ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ОТЧЁТ**

**по лабораторной работе № 6**

**«Построение элементарных классов»**

**по дисциплине: «***Программирование***»**

Выполнил:Проверил:

Студент гр. «АБ-121», «АВТФ» *доцент кафедры ЗИ*

*Новиков Втюрин Александр Романович Архипова А. Б.*

«21» декабря 2022г«\_\_\_» \_\_\_\_\_\_ 2022 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2022

**Цели и задачи работы:** изучение основных принципов создания классов с арифметическими операциями.

**Задание к работе:**Реализовать вычислительный процесс с использованием классов. Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

**Part 1.**

**Задание 1. Создание класса согласно списка требований, предъявляемых к реализации интерфейса класса. Example (object-oriented** **programming, encapsulation).**

**Задание 2. Многофайловая организация проекта. Multi-file** **project**.

**Вариант 2**

Написать программу реализующую работу с дробями (числитель / знаменатель). Другие форматы задания дроби должны рассматриваться как исключение. Для числителя и знаменателя должна поддерживаться инкапсуляция (секция private).

Список требований, предъявляемых к реализации интерфейса класса:

1. Конструктор по умолчанию должен создавать дробь с числителем 0 и знаменателем 1.

2. Параметризованный конструктор должен выполнять сокращение дроби на значение Наибольшего общего делителя (алгоритм Евклида).

3. Знак дроби непосредственно относится к числителю.

4. Не допускается деление на ноль. В случае если такое произошло обработать исключение.

5. Класс дроби должен поддерживать корректные арифметические и логические операции (перегрузки +, -, /, \*, -1, степень, <=, >, >=, =).

6. Класс должен реализовывать оператор присваивания (единственный метод-модификатор в классе).

7. Задать логгер к классу, фиксирующий историю вызова различных операций класса.

**2. Multi-file project**

Реализовать многофайловый проект (\*.h, \*.cpp). Проверить последовательность сборки проекта в случае изменения файлов, обосновать многофайловую структуру проекта с размещением класса в двух файлов.

Ответить на вопрос: каким образом шаблонизация методов классов повлияет на Ваше разбиение кода по файлам.

**Part 2.**

**Создание класса согласно списка требований, предъявляемых к реализации интерфейса класса. Example (object-oriented** **programming, encapsulation). Обязательна многофайловая структура проекта.**

**Вариант 4 (0,5)**

Реализовать Class FunnyGame:

1. Поля - двумерный вектор размерности NxN, где N - параметр конструктора.

2. В конструкторе рандомным образом [N/2] ячейки двумерного массива приобретают значение true, остальные остаются false.

3. В начале игры считывается N чисел, если хотя бы [N/3] из них совпадают с теми, что были отмечены true - игра выиграна, необходимо вывести соответствующее сообщение.

4. Для определения победы реализовать отдельный метод bool isWin().

**Part 3.**

**Задание**: Сформировать обработку исключений для **Part1** и **Part2**.

1. **Создать абстрактный класс IError**– класс, определяющий интерфейс для хранения, описания и вывода ошибок. Поле – текст ошибки/код ошибки. Виртуальная функция print, которая по умолчанию печатает единственное поле.

2. **Отнаследовать от IError как минимум два класса ошибок** (не абстрактных). В обработках исключений вместо завершения программы возвращаться в START (для этого нужен event loop и(ли) state machine), если необходимо, и заносить информацию об ошибке в соответствующий типу ошибки класс.

3. **Реализовать функцию print\_error\_list**(std::vector err), которая будет выводить список всех ошибок, произошедших за время работы программы. Использовать повышение приведения типа.

