Size(256, 47);

this->restartButton->TabIndex = 14;

this->restartButton->Text = L"restart";

this->restartButton->UseVisualStyleBackColor = true;

this->restartButton->Click += gcnew System::EventHandler(this, &MyForm::restartButton\_Click);

//

// badCellLabel

//

this->badCellLabel->AutoSize = true;

this->badCellLabel->ForeColor = System::Drawing::Color::Red;

this->badCellLabel->Location = System::Drawing::Point(332, 346);

this->badCellLabel->Name = L"badCellLabel";

this->badCellLabel->Size = System::Drawing::Size(95, 17);

this->badCellLabel->TabIndex = 15;

this->badCellLabel->Text = L"bad cell value";

this->badCellLabel->Visible = false;

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(787, 438);

this->Controls->Add(this->badCellLabel);

this->Controls->Add(this->restartButton);

this->Controls->Add(this->badColumnsLabel);

this->Controls->Add(this->badRowsLabel);

this->Controls->Add(this->performExcerciseButton);

this->Controls->Add(this->genRandomMatrixInButton);

this->Controls->Add(this->setRowsColumnsButton);

this->Controls->Add(this->rowsAmountTextBox);

this->Controls->Add(this->columnsAmountTextBox);

this->Controls->Add(this->matrixOutDGV);

this->Controls->Add(this->matrixInDGV);

this->Controls->Add(this->matrixOutLabel);

this->Controls->Add(this->matrixInLabel);

this->Controls->Add(this->columnsAmountLabel);

this->Controls->Add(this->rowsAmountLabel);

this->Controls->Add(this->greetingLabel);

this->Name = L"MyForm";

this->Text = L"MyForm";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->matrixInDGV))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->matrixOutDGV))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private:

bool goodRows = false, goodColumns = false;

int rowsAmount = 0, columnsAmount = 0;

int\*\* matrixIn = nullptr, \*\*matrixOut = nullptr;

int columnWidth = 30;//px

System::Void rowsAmountTextBox\_TextChanged(System::Object^ sender, System::EventArgs^ e);

System::Void columnsAmountTextBox\_TextChanged(System::Object^ sender, System::EventArgs^ e);

System::Void setRowsColumnsButton\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void genRandomMatrixInButton\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void performExcerciseButton\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void restartButton\_Click(System::Object^ sender, System::EventArgs^ e);

void updateMatrixInDGV();

void updateMatrixOutDGV();

};

}

“MyForm.cpp”

#include "MyForm.h"

#include "matrix.h"

#include <Windows.h>

using namespace opi6;///opi6

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MyForm);

return 0;

}

//checks if new Text can be parsed to Int, if not shows error msg

System::Void opi6::MyForm::rowsAmountTextBox\_TextChanged(System::Object^ sender, System::EventArgs^ e) {

bool isInt = Int32::TryParse(rowsAmountTextBox->Text, rowsAmount);

goodRows = (isInt && rowsAmount > 0);

setRowsColumnsButton->Enabled = (goodRows && goodColumns);//

badRowsLabel->Visible = !goodRows;

}

//checks if new Text can be parsed to Int, if not shows error msg

System::Void opi6::MyForm::columnsAmountTextBox\_TextChanged(System::Object^ sender, System::EventArgs^ e) {

bool isInt = Int32::TryParse(columnsAmountTextBox->Text, columnsAmount);

goodColumns = (isInt && columnsAmount > 0);

setRowsColumnsButton->Enabled = (goodRows && goodColumns);//

badColumnsLabel->Visible = !goodColumns;

}

//updates value of rowsAmount and columnsAmount from TextBoxes

System::Void opi6::MyForm::setRowsColumnsButton\_Click(System::Object^ sender, System::EventArgs^ e) {

matrixIn = createMatrix<int>(rowsAmount, columnsAmount);

setMatrix0(matrixIn, rowsAmount, columnsAmount);

matrixInDGV->RowCount = rowsAmount;

matrixInDGV->ColumnCount = columnsAmount;

updateMatrixInDGV();

matrixOut = createMatrix<int>(rowsAmount, columnsAmount);

setMatrix0(matrixOut, rowsAmount, columnsAmount);

matrixOutDGV->RowCount = rowsAmount;

matrixOutDGV->ColumnCount = columnsAmount;

updateMatrixOutDGV();

for (int j = 0; j < columnsAmount; j++) {//setWidth

matrixInDGV->Columns[j]->Width = columnWidth;

matrixOutDGV->Columns[j]->Width = columnWidth;

