***Лабораторна робота №2***

*Автор: Тукало Іван*

***Завдання 1(MyMatrix)***

Реалізуйте клас MyMatrix, який зберігає прямокутну числову матрицю і вміє виконувати дії додавання матриць та множення матриць.

Матриця повинна містити єдине поле рівня доступу private або protected — масив із власне елементами матриці (типу double[,]) .

Слід реалізувати конструктори, оператори, властивості та методи (усі публічні, якщо тільки не вказано іншого). Слід зробити цей клас partial, розбивши на два .cs:-файли:

Файл MatrixData.cs:

* Конструктори:
* копіюючий з іншого примірника цього самого класу MyMatrix;
* з двовимірного масиву типу double[,];
* з «зубчастого» масиву double-ів, якщо він фактично прямокутний;
* з масиву рядків, якщо фактично ці рядки містять записані через пробіли та/або числа, а кількість цих чисел у різних рядках однакова.
* з рядка, що містить як пробіли та/або табуляції (їх трактувати як роздільники чисел у одному рядку матриці), так і символи переведення рядка (їх трактувати як роздільники рядків) — якщо фактичні дані того рядка утворюють прямокутну числову матрицю; зокрема, щоб цим конструктором можна було створити матрицю з рядка, раніше сформованого методом ToString (див. далі))
* Властивості (Properties) Height та Width, що дозволяють взнати (але не дозволяють змінити) «висоту» (кількість рядків) та «ширину» (кількість стовпчиків)
* Java-style getter-и (без setter-ів) кількості рядків getHeight() та кількості стовпчиків getWidth()
* Індексатори, що дозволяють публічно доступатися до будь-якого окремого елемента матриці (як взнавати його значення, так і змінювати)
* Java-style getter та setter для окремого елемента (getter має два параметри — номер рядка і номер стовпчика, setter має три параметра — номер рядка, номер стовпчика, і значення, яке записати у той рядок і стовпчик)
* override public String ToString(), який формуватиме (табуляціями та переведеннями рядка) зручне для сприйняття людиною прямокутне подання числової матриці; метод повинен мати саме вищенаведений заголовок, бо саме так вдається забезпечити, щоб примірники цього класу можна було виводити просто через Console.WriteLine(A) (де A – примірник цього самого класу MyMatrix)

Файл MatrixOperations.cs:

* operator + додавання двох матриць (лише якщо вони мають однаковий розмір)
* operator \* множення двох матриць (лише якщо кількість стовпчиків першої дорівнює кількості рядків другої)
* Метод (не статичний; private або protected) GetTransponedArray(), що повертає новостворений масив double[,] (не MyMatrix, а просто масив), у якому вміст елементів транспонований відносно тієї матриці, для якої він викликався
* Метод (не статичний) MyMatrix GetTransponedCopy(), який би створював новий примірник MyMatrix, у якому вміст матриці транспонований відносно тієї, для якої він викликався; технічну роботу зі власне транспонування не повторювати, а використати результат GetTransponedArray()
* Метод (не статичний) void TransponeMe(), який би замінював матрицю, для якої викликається, на транспоновану (теж використати GetTransponedArray(), але щоб у результаті змінився сам this-примірник MyMatrix).

В усіх випадках неправильних даних (намагання створити матрицю з фактично не прямокутного «зубчастого» масиву; намагання створити матрицю з масиву рядків, який фактично містить взагалі не числа; намагання доступитися до неіснуючого елемента; намагання додати матриці різного розміру, тощо) пропонується, щоб виконання програми просто аварійно завершувалося. Рятування ситуації шляхом коректної роботи з виключними cитуаціями (exception-ами) – правильний підхід, вітається (хоч і не вимагається). Намагання нівелювати ці помилки будь-якими іншими засобами забороняються.