**С++**

**main.cpp**

#include "Fraction.h"

#include "IError.h"

using namespace std;

int main()

{

try {

int choice;

cout << "What task to complete(PartI - 1, PartII - 2, Exit - 3): ";

cin >> choice;

if (choice == 1)

{

PartI();

}

else if (choice == 2)

{

PartII();

}

else if (choice == 3)

{

exit(0);

}

}

catch (IncorrectInput& e){

e.print();

main();

}

}

**Fraction.h**

#pragma once

#include <iostream>

#include <sstream>

#include <vector>

#include "IError.h"

using namespace std;

class Fraction {

private:

int numerator, denominator;

vector <string> logger;

public:

Fraction();

Fraction(int a, int b);

friend ostream& operator<<(ostream& os, const Fraction& fraction) {

os << fraction.numerator << " / " << fraction.denominator;

return os;

}

friend istream& operator>>(istream& is, Fraction& fraction) {

try {

cout << "Enter Numerator ";

is >> fraction.numerator;

cout << "Enter Denominator ";

is >> fraction.denominator;

if (is.fail())

{

throw IncorrectInput();

}

if (fraction.denominator == 0)

{

throw DivisionByZero();

}

}

catch (DivisionByZero& e)

{

e.print();

exit(0);

}

return is;

}

Fraction operator=(const Fraction& sravn) {

return \*this;

}

Fraction operator+(const Fraction& fraction2);

Fraction operator-(const Fraction& fraction2);

Fraction operator/( Fraction& fraction2);

Fraction operator\*(const Fraction& fraction2);

Fraction pow(const Fraction& fraction2, int a);

bool operator<(const Fraction& fraction2);

bool operator<=(const Fraction& fraction2);

bool operator>(const Fraction& fraction2);

bool operator >=(const Fraction& fraction2);

};

int nod(int a, int b);

int PartI();

int PartII();

**Fraction.cpp**

#include "Fraction.h"

int nod(int a, int b)

{

int t;

if (a < b) { t = a; a = b; b = t; }

while (b != 0) {

t = b;

b = a % b;

a = t;

}

return a;

}

Fraction::Fraction() {

numerator = 0;

denominator = 1;

logger.push\_back("Create default Fraction\n");

cout << "Create default Fraction\n";

}

Fraction::Fraction(int a, int b){

int getNod = nod(a, b);

numerator = a / getNod;

denominator = b / getNod;

if (b == 0) {

cout << "denominator could not be 0, now denominator = 1\n";

denominator = 1;

}

if (numerator < 0 && denominator < 0) { numerator \*= -1; denominator += -1; }

logger.push\_back("Create new Fraction\n");

cout << "Create new Fraction\n";

}

Fraction Fraction::operator+(const Fraction& fraction2) {

if (fraction2.denominator == this->denominator) {

return Fraction((this->numerator + fraction2.numerator), (this->denominator));

}

else {

int sum\_den = fraction2.denominator \* this->denominator;

int sum\_num = fraction2.numerator \* this->denominator + this->numerator \* fraction2.denominator;

return Fraction((sum\_num), (sum\_den));

}

logger.push\_back("Do summary\n");

cout << "Do summary\n";

}

Fraction Fraction::operator-(const Fraction& fraction2) {

if (fraction2.denominator == this->denominator) {

return Fraction((this->numerator - fraction2.numerator), (this->denominator));

}

else {

int sum\_den = fraction2.denominator \* this->denominator;

int sum\_num = this->numerator \* fraction2.denominator - fraction2.numerator \* this->denominator;

return Fraction((sum\_num), (sum\_den));

}

}

Fraction Fraction::operator/( Fraction& fraction2) {

try {

swap(fraction2.numerator, fraction2.denominator);

logger.push\_back("DIVISION OBJECT\n");

cout << ("DIVISION OBJECT\n");

return Fraction(fraction2.numerator \* this->numerator, fraction2.denominator \* this->denominator);

}

catch(IncorrectInput&e) {

e.print();

}

}

Fraction Fraction::operator\*(const Fraction& fraction2) {

logger.push\_back("MULTIPLE OBJECT\n");

cout << ("MULTIPLE OBJECT\n");

return Fraction(fraction2.numerator \* this->numerator, fraction2.denominator \* this->denominator);

}

Fraction Fraction::pow(const Fraction& fraction2, int a) {

int num\_degree = fraction2.numerator, den\_degree = fraction2.denominator;

bool invert = false;

if (a < 0) {

a \*= -1;

invert = true;

