**Міністерство освіти України**

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Факультет інформатики та обчислювальної техніки**

**ЗВІТ**

до лабораторної роботи № 2

з дисципліни “ООП”

Виконав:

Студент групи ІП-71

Козирєв Олександр Анатолійович

Київ 2018

ЗМІСТ

[ВСТУП 3](#_Toc525399384)

[1. Мета роботи 4](#_Toc525399385)

[2. Постановка задачі 5](#_Toc525399386)

[3. Теоретичні положення 6](#_Toc525399387)

[4. UML-ДІАГРАМА КЛАСІВ 8](#_Toc525399388)

[5. Вихідний код програми 9](#_Toc525399389)

[6. ПРИКЛАД РОБОТИ ПРОГРАМИ 18](#_Toc525399390)

[ВИСНОВКИ 19](#_Toc525399391)

# ВСТУП

**Об'є́ктно-орієнто́ване програмува́ння (ООП)** — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: інкапсуляція, успадкування, поліморфізм. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами).

Основним поняттям ООП є об'єкт. **Об'єкт** можна визначити як певну сукупність даних(характеристик об'єкта) та методів роботи з ними. Для класифікації об'єктів у ООП використовують класи. **Клас** служить зразком для створення обєкту, тобто об'єкт є нічим іншим, ніж копією класу.

Існування ООП можливе завдяки трьом основним парадигмам **на яких базується саме ООП:**

- **Інкапсуляція**. Також відома як приховування даних. Зміст інкапсуляції полягає у приховуванні від зовнішнього користувача деталей реалізації об'єкта, замість цього надаючи інтерфейс взаємодії з ним.

- **Успадкування.** Це означає, що об'єкти (класи) можуть переймати деякі властивості у своїх прабатьків. Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть мати нові власні атрибути. Тобто утворюється ієрархія з класів, де від основного класу(так званого, предка) походять усі інші класи.

- **Поліморфізм** означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по різному на однакові повідомлення. Це спричинене зміною в одного з класів якогось методу(процедури, функції), шляхом запису іншого алгоритму. Як приклад, деяка комп'ютерна програма при натисканні клавіші Esc завершить роботу, інша ж програма після натискання кнопки Esc тільки відкриє меню даної програми.

# 1. Мета роботи

**Мета роботи** - вивчити особливості дружніх функцій (або їх еквівалентів) та операторних функцій. Освоїти принципи написання функцій переведення типів об’єктів, перевантаження операторів і дружніх функцій або їх еквівалентів.

# 2. Постановка задачі

Спроектировать класс «Bignum\_arithmetic», который содержит: длинное число, т.е. число, значения которого превышает максимально (минимально) допустимые значения целочисленного типа (MIN\_INT и MAX\_INT). Спроектировать класс «Fraction», который содержит: дробь в формате «±m/n», правильную или неправильную. Для того чтобы избежать потерь точности числитель (m) и знаменатель (n) следует сделать длинными числами (объектами класса Bignum\_arithmetic). Спроектировать класс «Matrix», который содержит: высоту и ширину матрицы, а сама матрица состоит из объектов класса «Fraction». Для класса «Matrix» определить: операцию сложения «+», операцию вычитания «-», операцию умножения «\*», операцию деления на число «/» (причем это может быть как объект класса «Fraction» или «Bignum\_arithmetic» так и простое число), эти же операции в сокращенной форме, операцию унарный минус «-», операцию нахождения обратной матрицы «!». Элементы матрицы при возможности стоит сокращать. При сокращении, для нахождения наибольшего общего делителя использовать бинарный алгоритм. Для классов «Bignum\_arithmetic» и «Fraction» достаточно определить только необходимые операции. При необходимости разрешается определять другие операции (например «=») и методы (например, getter, setter и прочее). Продемонстрировать каждую операцию класса «Matrix».

# 3. Теоретичні положення

**Довга арифметика** — в обчислювальній техніці операції над числами, розрядність яких перевищує довжину машинного слова даної обчислювальної машини. По суті арифметика з великими числами являє собою набір алгоритмів виконання базових операцій (додавання, множення, зведення в степінь …) над числами, реалізованими не апаратно, а програмно, використовуючи більш базові апаратні засоби роботи з числами менших порядків.