}

genRandomMatrixInButton->Enabled = true;

performExcerciseButton->Enabled = true;

rowsAmountTextBox->Enabled = false;

columnsAmountTextBox->Enabled = false;

setRowsColumnsButton->Enabled = false;

}

//generates new matrixIn and shows it on screen

System::Void opi6::MyForm::genRandomMatrixInButton\_Click(System::Object^ sender, System::EventArgs^ e) {

genMatrix(matrixIn, rowsAmount, columnsAmount);

updateMatrixInDGV();

}

//updates matrixOut from matrixIn

System::Void opi6::MyForm::performExcerciseButton\_Click(System::Object^ sender, System::EventArgs^ e) {

bool isInt = true;

for (int i = 0; i < rowsAmount && isInt; i++)//read matrixInDGV to matrixIn with validation

for (int j = 0; j < columnsAmount && isInt; j++)

isInt = Int32::TryParse(matrixInDGV->Rows[i]->Cells[j]->Value->ToString(), matrixOut[i][j]);

if (isInt) {

exercise(matrixOut, rowsAmount, columnsAmount);

updateMatrixOutDGV();

matrixInDGV->ReadOnly = true;

genRandomMatrixInButton->Enabled = false;

performExcerciseButton->Enabled = false;

badCellLabel->Visible = false;

}

else

badCellLabel->Visible = true;

}

//"restarts" app

System::Void opi6::MyForm::restartButton\_Click(System::Object^ sender, System::EventArgs^ e) {

if (matrixIn)

deleteMatrix(matrixIn, rowsAmount, columnsAmount);

if (matrixOut)

deleteMatrix(matrixOut, rowsAmount, columnsAmount);

genRandomMatrixInButton->Enabled = false;

performExcerciseButton->Enabled = false;

columnsAmountTextBox->Enabled = true;

rowsAmountTextBox->Enabled = true;

columnsAmountTextBox->Text = "";

rowsAmountTextBox->Text = "";

badRowsLabel->Visible = false;

badColumnsLabel->Visible = false;

matrixInDGV->RowCount = 0;

matrixOutDGV->RowCount = 0;

matrixInDGV->ColumnCount = 0;

matrixOutDGV->ColumnCount = 0;

rowsAmount = columnsAmount = 0;

}

//updates cell of matrixInDGV from matrixIn

void opi6::MyForm::updateMatrixInDGV() {

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

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

matrixInDGV->Rows[i]->Cells[j]->Value = matrixIn[i][j];

}

//updates cell of matrixOutDGV from matrixOut

void opi6::MyForm::updateMatrixOutDGV() {

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

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

matrixOutDGV->Rows[i]->Cells[j]->Value = matrixOut[i][j];

}

“matrix.h”

#ifndef \_\_MATRIX\_H\_\_

#define \_\_MATRIX\_H\_\_

#include <iostream>

#include <iomanip>

#include <random>

#include <chrono>

#include <bitset>

#include <string>

//allocate memory for matrix

template<typename T>

T\*\* createMatrix(int rowsAmount, int columnsAmount) {//allocate memory for matrix

T\*\* matrix2d = new T \* [rowsAmount];

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

matrix2d[i] = new T[columnsAmount];

return matrix2d;

}

//free up the memory

template<typename T>

void deleteMatrix(T\*\* matrix2d, int rowsAmount, int columnsAmount) {

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

delete[]matrix2d[i];

delete[]matrix2d;

}

//set random values for matrix

template<typename T>

void genMatrix(T\*\* matrix2d, int rowsAmount, int columnsAmount) {

typedef std::chrono::high\_resolution\_clock myclock;

myclock::time\_point beginning = myclock::now();

std::uniform\_int\_distribution<int> valueDistribution(0, 100);

myclock::duration d = myclock::now() - beginning;

unsigned seed = d.count();

std::default\_random\_engine generator(seed);

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

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

matrix2d[i][j] = valueDistribution(generator);

}

template<typename T>

void printMatrix(T\*\* matrix2d, int rowsAmount, int columnsAmount) {

std::cout << std::left;

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

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

std::cout << std::setw(12) << matrix2d[i][j] << " ";

std::cout << std::endl;

}

std::cout << std::right;