}

while (a != 1) {

num\_degree \*= fraction2.numerator;

den\_degree \*= fraction2.denominator;

a--;

}

if (invert)swap(num\_degree, den\_degree);

logger.push\_back("DEGREE OPERATOR\n");

cout << ("DEGREE OPERATOR\n");

return Fraction(num\_degree, den\_degree);

}

bool Fraction::operator<(const Fraction& fraction2) {

logger.push\_back("LESS OPERATOR\n");

cout << ("LESS OPERATOR\n");

if (fraction2.denominator == this->denominator) {

return this->numerator < fraction2.numerator;

}

else return this->numerator \* fraction2.denominator < fraction2.numerator\* this->denominator;

}

bool Fraction::operator<=(const Fraction& fraction2) {

logger.push\_back("EQUALS OR LESS OPERATOR\n");

cout << ("EQUALS OR LESS OPERATOR\n");

if (fraction2.denominator == this->denominator) {

return this->numerator <= fraction2.numerator;

}

else return this->numerator \* fraction2.denominator <= fraction2.numerator \* this->denominator;

}

bool Fraction::operator>(const Fraction& fraction2) {

logger.push\_back("LESS OPERATOR\n");

cout << ("LESS OPERATOR\n");

if (fraction2.denominator == this->denominator) {

return this->numerator > fraction2.numerator;

}

else return this->numerator \* fraction2.denominator > fraction2.denominator \* this->denominator;

}

bool Fraction::operator>=(const Fraction& fraction2) {

logger.push\_back("EQUALS OR MORE OPERATOR\n");

cout << ("EQUALS OR MORE OPERATOR\n");

if (fraction2.denominator == this->denominator) {

return this->numerator >= fraction2.numerator;

}

else return this->numerator \* fraction2.denominator >= fraction2.numerator \* this->denominator;

}

**Part1.cpp**

#include "Fraction.h"

int PartI()

{

Fraction number(7, 23);

Fraction number1;

cin >> number1;

cout << (number > number1) << endl;

cout << (number >= number1) << endl;

cout << (number < number1) << endl;

cout << (number <= number1) << endl;

cout << (number + number1) << endl;

cout << (number - number1) << endl;

cout << (number \* number1) << endl;

cout << (number / number1) << endl;

system("pause");

return 0;

}

**Part2.cpp**

#include <ctime>

#include <cstdlib>

#include <iostream>

#include <vector>

#include "Fraction.h"

using namespace std;

class FunnyGame {

public:

vector < vector < bool >> matrica;

FunnyGame(int n) {

matrica = vector < vector < bool >>(n, vector<bool>(n, false));

int stroka, stolbec;

for (int i = 0; i != n / 2; i++) {

stroka = rand() % n;

stolbec = rand() % n;

while (matrica[stroka][stolbec] == true) {

stroka = rand() % n;

stolbec = rand() % n;

}

matrica[stroka][stolbec] = true;

}

}

bool isWin(int n) {

int q = 0;

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

if (matrica[rand() % n][rand() % n] == true) q++;

}

if (q >= n / 3) return true;

else return false;

}

};

int PartII()

{

srand(time(0));

int n;

cin >> n;

FunnyGame FG(n);

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

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

cout << FG.matrica[i][j] << " ";

}

cout << endl;

}

bool win = FG.isWin(n);

if (win)

cout << endl << "win" << endl;

else cout << endl << "lose" << endl;;

system("pause");

return 0;

}

**IError.h**

#pragma once

#include <iostream>

using namespace std;

class IError

{

public:

string description = "Error: ";

virtual void print() {

std::cout << description << std::endl;

}

};

class IncorrectInput : IError {

public:

IncorrectInput() {};

void print() {

cout << description << " Incorrect input " << std::endl;

}

};

class DivisionByZero : IError {

public:

DivisionByZero() {};