### **Алгоритми множення**

#### **- Базовий**

#### **- Множення Карацуби**

#### **- FFT Multiplication**

#### **Алгоритми ділення**

* + Single Limb Division
  + Basecase Division
  + Divide and Conquer Division
  + Block-Wise Barrett Division
  + Exact Division
  + Exact Remainder
  + Small Quotient Division

**Ма́триця** — [математичний об'єкт](https://uk.wikipedia.org/wiki/Математичний_об'єкт), записаний у вигляді прямокутної таблиці чисел (чи елементів [кільця](https://uk.wikipedia.org/wiki/Кільце_(алгебра))), він допускає операції ([додавання](https://uk.wikipedia.org/wiki/Додавання), [віднімання](https://uk.wikipedia.org/wiki/Віднімання), множення та [множення на скаляр](https://uk.wikipedia.org/wiki/Множення_на_скаляр)). Зазвичай матриці представляються двовимірними (прямокутними) таблицями. Іноді розглядають багатовимірні матриці або матриці непрямокутної форми.

Матриці є корисними для запису даних, що залежать від двох категорій, наприклад: для коефіцієнтів систем лінійних рівнянь та [лінійних перетворень](https://uk.wikipedia.org/wiki/Лінійний_оператор).

## **Дії над матрицями:**

### - Операція порівняння

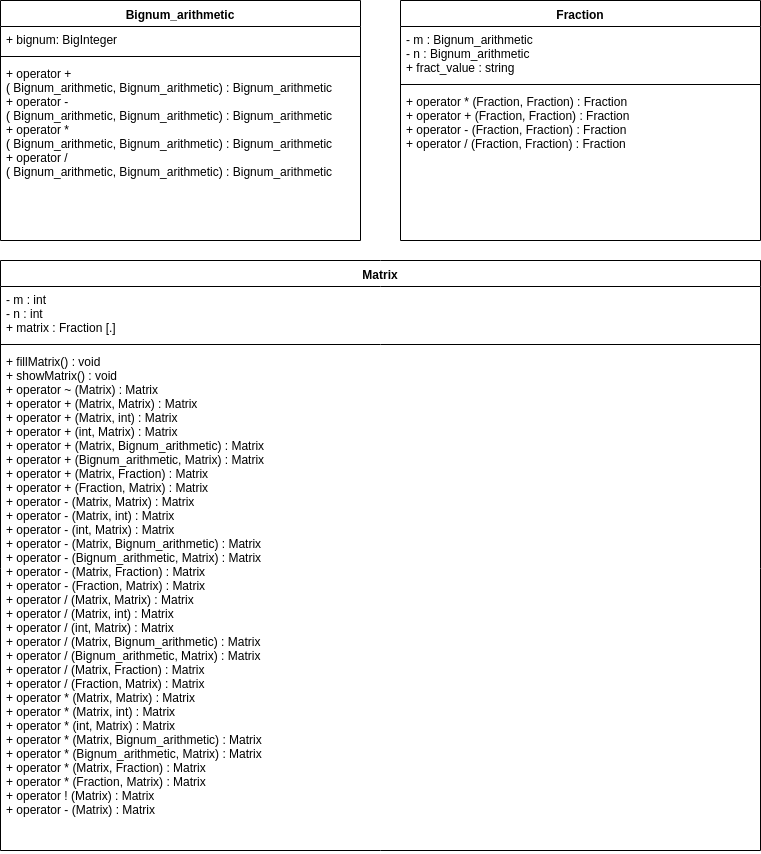
### - Додавання

### - Множення на скаляр

### - Множення матриць

### - Транспонування

# UML-ДІАГРАМА КЛАСІВ



# Вихідний код програми

1. Program.cs

using System;

namespace lab2\_c

{

class Program

{

static void Main(string[] args)

{

Matrix matrix1 = new Matrix(3, 5);