}

//sorts column using insertion sort algorithm

template <typename T>

void sortColumn(T\*\* matrix2d, int column, int rowsAmount) {

for (int i = 1; i < rowsAmount; i++)

if (matrix2d[i][column] < matrix2d[i - 1][column]) {//insertionSort

T temp = matrix2d[i][column];

int j = i - 1;

for (j; (j >= 0) && (temp < matrix2d[j][column]); j--)

matrix2d[j + 1][column] = matrix2d[j][column];//find position and shift to right elements between new and old position

matrix2d[j + 1][column] = temp;

}

}

template <typename T>

void flipColumn(T\*\* matrix2d, int column, int rowsAmount) {

for (int i = 0, j = rowsAmount - 1; i <= j; i++, j--) {

T temp = matrix2d[i][column];

matrix2d[i][column] = matrix2d[j][column];

matrix2d[j][column] = temp;

}

}

bool hasDuplicateDigits(int a) {

bool retValue = false;

a = abs(a);

if (a > 999999999)//>10digits, at least one is duplicate

retValue = true;

else if (a > 10) {//if >= 2digits

std::bitset<10> digitsUsed;

int currentDigit;

do {

currentDigit = a % 10;

a = (a - currentDigit) / 10;

if (digitsUsed[currentDigit])

retValue = true;

else

digitsUsed[currentDigit] = true;

} while (a > 0 && !retValue);

}

return retValue;

}

//checks if column has any elemnts with duplicate digits

template <typename T>

bool columnElementsDuplicateDigits(T\*\* matrix2d, int column, int rowsAmount) {

bool has = false;

T currentElement;

for (int i = 0; i < rowsAmount && !has; i++)//has = if matrix2d[i][column] has >= 2 same numerals

has = hasDuplicateDigits(matrix2d[i][column]);

return has;

}

//performs a check on every column and sorts it if it has any elements with duplicate digits, flips if not

template <typename T>

void exercise(T\*\* matrix2d, int rowsAmount, int columnsAmount) {

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

if (columnElementsDuplicateDigits(matrix2d, i, rowsAmount))

sortColumn(matrix2d, i, rowsAmount);

else

flipColumn(matrix2d, i, rowsAmount);

}

int inputNum(int leftLim, int rightLim, std::string invitation) {

std::string cinStr;//////////////////////////////////////////

int num = leftLim - 1;

size\_t\* pos = new size\_t{ 0 };

do {//input validation

std::cout << invitation;

std::cin >> cinStr;

if (std::cin.peek() != '\n') {//if istream isn't empty

while (std::cin.get() != '\n');//clear istream

std::cout << "incorrect format" << std::endl;

}

else {

\*pos = 0;//clear old

try {

num = std::stoi(cinStr, pos);//convert input to INT format, set \*pos value to length of used substring

}

catch (std::invalid\_argument) {

std::cout << "incorrect format" << std::endl;

num = leftLim - 1;

}

catch (std::out\_of\_range) {

std::cout << "out of range" << std::endl;

num = leftLim - 1;

}

if (\*pos != 0)

if (\*pos < cinStr.size()) {

std::cout << "incorrect format" << std::endl;

num = leftLim - 1;

}

else if (num < leftLim || num >= rightLim) {

std::cout << "out of range" << std::endl;

num = leftLim - 1;

}

}

} while (num < leftLim || num >= rightLim);

return num;

}

//lets user to chnge any elements by indexes

template <typename T>

void setMatrix(T\*\* matrix2d, int rowsAmount, int columnsAmount) {

system("cls");

printMatrix(matrix2d, rowsAmount, columnsAmount);

char choice;

std::cout << "Do you want to enter any elements?(Y/N)";

std::cin >> choice;

while ((choice != 'Y' && choice != 'y' && choice != 'N' && choice != 'n') || (std::cin.peek() != '\n')) {

while (std::cin.get() != '\n');//clear istream

std::cout << "wrong input, try again." << std::endl;

std::cout << "Do you want to enter any elements?(Y/N)";

std::cin >> choice;

}

while (choice != 'N' && choice != 'n') {

int i = inputNum(0, rowsAmount, "enter row of element you want to change: ");

int j = inputNum(0, columnsAmount, "enter column of element you want to change: ");

std::string elementName = "A[" + std::to\_string(i) + "][" + std::to\_string(j) + "] = ";

matrix2d[i][j] = inputNum(-32768, 32767, elementName);

system("cls");

printMatrix(matrix2d, rowsAmount, columnsAmount);

std::cout << "Do you want to enter any elements?(Y/N)";

std::cin >> choice;

while ((choice != 'Y' && choice != 'y' && choice != 'N' && choice != 'n') || (std::cin.peek() != '\n')) {

while (std::cin.get() != '\n');//clear istream

std::cout << "wrong input, try again." << std::endl;

std::cout << "Do you want to enter any more elements?(Y/N)";

std::cin >> choice;