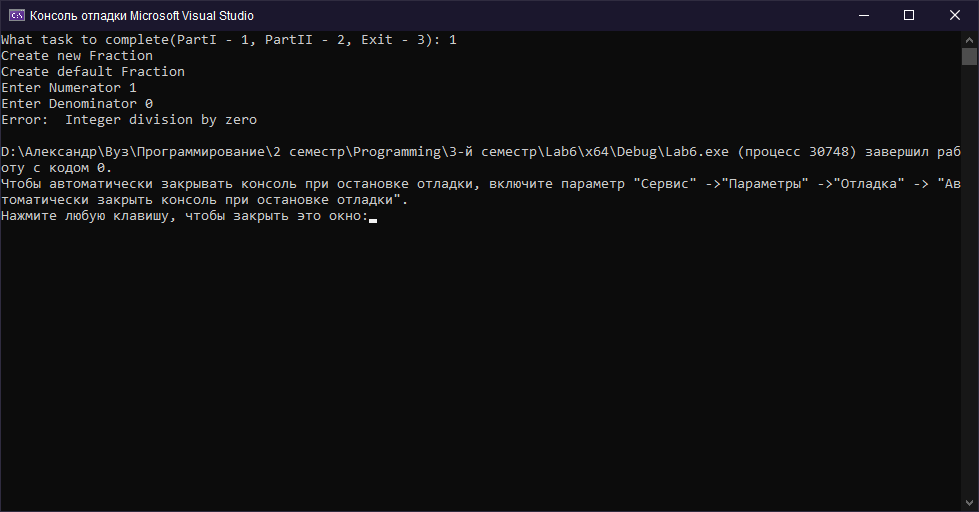
void print() {

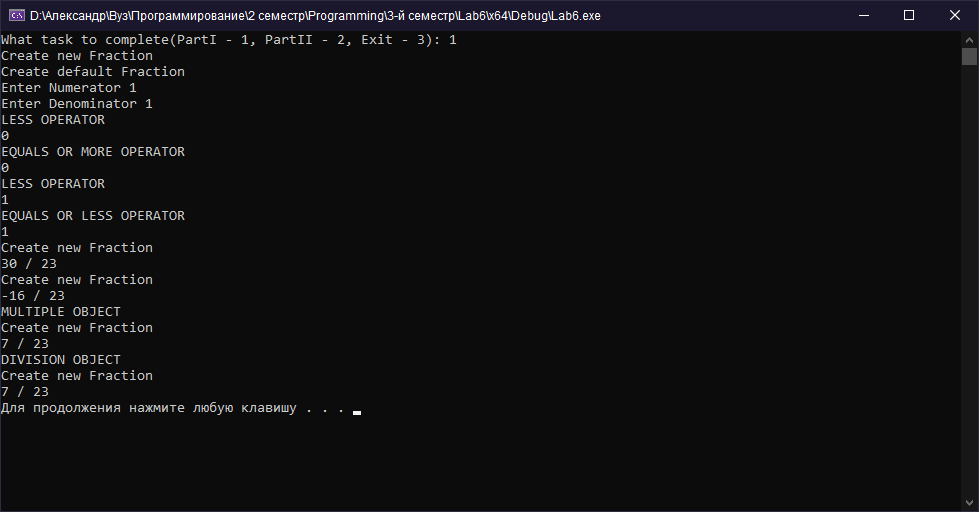
cout << description << " Integer division by zero " << std::endl;