Matrix matrix2 = new Matrix(3, 5);

Bignum\_arithmetic ba1 = new Bignum\_arithmetic("10");

Fraction fa1 = new Fraction("-2/1");

matrix1.fillMatrix();

matrix2.fillMatrix();

Matrix matrix3 = 5 + matrix1 + matrix2 - ba1 \* fa1 \* 1000;

Matrix show = ~matrix3;

Console.WriteLine();

matrix3.showMatrix();

// matrix1.showMatrix();

// Console.WriteLine();

// matrix3.showMatrix();

// Matrix matrix4 = -matrix3;

// Console.WriteLine();

// matrix4.showMatrix();

// Matrix matrix5 = !matrix3;

// Console.WriteLine();

// matrix5.showMatrix();

}

}

}

2. Bignum\_arithmetic.cs

using System;

using System.Collections.Generic;

using System.Numerics;

namespace lab2\_c

{

class Bignum\_arithmetic

{

public BigInteger bignum;

public Bignum\_arithmetic(string number) {

bignum = BigInteger.Parse(number);

}

public Bignum\_arithmetic(BigInteger number) {

bignum = number;

}

public static Bignum\_arithmetic operator +(Bignum\_arithmetic left, Bignum\_arithmetic right) {

return new Bignum\_arithmetic(BigInteger.Add(left.bignum, right.bignum));

}

public static Bignum\_arithmetic operator -(Bignum\_arithmetic left, Bignum\_arithmetic right) {

return new Bignum\_arithmetic(BigInteger.Subtract(left.bignum, right.bignum));

}

public static Bignum\_arithmetic operator \*(Bignum\_arithmetic left, Bignum\_arithmetic right) {

return new Bignum\_arithmetic(BigInteger.Multiply(left.bignum, right.bignum));

}

public static Bignum\_arithmetic operator /(Bignum\_arithmetic left, Bignum\_arithmetic right) {

return new Bignum\_arithmetic(BigInteger.Divide(left.bignum, right.bignum));

}

}

}

3. Fraction.cs

using System;

using System.Numerics;

namespace lab2\_c

{

class Fraction

{

private Bignum\_arithmetic m;

private Bignum\_arithmetic n;

public string fract\_value;

public Fraction(string value) {

fract\_value = value;

m = new Bignum\_arithmetic(value.Split("/")[0]);

n = new Bignum\_arithmetic(value.Split("/")[1]);

}

public static implicit operator Fraction (Bignum\_arithmetic ba) => (new Fraction($"{ba.bignum}/1"));

public static implicit operator Fraction (int value) => (new Fraction($"{value}/1"));

public static Fraction operator \*(Fraction f1, Fraction f2) {

BigInteger gcd = BigInteger.GreatestCommonDivisor(f1.m.bignum \* f2.m.bignum, f1.n.bignum \* f2.n.bignum);

return new Fraction($"{f1.m.bignum \* f2.m.bignum / gcd}/{f1.n.bignum \* f2.n.bignum / gcd}");

}

public static Fraction operator /(Fraction f1, Fraction f2) {

BigInteger gcd = BigInteger.GreatestCommonDivisor(f1.m.bignum \* f2.n.bignum, f1.n.bignum \* f2.m.bignum);

return new Fraction($"{f1.m.bignum \* f2.m.bignum / gcd}/{f1.n.bignum \* f2.n.bignum / gcd}");

}

public static Fraction operator +(Fraction f1, Fraction f2) {

BigInteger gcd = BigInteger.GreatestCommonDivisor(f1.m.bignum \* f2.n.bignum + f2.m.bignum \* f1.n.bignum, f1.n.bignum \* f2.n.bignum);

return new Fraction($"{f1.m.bignum \* f2.n.bignum + f2.m.bignum \* f1.n.bignum}/{f1.n.bignum \* f2.n.bignum}");

