ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ОТЧЁТ**

**по лабораторной работе № 9**

**«Динамические библиотеки»**

**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

**Задание к работе:**Разработать и проэксплуатировать динамические библиотеки.

**Задание 1 (4 балла)**

Разработать и проэксплуатировать динамические библиотеки на основе ЛР6 на двух языках программирования.

**С++**

**Реализация программы на С++**

**Lab9++.cpp**

#include "Fraction.h"

#include "IError.h"

using namespace std;

int main()

{

try {

int choice;

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

cin >> choice;

if (choice == 1)

{

PartI();

}

else if (choice == 2)

{

exit(0);

}

}

catch (IncorrectInput& e) {

e.print();

main();

}

}

**Динамические библиотеки С++**

**dllmain.cpp**

// dllmain.cpp : Определяет точку входа для приложения DLL.

#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved

)

{

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

**Fraction.h**

#pragma once

#include <iostream>

#include <sstream>

#include <vector>

#include "IError.h"

using namespace std;

extern "C++" \_\_declspec(dllexport) class Fraction {

private:

int numerator, denominator;

vector <string> logger;

public:

\_\_declspec(dllexport) Fraction();

\_\_declspec(dllexport) Fraction(int a, int b);

\_\_declspec(dllexport) friend ostream& operator<<(ostream& os, const Fraction& fraction) {

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

return os;

}

\_\_declspec(dllexport) 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;

}

\_\_declspec(dllexport) Fraction operator=(const Fraction& sravn) {

return \*this;

}

\_\_declspec(dllexport) Fraction operator+(const Fraction& fraction2);

\_\_declspec(dllexport) Fraction operator-(const Fraction& fraction2);

\_\_declspec(dllexport) Fraction operator/(Fraction& fraction2);

\_\_declspec(dllexport) Fraction operator\*(const Fraction& fraction2);

\_\_declspec(dllexport) Fraction pow(const Fraction& fraction2, int a);

\_\_declspec(dllexport) bool operator<(const Fraction& fraction2);

\_\_declspec(dllexport) bool operator<=(const Fraction& fraction2);

\_\_declspec(dllexport) bool operator>(const Fraction& fraction2);

\_\_declspec(dllexport) bool operator >=(const Fraction& fraction2);

};

\_\_declspec(dllexport) int nod(int a, int b);

\_\_declspec(dllexport) int PartI();

**framework.h**

#pragma once

#define WIN32\_LEAN\_AND\_MEAN // Исключите редко используемые компоненты из заголовков Windows

// Файлы заголовков Windows

#include <windows.h>

**IError.h**

#pragma once

#include <iostream>

using namespace std;

extern "C++" \_\_declspec(dllexport) class IError

{

public:

string description = "Error: ";

\_\_declspec(dllexport) virtual void print() {

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

}

};

extern "C++" \_\_declspec(dllexport) class IncorrectInput : IError {

public:

\_\_declspec(dllexport) IncorrectInput() {};

\_\_declspec(dllexport) void print() {

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

}

};

extern "C++" \_\_declspec(dllexport) class DivisionByZero : IError {

public:

\_\_declspec(dllexport) DivisionByZero() {};

\_\_declspec(dllexport) void print() {

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

}

};

**pch.h**

// pch.h: это предварительно скомпилированный заголовочный файл.

// Перечисленные ниже файлы компилируются только один раз, что ускоряет последующие сборки.

// Это также влияет на работу IntelliSense, включая многие функции просмотра и завершения кода.

// Однако изменение любого из приведенных здесь файлов между операциями сборки приведет к повторной компиляции всех(!) этих файлов.

// Не добавляйте сюда файлы, которые планируете часто изменять, так как в этом случае выигрыша в производительности не будет.

#ifndef PCH\_H

#define PCH\_H

// Добавьте сюда заголовочные файлы для предварительной компиляции

#include "framework.h"

#endif //PCH\_H

**Fraction.cpp**

#include "pch.h"

#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 "pch.h"

#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;