Частина завдання на бонусні бали; її можна не робити, але якщо робити, то захищати, а просто мовчки здавати, не захищаючи, заборонено (хто так зробить — вважається, що списа(в/ла), бали замість збільшитися стрімко зменшаться):

• Додати у MatrixOperations.cs також метод double CalcDeterminant() (не статичний), який би обчислював детермінант (визначник), лише для квадратної матриці, не змінюючи вміст матриці. (Зрозуміло, доведеться створити копію масиву і змінювати вміст цієї копії.) Оскільки обчислення детермінанта є вельми повільною операцією, зробити запам’ятоване й закешоване значення визначника, щоб не витрачати багато часу на обчислення вже обчисленого. (Для цього слід порушити другий абзац цієї постановки задачі, де сказано, що масив із власне елементами матриці — єдине поле класу; однак, повне виконання цього бонусного завдання є єдиною дозволеною причиною порушувати ту вимогу.) Додаткову складність дає також те, що матриця не immutable, а може змінюватися, тому після будь-якої зміни вмісту матриці не можна користуватися раніше обчисленим значенням детермінанту. Як врахувати це — визначити самостійно, але все це повинно бути поєднаним із розділенням класу (підтримкою його як partial), причому з можливістю включати в якийсь проект лише MatrixData.cs без MatrixOperations.cs

***Завдання 2(My\_time ООП)***

Взявши за основу завдання блок 1 лабораторної «структури та рядки–1» за 2-й семестр 1-го курсу (де про структури my\_time та my\_frac), реалізуйте клас MyTime або MyFrac відповідно (розподіл варіантів вказується викладачем окремо), що має описану функціональність. Деталі вимог щодо всяких різних деталей переосмислити самостійно, згідно того, що там просили зробити структуру й не просили дотримуватися ООП-підходу, а тепер треба зробити клас, дотримуючись ООП-підходу.

Ще раз: Ваше завдання для цієї лабораторної з ООП включає в себе необхідність вирішити, де з точки зору ООП слід робити інакше, а де краще просто дотриматися вимог, сформульованих у тій лабораторній для першого курсу.

Завдання з 1 курсу:

Структура

struct MyTime {

public int hour, minute, second;

public MyTime(int h, int m, int s)

{

// вміст допишіть самостійно

}

public override string ToString()

{

// вміст допишіть самостійно

}

};

подає час всередині доби.

Реалізувати функції (методи):

• конструктор MyTime(int h, int m, int s), який створює структуру за наданими годиною, хвилиною і секундою; цей конструктор, як і будь-який конструктор, має бути оголошений всередині структури, як вказано вище;

• public override string ToString(), всередині структури, як вказано вище (див. також спільний текст обох варіантів) — формуватиме рядкове подання цього моменту, у форматі "h:mm:ss", тобто h (від 0 до 23) може бути однією цифрою (від 0 до 9), а може і двома, а от хвилини mm і секунди ss завжди двома цифрами (в разі потреби починаючись із нуля); наполегливо рекомендується не писати включення цих стартових нулів if’ами чи якимись подібними засобами вручну, а знайти, як зробити це стандартними засобами/

Всі подальші функції (методи) робити не всередині цієї структури, а в основному класі (який, якщо Ви нічого не змінювали, називається class Program). Це порушує класичний ООП-підхід, але не будемо знову повторювати все, вже сказане у спільній для обох варіантів частині:

• static int TimeSinceMidnight(MyTime t) — перетворюватиме вказаний час (вигляду MyTime) у кількість секунд, що пройшли від початку доби;

• static MyTime TimeSinceMidnight(int t) — перетворюватиме кількість секунд, що пройшли від початку доби, у час у форматі MyTime;

• static MyTime AddOneSecond(MyTime t) — додаватиме до структури одну секунду (у цій функції (методі) та кількох подальших аналогічних, у ситуаціях, коли відбувся перехід за межі доби — рахувати час всередині нової доби, тобто просто приводити h у проміжок від 0 до 23, ніяк не позначаючи, що доба інша; така поведінка має дещо спільне з переповненням, але вимагається зробити саме так);

• static MyTime AddOneMinute(MyTime t) — додаватиме до структури одну хвилину;

• static MyTime AddOneHour(MyTime t) — додаватиме до структури одну годину;

• static MyTime AddSeconds(MyTime t, int s) — додаватиме до структури вказану кількість секунд s (яка може бути довільною, в тому числі більшою 60, і навіть більшою 24×60×60 = 86400, а також від’ємною);

• static int Difference(MyTime mt1, MyTime mt2) — вертатиме різницю між двома моментами, заданими структурами MyTime (припускаємо, ніби вони гарантовано стосуються однієї доби; результат може бути як додатній (mt1 пізніше mt2), так і від'ємний (mt1 раніше mt2) або нульовий).