}

public static Fraction operator -(Fraction f1, Fraction f2) {

BigInteger gcd = BigInteger.GreatestCommonDivisor(f1.m.bignum \* f2.n.bignum - f2.m.bignum \* f1.n.bignum, f1.n.bignum \* f2.n.bignum);

return new Fraction($"{f1.m.bignum \* f2.n.bignum - f2.m.bignum \* f1.n.bignum}/{f1.n.bignum \* f2.n.bignum}");

}

}

}

4. Matrix.cs

using System;

using System.Numerics;

namespace lab2\_c

{

class Matrix

{

private int m;

private int n;

public Fraction[,] matrix;

public Matrix (int width, int height) {

m = width;

n = height;

matrix = new Fraction[m, n];

}

public void fillMatrix () {

Random rand = new Random();

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

for (int j = 0; j < n; j++) {

int numerator = rand.Next(-10, 50);

int denuminator = rand.Next(-30, 100);

matrix[i,j] = new Fraction($"{numerator}/{denuminator}");

}

}

}

public void showMatrix () {

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

for (int j = 0; j < n; j++) {

Console.Write($"{matrix[i,j].fract\_value} ");

}

Console.WriteLine();

}

}

public static Matrix operator ~(Matrix m) {

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

for (int j = 0; j < m.n; j++) {

Console.Write($"{m.matrix[i,j].fract\_value} ");

}

Console.WriteLine();

}

return m;

}

public static Matrix operator +(Matrix m1, Matrix m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] + m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator +(Matrix m1, int m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] + m2;

}

}

return result;

}

public static Matrix operator +(int m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 + m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator +(Matrix m1, Bignum\_arithmetic m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] + m2;

}

}

return result;

}

public static Matrix operator +(Bignum\_arithmetic m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 + m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator +(Matrix m1, Fraction m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] + m2;

}

}

return result;

}

public static Matrix operator +(Fraction m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 + m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator -(Matrix m1, Matrix m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] - m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator -(Matrix m1, int m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] - m2;

}

}

return result;

}

public static Matrix operator -(int m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 - m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator -(Matrix m1, Bignum\_arithmetic m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] - m2;

}

}

return result;

}

public static Matrix operator -(Bignum\_arithmetic m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 - m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator -(Matrix m1, Fraction m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] - m2;

}

}

return result;

}

public static Matrix operator -(Fraction m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 - m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator \*(Matrix m1, Matrix m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] \* m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator \*(Matrix m1, int m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] \* m2;

}

}

return result;

}

public static Matrix operator \*(int m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 \* m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator \*(Matrix m1, Bignum\_arithmetic m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] \* m2;

}

}

return result;

}

public static Matrix operator \*(Bignum\_arithmetic m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 \* m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator \*(Matrix m1, Fraction m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] \* m2;

}

}

return result;

}

public static Matrix operator \*(Fraction m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 \* m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator /(Matrix m1, Matrix m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] / m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator /(Matrix m1, int m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] / m2;

}

}

return result;

}

public static Matrix operator /(int m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 / m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator /(Matrix m1, Bignum\_arithmetic m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] / m2;

}

}

return result;

}

public static Matrix operator /(Bignum\_arithmetic m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 / m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator /(Matrix m1, Fraction m2) {

Matrix result = new Matrix(m1.m, m1.n);

for (int i = 0; i < m1.m; i++) {

for (int j = 0; j < m1.n; j++) {

result.matrix[i,j] = m1.matrix[i,j] / m2;

}

}

return result;

}

public static Matrix operator /(Fraction m1, Matrix m2) {

Matrix result = new Matrix(m2.m, m2.n);

for (int i = 0; i < m2.m; i++) {

for (int j = 0; j < m2.n; j++) {

result.matrix[i,j] = m1 / m2.matrix[i,j];

}

}

return result;

}

public static Matrix operator -(Matrix matrix) {

for (int i = 0; i < matrix.m; i++) {

for (int j = 0; j < matrix.n; j++) {

matrix.matrix[i,j] = matrix.matrix[i,j] \* -1;

}

}

return matrix;

}

public static Matrix operator !(Matrix matrix) {

Matrix newMatrix = new Matrix(matrix.n, matrix.m);

for (int i = 0; i < matrix.m; i++) {

for (int j = 0; j < matrix.n; j++) {

newMatrix.matrix[j,i] = matrix.matrix[i,j];