}

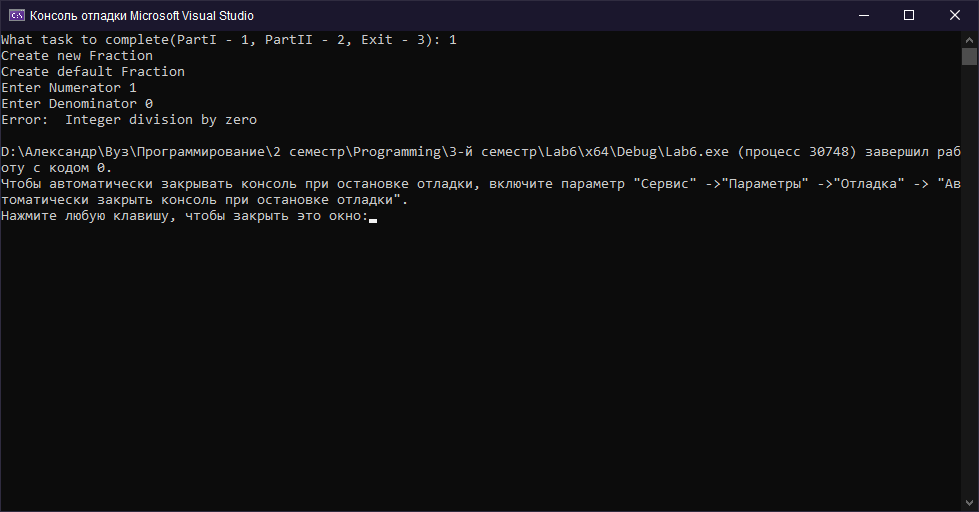
**pch.cpp**

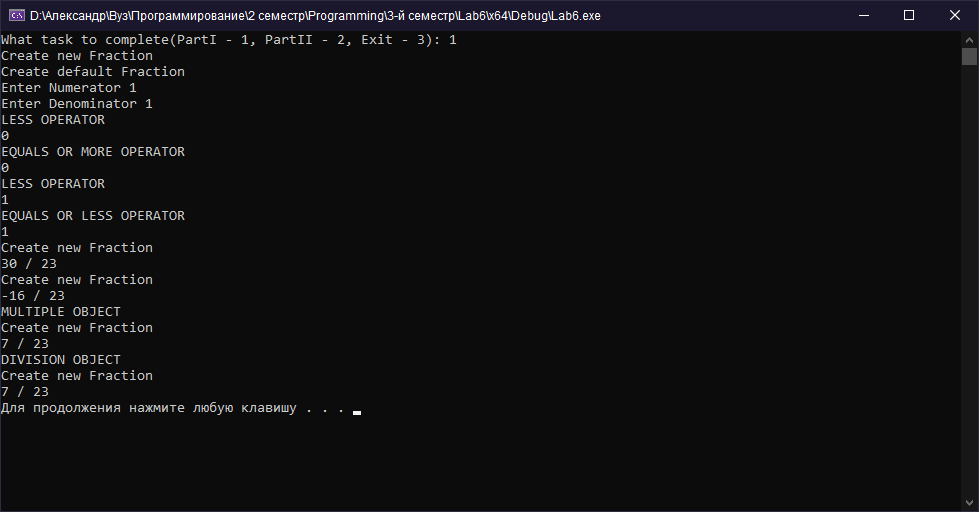
// pch.cpp: файл исходного кода, соответствующий предварительно скомпилированному заголовочному файлу

#include "pch.h"

// При использовании предварительно скомпилированных заголовочных файлов необходим следующий файл исходного кода для выполнения сборки.

****

****

****

**C#**

**Реализация программы на C#**

**Programm.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Lab9SharpDLL;

internal class Program

{

private static int Main(string[] args)

{

try

{

TaskI task1 = new TaskI();

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

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

if (choice == 1)

{

task1.PartI();

}

else if (choice == 2)

{

return 0;

}

Console.ReadKey();

return 0;

}

catch

{

IncorrectInput e = new IncorrectInput();

e.print();

Console.ReadKey();

return 0;

}

}

}

**Class1.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab9SharpDLL

{

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;

}

}

public 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();

}

}

}

**IError.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab9SharpDLL

{

public class IError

{

public string description = "Error: ";

public virtual void print()

{

Console.WriteLine(description);

}

}

public class IncorrectInput : IError

{

public IncorrectInput() { }

public void print()

{

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

}

}

public class DivisionByZero : IError

{

public DivisionByZero() { }

public void print()

{

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

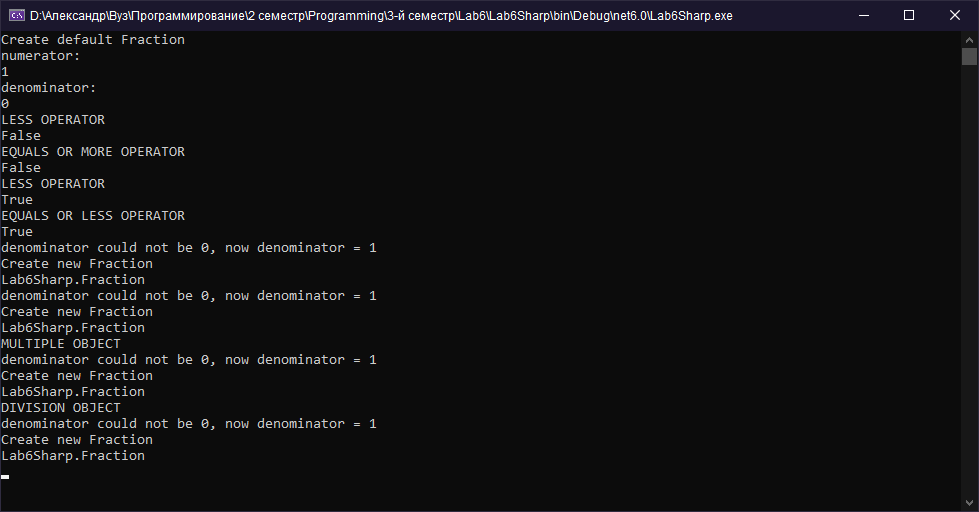
}

}

}

****

****

****