}

}

}

//set all elements to 0

template <typename T>

void setMatrix0(T\*\* matrix2d, int rowsAmount, int columnsAmount) {

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

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

matrix2d[i][j] = 0;

}

#endif

# **Продемонструвати та прокоментувати роботу програми на скринах.**

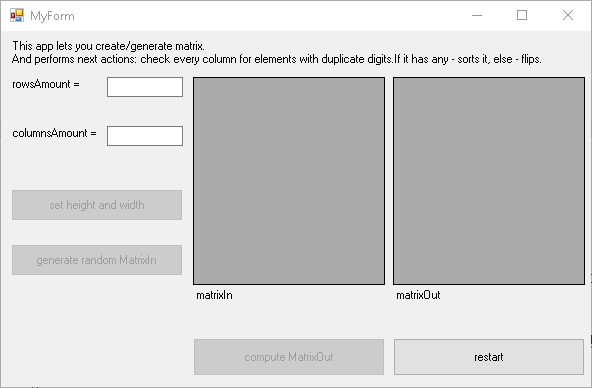


Рисунок 2

Початковий стан програми.

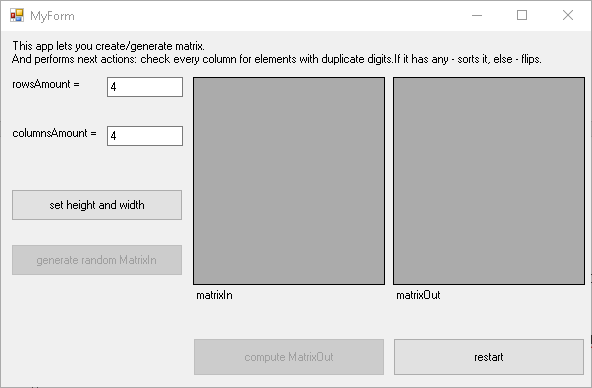


Рисунок 3

При введені коректних значень розмірності розблоковується кнопка її встановлення.

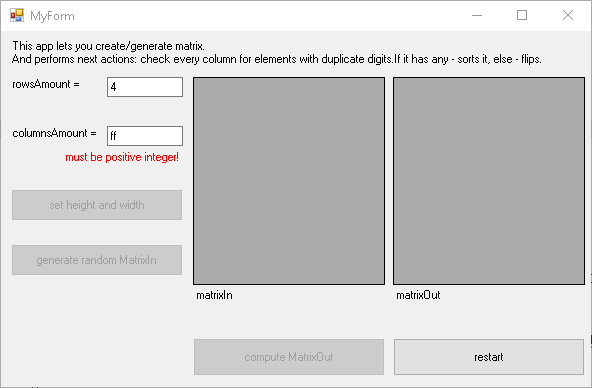


Рисунок 4

Якщо введені некоректні дані, кнопка блокується та з’являється повідомлення.

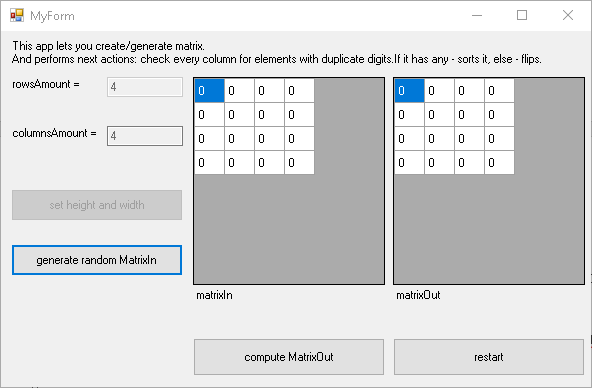


Рисунок 5

При встановлені розмірності, з’являються матриці заповнені нулями, вихідну матрицю не можна змінювати самостійно.