• static string WhatLesson(MyTime mt) — формуватиме рядок, в якому вказуватиме, згідно нашого реального розкладу дзвінків, одне зі значень «пари ще не почалися» , або «1-а пара», або «перерва між 1-ю та 2-ю парами», або «2-а пара», або «перерва між 2-ю та 3-ю парами», або «3-я пара», або «перерва між 3-ю та 4-ю парами», або «4-а пара», або «перерва між 4-ю та 5-ю парами», або «5-а пара», або «перерва між 5-ю та 6-ю парами», або «6-а пара», або «пари вже скінчилися». Факти, що бувають вихідні та що в деякі дні можуть бути не всі пари, проігнорувати (вважати, ніби завжди пар рівно 6).

У цьому варіанті, значну частину функцій (методів) треба реалізувати так, щоб вони використовували (викликали) інші функції (методи). Але так, щоб не виникало аж надто істотного сповільнення виконання. Наприклад, прямо заборонено робити AddOneMinute через 60-кратний повтор AddOneSecond, але заборонено не лише конкретно це, а й інші дуже вже неефективні (в десятки разів повільніші за можливі альтернативи) вираження.

Але не варто перейматися, наприклад, тим, що доводиться зробити шість множень&ділень замість трьох. Якщо можна справді спростити код, або зробити його зрозумілішим, чи більш узгодженим, але код працюватиме у 2–5 разів повільніше — варто так і вчинити. Нарешті, рекомендованою є, наприклад, така реалізація одного зі згаданих методів:

static MyTime TimeSinceMidnight(int t)

{

int secPerDay = 60\*60\*24;

t %= secPerDay;

// приводимо t до проміжку, можливого в межах однієї доби,

// враховуючи, що початкове значення t може бути й від’ємним

if (t < 0)

t += secPerDay;

int h = t / 3600;

int m = (t / 60) % 60;

int s = t % 60;

return new MyTime(h, m, s);

}

(можна прямо її і включити у свій розв’язок).

***Код(2 блока в одній прогамі):***

Program:

using System;

using System.Text;

using static MyMatrix;

partial class Program

{

static void DoBlock\_1()

{

Console.WriteLine("Введіть першу матрицю у вигляді чисел, розділених пробілами і символами переходу на новий рядок:");

var matrix1 = ReadMatrix();

Console.WriteLine("\nВведіть другу матрицю у вигляді чисел, розділених пробілами і символами переходу на новий рядок:");

var matrix2 = ReadMatrix();

Console.WriteLine("\nВведені матриці:");

Console.WriteLine("Матриця 1:");

Console.WriteLine(matrix1);

Console.WriteLine("Матриця 2:");

Console.WriteLine(matrix2);

Console.WriteLine("\n=== Виберіть дію ===");

Console.WriteLine("1 - Додати матриці");

Console.WriteLine("2 - Помножити матриці");

Console.WriteLine("3 - Транспонувати першу матрицю");

Console.WriteLine("4 - Транспонувати другу матрицю");

Console.WriteLine("5 - Обчислити визначник першої матриці");

Console.WriteLine("6 - Обчислити визначник другої матриці");

Console.WriteLine("7 - Змінити значення першої матриці");

Console.WriteLine("8 - Змінити значення другої матриці");

Console.WriteLine("0 - Вихід");

while (true)

{

Console.Write("\nВаш вибір: ");

string choice = Console.ReadLine();

try

{

switch (choice)

{

case "1":

Console.WriteLine("\nРезультат додавання:");

Console.WriteLine(matrix1 + matrix2);

break;

case "2":

Console.WriteLine("\nРезультат множення:");

Console.WriteLine(matrix1 \* matrix2);

break;

case "3":

Console.WriteLine("\nТранспонована перша матриця:");