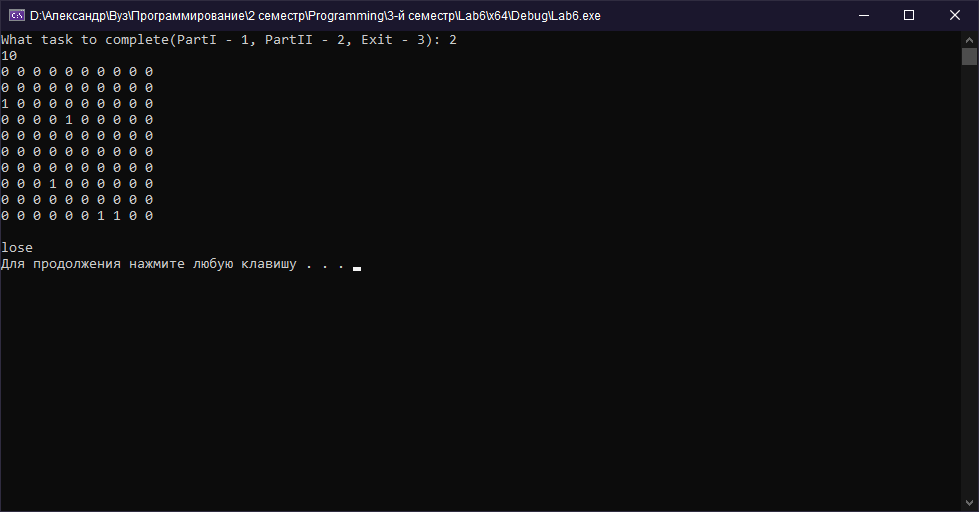
}

};

****

****

****

****

**C#**

**Main.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab6Sharp

{

static class Program

{

static int Main(string[] args)

{

try

{

TaskI task1 = new TaskI();

TaskII task2 = new TaskII();

Console.WriteLine("What task to complete(PartI - 1, PartII - 2, Exit - 3): ");

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

if (choice == 1)

{

task1.PartI();

}

else if (choice == 2)

{

task2.PartII();

}

else if (choice == 3)

{

return 0;

}

Console.ReadKey();

return 0;

}

catch

{

IncorrectInput e = new IncorrectInput();

e.print();

Console.ReadKey();

return 0;

}

}

}

}

**PartI.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab6Sharp

{

public class Fraction

{

public int numerator, denominator;

private List<string> logger = new List<string>();

public Fraction()

{

numerator = 0;

denominator = 1;

logger.Add("Create default Fraction\n");

Console.Write("Create default Fraction\n");

}

public Fraction(int a, int b)

{

int getNod = nod(a, b);

numerator = a / getNod;

denominator = b / getNod;

if (b == 0)

{

Console.Write("denominator could not be 0, now denominator = 1\n");

denominator = 1;

}

if (numerator < 0 && denominator < 0) { numerator \*= -1; denominator += -1; }

logger.Add("Create new Fraction\n");

Console.Write("Create new Fraction\n");

}

static int nod(int a, int b)

{

int t;

if (a < b) { t = a; a = b; b = t; }

while (b != 0)

{

t = b;

b = a % b;

a = t;

}

return a;

}

static void swap(int a, int b)

{

int c;

c = a;

a = b;

b = a;

}

public static Fraction operator + (Fraction fraction1, Fraction fraction2)

{

if (fraction2.denominator == fraction1.denominator)

{

Fraction fraction = new Fraction((fraction1.numerator + fraction2.numerator), (fraction1.denominator));

return fraction;

}

else

{

int sum\_den = fraction2.denominator \* fraction1.denominator;

int sum\_num = fraction2.numerator \* fraction1.denominator + fraction1.numerator \* fraction2.denominator;

Fraction fraction = new Fraction((sum\_num), (sum\_den));

return fraction;

}

fraction1.logger.Add("Do summary\n");

Console.Write( "Do summary\n");

}

public static Fraction operator - (Fraction fraction1, Fraction fraction2)

{

if (fraction2.denominator == fraction1.denominator)

{

Fraction fraction = new Fraction((fraction1.numerator - fraction2.numerator), (fraction1.denominator));

return fraction;

}

else

{

int sum\_den = fraction2.denominator \* fraction1.denominator;

int sum\_num = fraction1.numerator \* fraction2.denominator - fraction2.numerator \* fraction1.denominator;

Fraction fraction = new Fraction((sum\_num), (sum\_den));

return fraction;

}

}

public static Fraction operator / (Fraction fraction1, Fraction fraction2)

{

try

{

swap(fraction2.numerator, fraction2.denominator);

fraction1.logger.Add("DIVISION OBJECT\n");

Console.Write("DIVISION OBJECT\n");