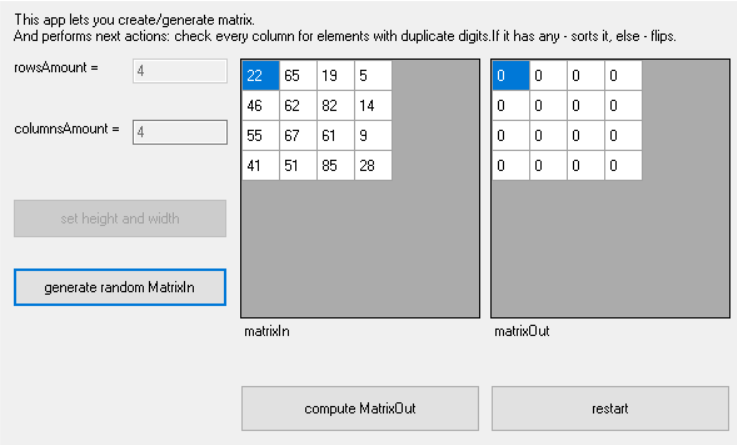


Рисунок 6

Генерація/введення вхідної матриці.

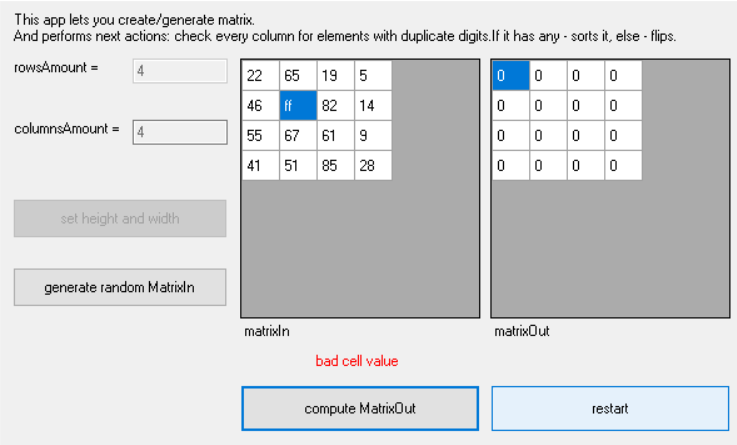


Рисунок 7

Якщо ввести некоректне значення та спробувати порахувати результат з’явиться повідомлення про помилку.

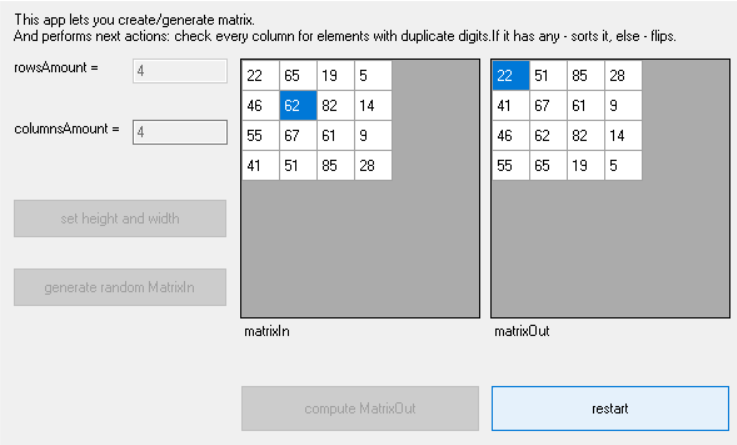


Рисунок 8

Розрахунок результату.

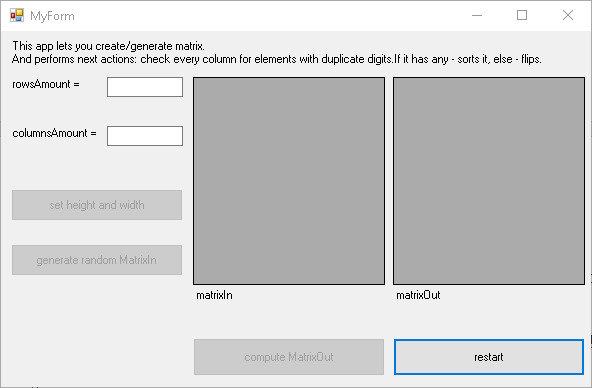


Рисунок 9

“Перезапуск” програми.

# **Висновки щодо особливостей розробки програми із графічним інтерфейсом.**

Розробляти інтерфейс користувача з допомогою вже існуючих фреймворків дуже зручно та відносно просто, бо взаємовідносини об’єктів вже розроблені і програмісту потрібно лише використати ці об’єкти, доповнюючи їх своїм кодом. Також подібні фреймворки зручні бо їх можна використовувати при розробці на різних мовах, а іноді навіть різних платформах. Також такі програми набагато зручніші в використанні ніж програми з подібним функціоналом виконані в консолі, бо в консолі можна здійснити тільки лінійний діалог.