Console.WriteLine(matrix1.GetTransponedCopy());

break;

case "4":

Console.WriteLine("\nТранспонована друга матриця:");

Console.WriteLine(matrix2.GetTransponedCopy());

break;

case "5":

Console.WriteLine("\nВизначник першої матриці:");

Console.WriteLine(matrix1.CalcDeterminant());

break;

case "6":

Console.WriteLine("\nВизначник другої матриці:");

Console.WriteLine(matrix2.CalcDeterminant());

break;

case "7":

Console.WriteLine("\nЗміна значення у першій матриці:");

ModifyMatrix(matrix1);

break;

case "8":

Console.WriteLine("\nЗміна значення у другій матриці:");

ModifyMatrix(matrix2);

break;

case "0":

Console.WriteLine("\nЗавершую...");

return;

default:

Console.WriteLine("Некоректний вибір. Спробуйте ще раз.");

break;

}

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

}

static MyMatrix ReadMatrix()

{

while (true)

{

Console.Write("Введіть матрицю (порожній рядок для завершення вводу):\n");

string input = ReadMultilineInput();

try

{

return new MyMatrix(input);

}

catch (Exception ex)

{

Console.WriteLine($"Помилка створення матриці: {ex.Message}");

Console.WriteLine("Спробуйте ще раз.");

}

}

}

static string ReadMultilineInput()

{

var inputBuilder = new System.Text.StringBuilder();

string line;

while (!string.IsNullOrWhiteSpace(line = Console.ReadLine()))

{

inputBuilder.AppendLine(line);

}

return inputBuilder.ToString();

}

static void DoBlock\_2()

{

MyTime t1 = GetTime("t1");

MyTime t2 = GetTime("t2");

Console.WriteLine($"t1: {t1}");

Console.WriteLine($"t2: {t2}");

Console.WriteLine("Час t1: " + t1);

Console.WriteLine("Час t2: " + t2);

Console.WriteLine("Різниця між t1 і t2: " + MyTime.Difference(t1, t2) + " секунд");

Console.WriteLine("t1 + 1 секунда: " + t1.AddOneSecond());

Console.WriteLine("t1 + 1 хвилина: " + t1.AddOneMinute());

Console.WriteLine("t1 + 1 година: " + t1.AddOneHour());

Console.WriteLine("Виберіть метод введення:");

Console.WriteLine("1. Введення вручну");

Console.WriteLine("2. Випадковий вибір");

int choice = int.Parse(Console.ReadLine());

int totalSeconds = 0;

if (choice == 1)

{

// Користувач вводить кількість секунд вручну

Console.WriteLine("Введіть кількість секунд:");

totalSeconds = int.Parse(Console.ReadLine());

}

else if (choice == 2)

{

// Користувач вводить мінімум і максимум для випадкового вибору

Console.WriteLine("Введіть мінімум для секунд:");

int minSeconds = int.Parse(Console.ReadLine());

Console.WriteLine("Введіть максимум для секунд:");

int maxSeconds = int.Parse(Console.ReadLine());

Random rand = new Random();

totalSeconds = rand.Next(minSeconds, maxSeconds);

Console.WriteLine($"Випадковий вибір: {totalSeconds} секунд");

}

Console.WriteLine("t1 + " + totalSeconds + " секунд: " + t1.AddSeconds(totalSeconds));

Console.WriteLine(MyTime.WhatLesson());

}

static MyTime GetTime(string label)

{

Console.WriteLine($"Як ви хочете задати час для {label}? (1 - вручну, 2 - випадковий):");

string choice = Console.ReadLine();

if (choice == "1")

{

return MyTime.InputFromConsole(label);

}

else if (choice == "2")

{

return MyTime.GenerateRandomTime();

}

else

{

Console.WriteLine("Невірний вибір! Буде використано випадковий час.");

return MyTime.GenerateRandomTime();

}

}

class MyTime

{

public int Hour { get; private set; }

public int Minute { get; private set; }

public int Second { get; private set; }

public MyTime(int hour, int minute, int second)

{

if (hour < 0 || hour >= 24 || minute < 0 || minute >= 60 || second < 0 || second >= 60)

throw new ArgumentException("Неприпустимий час.");

