Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Уфимский государственный авиационный технический университет»

Кафедра ВВТиС

Отчет по лабораторной работе №7  
по дисциплине «Программирование»

«Перегрузка операций»

Группа ПМИ-150

Выполнил \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ Поташин А.В.

(дата) (подпись) (ФИО)

Проверил \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

(дата) (подпись) (ФИО)

Уфа 2019

# 1. Цели и задачи

Целью лабораторной работы является изучить способы создания безопасного кода с использованием классов обработки исключительных ситуаций.

Вариант 3.

1. Создать приложение, использующее в вычислениях функцию:



Предусмотреть обработку следующих исключительных ситуаций с генерацией исключения в виде соответствующей текстовой строки: деление на ноль, отрицательный аргумент под корнем.

2. Создать класс «целочисленная матрица». Предусмотреть логическое поле, позволяющее разрешить/запретить изменение размерности матрицы. Перегрузить операции присваивания и ввода/вывода из/в поток. Перегрузить арифметические операции сложения, вычитания, умножения в соответствии с правилами матричной алгебры. Создать метод класса для транспонирования матрицы. Создать сопутствующий класс для следующих исключительных ситуаций:

• несовпадающая размерность при присваивании для матрицы с неизменяемой размерностью – вернуть в классе-исключении размеры левой и правой матриц;

• несовпадающая размерность при арифметической операции – вернуть в классеисключении тип операции и размеры левой и правой матриц;

• транспонирование симметричной квадратной матрицы, то есть операция в принципе не имеет смысла.

Создать демонстрационное приложение и продемонстрировать корректное выполнение всех операций, а также обработку всех исключительных ситуаций.

# 2. Теоретическая часть

## Как создается защищенный блок кода?

## Как описывается процедура обработки конкретного исключения?

## Как генерируется исключение?

## Какие стандартные классы исключений вы знаете? (перечислить не менее трех)

## Какой класс является базовым для всех стандартных классов исключений?

## Как можно ограничить список исключений, которые могут генерироваться в функции?

## 2.1. Ответы на контрольные вопросы

## Try{}

1. В защищенном блоке с помощью throw выбрасывается некоторое сообщение, перехватываемое catch(){} блоком, в котором уже обрабатывается сама ситуация.
2. Throw
3. logic\_error, range\_error, length\_error
4. exception
5. перечислить список допустимых исключений в throw

# 3. ПРАКТИЧЕСКАЯ ЧАСТЬ

**4.1**

**Lqb2\_3.cpp**

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <iostream>

using namespace std;

double function(double x, double b) {

try {

if (sinh(x - b) == 0) {

cout << "Division by zero!" << endl;

throw 1;

}

if (cosh(x + b) / sinh(x - b) < 0) {

cout << "Negative value under the root!" << endl;

throw 2;

}

return sqrt(cosh(x + b) / sinh(x - b));

}

catch (int error) {

cout << "Error catched! Type code : ";

return error;

}

}

int main()

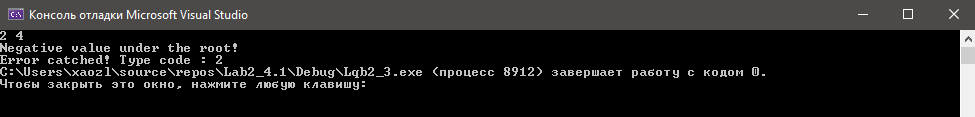
{

double a, b;

cin >> a >> b;

cout << function(a, b);

return 0;

}  

**4.2**

**Lab2\_4.2.cpp**

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include "Matrix.h"

#include "MatrixOperation.h"

#include "MatrixException.h"

#include <iostream>

#include <random>

#include <ctime>

int main()

{

srand((unsigned int)time(NULL));

MatrixOperation\* matr = new MatrixOperation();

MatrixOperation\* matrForChange = new MatrixOperation(3, 2, false);

MatrixOperation\* matrNotForChange = new MatrixOperation(4, 2, true);

cout << "Matrix generation unit:" << endl;

cout << \*matr;

cout << \*matrForChange;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Matrix enter unit:" << endl;

cin >> \*matrForChange;

cout << \*matrForChange;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Matrix operation unit:" << endl;

\*matrForChange = \*matr;

cout << "operator '=' :" << \*matrForChange;

MatrixOperation a = \*matrForChange + \*matrForChange;

cout << "operator '+' :" << a;

MatrixOperation d = \*matrForChange - \*matrForChange;

cout << "operator '-' :" << d;

MatrixOperation\* matrForMult = new MatrixOperation(2, 3, true);

cout << \*matrForMult;

cout << \*matrNotForChange;

MatrixOperation f = \*matrNotForChange \* \*matrForMult;

cout << "operator '\*' :" << f;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Matrix transparate unit:" << endl;

MatrixOperation\* matrForTransparate = new MatrixOperation(6, 3, false);

cout << \*matrForTransparate;

MatrixOperation c = matrForTransparate->transposition();

cout << c;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Matrix exeption unit:" << endl;