Fraction fraction = new Fraction(fraction2.numerator \* fraction1.numerator, fraction2.denominator \* fraction1.denominator);

return fraction;

}

catch {

IncorrectInput e = new IncorrectInput();

e.print();

return null;

}

}

public static Fraction operator \* (Fraction fraction1, Fraction fraction2)

{

fraction1.logger.Add("MULTIPLE OBJECT\n");

Console.Write("MULTIPLE OBJECT\n");

Fraction fraction = new Fraction(fraction2.numerator \* fraction1.numerator, fraction2.denominator \* fraction1.denominator);

return fraction;

}

public static bool operator < (Fraction fraction1, Fraction fraction2)

{

fraction1.logger.Add("LESS OPERATOR\n");

Console.Write("LESS OPERATOR\n");

if (fraction2.denominator == fraction1.denominator)

{

return fraction1.numerator < fraction2.numerator;

}

else return fraction1.numerator \* fraction2.denominator < fraction2.numerator \* fraction1.denominator;

}

public static bool operator <= (Fraction fraction1, Fraction fraction2)

{

fraction1.logger.Add("EQUALS OR LESS OPERATOR\n");

Console.Write ("EQUALS OR LESS OPERATOR\n");

if (fraction2.denominator == fraction1.denominator)

{

return fraction1.numerator <= fraction2.numerator;

}

else return fraction1.numerator \* fraction2.denominator <= fraction2.numerator \* fraction1.denominator;

}

public static bool operator > (Fraction fraction1, Fraction fraction2)

{

fraction1.logger.Add("LESS OPERATOR\n");

Console.Write("LESS OPERATOR\n");

if (fraction2.denominator == fraction1.denominator)

{

return fraction1.numerator > fraction2.numerator;

}

else return fraction1.numerator \* fraction2.denominator > fraction2.denominator \* fraction1.denominator;

}

public static bool operator >= (Fraction fraction1, Fraction fraction2)

{

fraction1.logger.Add("EQUALS OR MORE OPERATOR\n");

Console.Write ("EQUALS OR MORE OPERATOR\n");

if (fraction2.denominator == fraction1.denominator)

{

return fraction1.numerator >= fraction2.numerator;

}

else return fraction1.numerator \* fraction2.denominator >= fraction2.numerator \* fraction1.denominator;

}

}

class TaskI

{

public void PartI()

{

Fraction number1 = new Fraction(7,23);

Fraction number2 = new Fraction();

Console.WriteLine("numerator: ");

number2.numerator = int.Parse(Console.ReadLine());

Console.WriteLine("denominator: ");

number2.denominator = int.Parse(Console.ReadLine());

Console.WriteLine(number1 > number2);

Console.WriteLine(number1 >= number2);

Console.WriteLine(number1 < number2);

Console.WriteLine(number1 <= number2);

Console.WriteLine(number1 + number2);

Console.WriteLine(number1 - number2);

Console.WriteLine(number1 \* number2);

Console.WriteLine(number1 / number2);

Console.ReadKey();

}

}

}

**PartII.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab6Sharp

{

public class FunnyGame

{

public bool[,] matric;

public FunnyGame(int n)

{

int stroka, stolbec;

matric = new bool[n,n];

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

{

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

{

matric[i,j] = false;

}

}

Random rnd = new Random();

for (int i = 0; i != n / 2; i++)

{

stroka= rnd.Next(0,n);

stolbec= rnd.Next(0,n);

while (matric[stroka,stolbec] == true)

{

stroka = rnd.Next(0,n);

stolbec = rnd.Next(0, n);

}

matric[stroka, stolbec] = true;

}

}

public bool isWin(int n)

{

Random rnd = new Random();

int q = 0;

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

{

if (matric[rnd.Next(0,n), rnd.Next(0, n)] == true) q++;

}

if (q >= (n / 3)) return true;

else return false;

}

}

class TaskII

{

public void PartII()

{

Random rnd = new Random();

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

FunnyGame FG = new FunnyGame(n);

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

{

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

{

Console.Write(Convert.ToInt32(FG.matric[i, j]));

Console.Write(" ");