Hour = hour;

Minute = minute;

Second = second;

}

public override string ToString()

{

return $"{Hour:D2}:{Minute:D2}:{Second:D2}";

}

public static MyTime InputFromConsole(string label)

{

Console.WriteLine($"Введіть час для {label} (година хвилина секунда, через пробіл):");

string[] parts = Console.ReadLine().Split();

int h = int.Parse(parts[0]);

int m = int.Parse(parts[1]);

int s = int.Parse(parts[2]);

return new MyTime(h, m, s);

}

public static MyTime GenerateRandomTime()

{

Random random = new Random();

int h = random.Next(0, 24);

int m = random.Next(0, 60);

int s = random.Next(0, 60);

return new MyTime(h, m, s);

}

// Нестатичний метод для обчислення секунд з початку доби

public int TimeSinceMidnight()

{

return Hour \* 3600 + Minute \* 60 + Second;

}

// Статичний метод залишаємо для сумісності

public static MyTime TimeSinceMidnight(int seconds)

{

const int secPerDay = 86400;

seconds %= secPerDay;

if (seconds < 0) seconds += secPerDay;

int h = seconds / 3600;

int m = (seconds / 60) % 60;

int s = seconds % 60;

return new MyTime(h, m, s);

}

public MyTime AddOneSecond() => AddSeconds(1);

public MyTime AddOneMinute() => AddSeconds(60);

public MyTime AddOneHour() => AddSeconds(3600);

public MyTime AddSeconds(int seconds)

{

int totalSeconds = TimeSinceMidnight() + seconds;

return TimeSinceMidnight(totalSeconds);

}

public static int Difference(MyTime t1, MyTime t2)

{

return t1.TimeSinceMidnight() - t2.TimeSinceMidnight();

}

public static string WhatLesson()

{

MyTime now = GetTime("now");

Console.WriteLine($"Зараз: {now}");

int totalSeconds = now.TimeSinceMidnight();

if (totalSeconds < new MyTime(8, 0, 0).TimeSinceMidnight())

return "Пари ще не почалися";

else if (totalSeconds < new MyTime(9, 20, 0).TimeSinceMidnight())

return "1-а пара";

else if (totalSeconds < new MyTime(9, 40, 0).TimeSinceMidnight())

return "Перерва між 1-ю та 2-ю парами";

else if (totalSeconds < new MyTime(11, 0, 0).TimeSinceMidnight())

return "2-а пара";

else if (totalSeconds < new MyTime(11, 20, 0).TimeSinceMidnight())

return "Перерва між 2-ю та 3-ю парами";

else if (totalSeconds < new MyTime(12, 40, 0).TimeSinceMidnight())

return "3-я пара";

else if (totalSeconds < new MyTime(13, 0, 0).TimeSinceMidnight())

return "Перерва між 3-ю та 4-ю парами";

else if (totalSeconds < new MyTime(14, 20, 0).TimeSinceMidnight())

return "4-а пара";

else if (totalSeconds < new MyTime(15, 40, 0).TimeSinceMidnight())

return "Перерва між 4-ю та 5-ю парами";

else if (totalSeconds < new MyTime(16, 0, 0).TimeSinceMidnight())

return "5-а пара";

else if (totalSeconds < new MyTime(16, 20, 0).TimeSinceMidnight())

return "Перерва між 5-ю та 6-ю парами";

else if (totalSeconds < new MyTime(17, 40, 0).TimeSinceMidnight())

return "6-а пара";

else

return "Пари вже скінчилися";

}

}

static void Main(string[] args)

{

System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");

Console.OutputEncoding = UTF8Encoding.UTF8;

Console.ForegroundColor = ConsoleColor.Black;

Console.BackgroundColor = ConsoleColor.White;

Console.Clear();

int choice;

do

{

Console.WriteLine("-Для виконання блоку 1 - MyMatrix");

Console.WriteLine("-Для виконання блоку 2 - my\_time ООП");

choice = int.Parse(Console.ReadLine());

switch (choice)

{

case 1:

Console.WriteLine("Виконую блок 1 - MyMatrix");