}

}

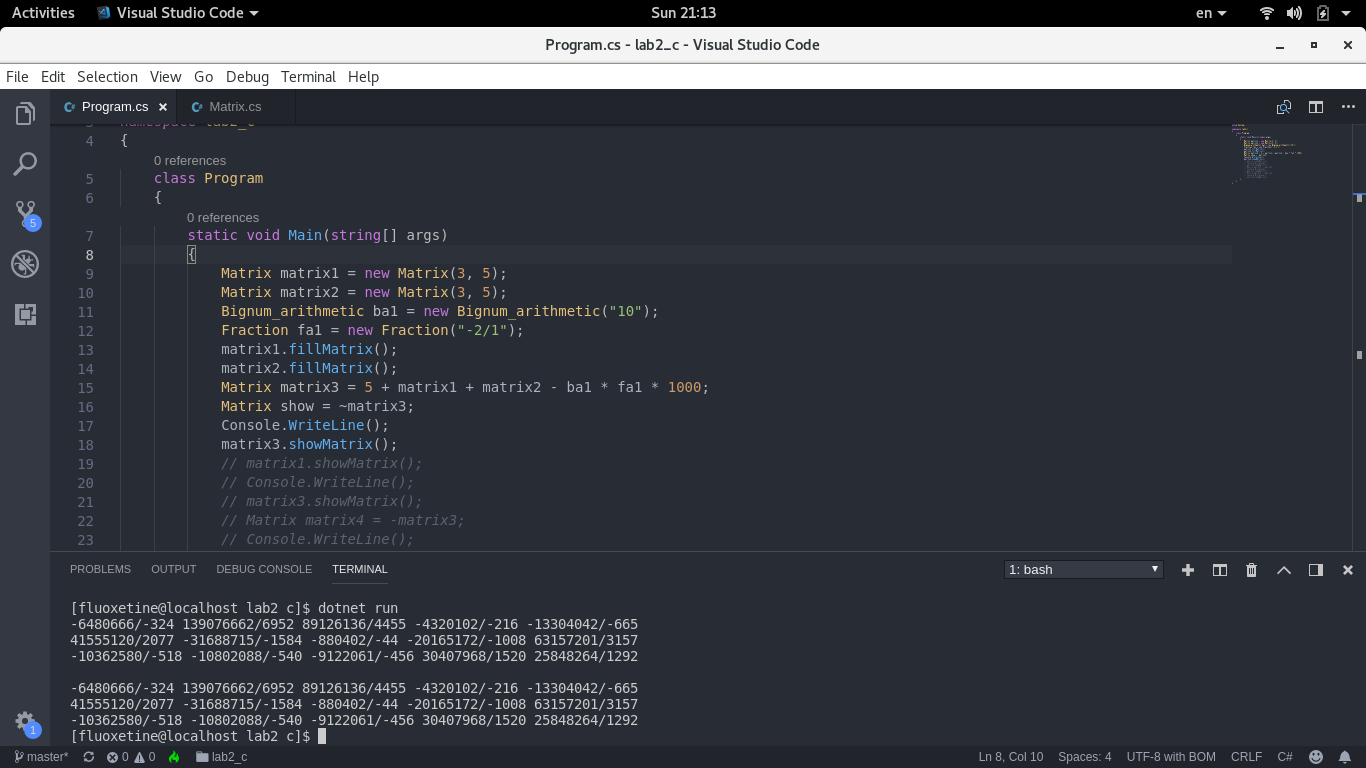
return newMatrix;

}

}

}

# ПРИКЛАД РОБОТИ ПРОГРАМИ



# ВИСНОВКИ

В рамках даної лабораторної роботи вдалося ознайомитися і використати на практиці принципи написання функцій переведення типів об’єктів, перевантаження операторів.

Були створені класи для операцій с довгими числами, клас для операцій над дробами, клас операцій над матрицями, налаштована їх взаємодія між собою.