}

Console.WriteLine("");

}

bool win = FG.isWin(n);

if (win)

{

Console.WriteLine("\n win");

}

Console.WriteLine("\n lose");

Console.ReadKey();

}

}

}

**IError.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab6Sharp

{

class IError

{

public string description = "Error: ";

public virtual void print()

{

Console.WriteLine(description);

}

}

class IncorrectInput : IError

{

public IncorrectInput() { }

public void print()

{

Console.WriteLine("{1} Incorrect input", description);

}

}

class DivisionByZero : IError

{

public DivisionByZero() { }

public void print()

{

Console.WriteLine("{1} Integer division by zero", description);

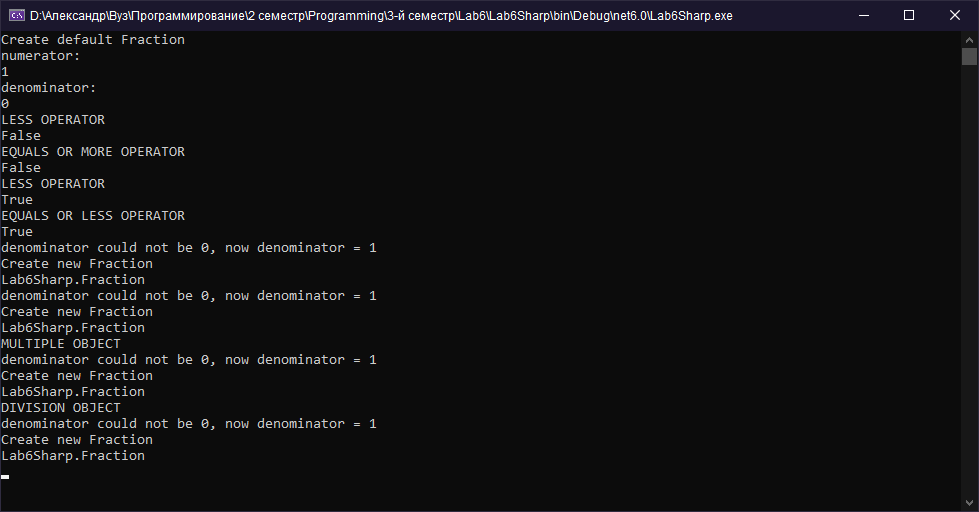
}

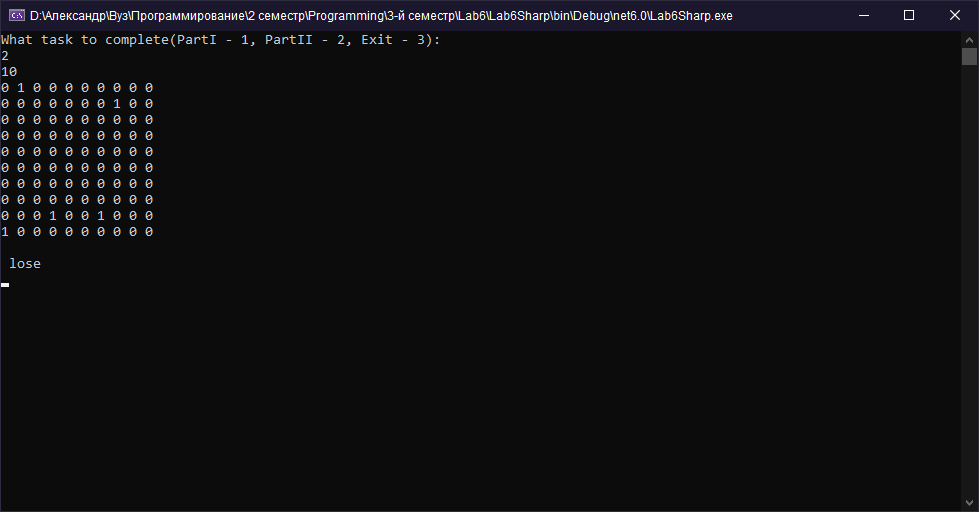
}

}

****

****

****

****