DoBlock\_1();

Console.WriteLine();

break;

case 2:

Console.WriteLine("Виконую блок 2 - my\_time ООП");

DoBlock\_2();

Console.WriteLine();

break;

case 0:

Console.WriteLine("Зараз завершимо, тільки натисніть будь ласка ще раз Enter");

Console.ReadLine();

break;

default:

Console.WriteLine("Команда ``{0}'' не розпізнана. Зробіь, будь ласка, вибір із 1, 2 і 0.", choice);

Console.WriteLine();

break;

}

} while (choice != 0);

}

}

MatrixData:  
  
using System;

using System.Text;

public partial class MyMatrix

{

protected double[,] matrix;

protected double? ChanceDeterminant = null;

public bool isMod = true;

// Конструктор копіювання

public MyMatrix(MyMatrix other)

{

if (other == null)

throw new ArgumentNullException(nameof(other));

InitializeMatrix(other.matrix);

}

// Конструктор з двовимірного масиву

public MyMatrix(double[,] data)

{

if (data == null)

throw new ArgumentNullException(nameof(data));

InitializeMatrix(data);

}

// Конструктор з зубчастого масиву

public MyMatrix(double[][] data)

{

if (data == null)

throw new ArgumentNullException(nameof(data));

ValidateJaggedArray(data);

matrix = new double[data.Length, data[0].Length];

FillMatrixFromJaggedArray(data);

}

// Конструктор з масиву рядків

public MyMatrix(string[] rows)

{

if (rows == null || rows.Length == 0)

throw new ArgumentException("Рядки не можуть бути порожніми.");

matrix = ParseRowsToMatrix(rows);

InvalidateCache();// Очищення кешу після зміни

}

// Конструктор з одного рядка, який делегує конструктору з масиву рядків

public MyMatrix(string input) : this(input?.Split(new[] { '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries))

{

if (string.IsNullOrWhiteSpace(input))

throw new ArgumentException("Вхідні дані не можуть бути порожніми.");

}

// Приватний метод для ініціалізації матриці

private void InitializeMatrix(double[,] source)

{

int rows = source.GetLength(0);

int cols = source.GetLength(1);

matrix = new double[rows, cols];

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

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

matrix[i, j] = source[i, j];

InvalidateCache();// Очищення кешу після зміни

}

// Приватний метод для перевірки зубчастого масиву

private void ValidateJaggedArray(double[][] data)

{

int cols = data[0].Length;

foreach (var row in data)

{

if (row.Length != cols)

throw new ArgumentException("Масив не є прямокутним.");

}

}

// Приватний метод для заповнення матриці із зубчастого масиву

private void FillMatrixFromJaggedArray(double[][] data)

{

for (int i = 0; i < data.Length; i++)

for (int j = 0; j < data[i].Length; j++)

matrix[i, j] = data[i][j];

InvalidateCache();// Очищення кешу після зміни

}

// Приватний метод для парсингу рядків у матрицю

private double[,] ParseRowsToMatrix(string[] rows)

{

int rowCount = rows.Length;

int colCount = rows[0].Split(new[] { ' ', '\t' }, StringSplitOptions.RemoveEmptyEntries).Length;

var result = new double[rowCount, colCount];

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

{

var elements = rows[i].Split(new[] { ' ', '\t' }, StringSplitOptions.RemoveEmptyEntries);

if (elements.Length != colCount)

throw new ArgumentException("Матриця не є прямокутною.");

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

result[i, j] = double.Parse(elements[j]);

}

return result;

InvalidateCache();// Очищення кешу після зміни

}

// Властивості

public int Height => matrix.GetLength(0);

public int Width => matrix.GetLength(1);

public int GetHeight() => Height;

public int GetWidth() => Width;

// Індексатор

public double this[int row, int col]

{

get

{

ValidateIndices(row, col);

return matrix[row, col];

}

set

{

ValidateIndices(row, col);

matrix[row, col] = value;

InvalidateCache(); // Очищення кешу після зміни

}

}

public void InvalidateCache()

{

determinantCache = null;

ChanceDeterminant = null; // Якщо використовується альтернативний кеш

isMod = true; // Оновлюємо стан матриці

}

// Getter/Setter у Java-стилі

public double GetElement(int row, int col) => this[row, col];

public void SetElement(int row, int col, double value)

{

this[row, col] = value;

InvalidateCache();// Очищення кешу після зміни

}

// Переозначення ToString

public override string ToString()

{

var sb = new StringBuilder();

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

{

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

sb.Append(matrix[i, j] + "\t");

sb.AppendLine();

}

return sb.ToString();

}

private void ValidateIndices(int row, int col)

{

if (row < 0 || row >= Height || col < 0 || col >= Width)

throw new IndexOutOfRangeException("Індекси виходять за межі матриці.");

}

}

MatrixOperations:

public partial class MyMatrix

{

// Оператор додавання

public static MyMatrix operator +(MyMatrix a, MyMatrix b)

{

if (a.Height != b.Height || a.Width != b.Width)

throw new InvalidOperationException("Матриці мають різні розміри.");

var result = new double[a.Height, a.Width];

for (int i = 0; i < a.Height; i++)

for (int j = 0; j < a.Width; j++)

result[i, j] = a[i, j] + b[i, j];

return new MyMatrix(result);

}

// Оператор множення

public static MyMatrix operator \*(MyMatrix a, MyMatrix b)

{

if (a.Width != b.Height)

throw new InvalidOperationException("Матриці не можна множити (некоректні розміри).");

var result = new double[a.Height, b.Width];

for (int i = 0; i < a.Height; i++)

for (int j = 0; j < b.Width; j++)

for (int k = 0; k < a.Width; k++)

result[i, j] += a[i, k] \* b[k, j];

return new MyMatrix(result);

}

// Транспонування

private double[,] GetTransponedArray()

{

var result = new double[Width, Height];

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

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

result[j, i] = matrix[i, j];

return result;

}

public MyMatrix GetTransponedCopy() => new MyMatrix(GetTransponedArray());

public void TransponeMe() => matrix = GetTransponedArray();

// Додаткове: визначник

private double? determinantCache = null;

public double CalcDeterminant()

{

if (Height != Width)

throw new InvalidOperationException("Визначник обчислюється тільки для квадратних матриць.");

// Якщо визначник уже обчислений і матриця не модифікована, повертаємо кешоване значення

if (determinantCache.HasValue && !isMod)

return determinantCache.Value;

// Обчислення визначника

determinantCache = CalculateDeterminantInternal((double[,])matrix.Clone());

isMod = false; // Скидаємо прапорець модифікації

return determinantCache.Value;

}

private double CalculateDeterminantInternal(double[,] data)

{

int n = data.GetLength(0);

if (n == 1)

return data[0, 0];

if (n == 2)

return data[0, 0] \* data[1, 1] - data[0, 1] \* data[1, 0];

double determinant = 0;

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

{

var minor = GetMinor(data, 0, col);

determinant += (col % 2 == 0 ? 1 : -1) \* data[0, col] \* CalculateDeterminantInternal(minor);

}

return determinant;

}

private double[,] GetMinor(double[,] matrix, int row, int col)

{

int n = matrix.GetLength(0);

var minor = new double[n - 1, n - 1];

for (int i = 0, mi = 0; i < n; i++)

{

if (i == row) continue;

for (int j = 0, mj = 0; j < n; j++)

{

if (j == col) continue;

minor[mi, mj++] = matrix[i, j];

}

mi++;

}

return minor;

}

public static void ModifyMatrix(MyMatrix matrix)

{

while (true)

{

Console.WriteLine("\nПоточна матриця:");

Console.WriteLine(matrix);

try

{

Console.Write("Введіть номер рядка (0-based, або -1 для виходу): ");

int row = int.Parse(Console.ReadLine());

if (row == -1) break;

Console.Write("Введіть номер стовпця (0-based): ");

int col = int.Parse(Console.ReadLine());

Console.Write("Введіть нове значення: ");

double value = double.Parse(Console.ReadLine());

matrix[row, col] = value; // Зміна значення в матриці

Console.WriteLine("\nЕлемент успішно змінено.");

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

}

}