MatrixOperation b;

try {

b = \*matrForChange + \*matrNotForChange;

}

catch (MatrixException & ex) {

ex.showMessage();

}

try {

b = \*matrForChange - \*matrNotForChange;

}

catch (MatrixException & ex) {

ex.showMessage();

}

try {

\*matrNotForChange = \*matrForMult;

}

catch (MatrixException & ex) {

ex.showMessage();

}

try {

cin >> \*matrNotForChange;

}

catch (MatrixException & ex) {

ex.showMessage();

}

try {

\*matr\*\* matrNotForChange;

}

catch (MatrixException & ex) {

ex.showMessage();

}

return 0;

}

**Matrix.h**

#pragma once

#include <iostream>

#include <math.h>

#include <random>

#include <ctime>

using namespace std;

class Matrix

{

public:

Matrix();

Matrix(int);

Matrix(int, bool);

Matrix(int, int);

Matrix(int, int, bool);

~Matrix();

void setOrder(int, int);

int random(int, int);

void fillMatrix();

bool getChangeStatus();

protected:

bool staticMatrix;

int lines, colums;

int\*\* matrix;

};

**Matrix.cpp**

#include "Matrix.h"

#include <iostream>

#include <math.h>

#include <random>

#include <ctime>

using namespace std;

int Matrix::random(int a, int b) {

int minimum = min(a, b);

int maximum = max(a, b);

return rand() % (maximum - minimum) + minimum;

};

void Matrix::setOrder(int m, int n) {

this->lines = m;

this->colums = n;

}

void Matrix::fillMatrix() {

for (int i = 0; i < this->lines; i++)

{

for (int j = 0; j < this->colums; j++)

{

this->matrix[i][j] = random(-9, 9);

}

}

}

bool Matrix::getChangeStatus() {

return this->staticMatrix;

}

Matrix::Matrix()

{

//cout << " costruct " << this << endl;

this->staticMatrix = true;

this->setOrder(3, 3);

matrix = new int\* [this->lines];

for (int i = 0; i < this->lines; i++)

{

matrix[i] = new int[this->colums];

}

this->fillMatrix();

}

Matrix::Matrix(int n)

{

//cout << " costruct n" << this << endl;

this->staticMatrix = true;

this->setOrder(n, n);

matrix = new int\* [this->lines];

for (int i = 0; i < this->lines; i++)

{

matrix[i] = new int[this->colums];

}

this->fillMatrix();

}

Matrix::Matrix(int n, bool change)

{

//cout << " costruct n change" << this << endl;

this->staticMatrix = change;

this->setOrder(n, n);

matrix = new int\*[this->lines];

for (int i = 0; i < this->lines; i++)

{

matrix[i] = new int[this->colums];

}

this->fillMatrix();

}

Matrix::Matrix(int m, int n)

{

//cout << " costruct n m" << this << endl;

this->staticMatrix = true;

this->setOrder(m, n);

matrix = new int\* [this->lines];

for (int i = 0; i < this->lines; i++)

{

matrix[i] = new int[this->colums];

}

this->fillMatrix();

}

Matrix::Matrix(int m, int n, bool change)

{

//cout << " costruct n m change" << this << endl;

this->staticMatrix = change;

this->setOrder(m, n);

matrix = new int\* [this->lines];

for (int i = 0; i < this->lines; i++)

{

matrix[i] = new int[this->colums];

}

this->fillMatrix();

}

Matrix::~Matrix()

{

//cout << " destruct called!" << this << endl;

for (int i = 0; i < this->lines; i++)

{

delete[]matrix[i];

}

delete[]matrix;

}

**MatrixOperation.h**

#pragma once

#include "MatrixException.h"

#include "Matrix.h"

#include <iostream>

#include <math.h>

#include <random>

#include <ctime>

class MatrixOperation :

protected Matrix

{

public:

MatrixOperation operator+(const MatrixOperation&);

MatrixOperation operator-(const MatrixOperation&);

MatrixOperation operator\*(const MatrixOperation&);

MatrixOperation& operator=(const MatrixOperation&);

MatrixOperation transposition();

friend ostream& operator<<(ostream&, MatrixOperation&);

friend istream& operator>>(istream&, MatrixOperation&);

MatrixOperation();

MatrixOperation(int);

MatrixOperation(int, bool);

MatrixOperation(int, int);

MatrixOperation(int, int, bool);

MatrixOperation(const MatrixOperation&);

~MatrixOperation();

};

**MatrixOperation.cpp**

#include "MatrixOperation.h"

#include "MatrixException.h"

#include <iostream>

#include <math.h>

#include <random>

#include <ctime>

using namespace std;

MatrixOperation::MatrixOperation() {}

MatrixOperation::MatrixOperation(int n) : Matrix(n) {}

MatrixOperation::MatrixOperation(int n, bool change) : Matrix(n, change) {}

MatrixOperation::MatrixOperation(int n, int m) : Matrix(n, m) {}

MatrixOperation::MatrixOperation(int n, int m, bool change) : Matrix(n, m, change) {}

MatrixOperation::MatrixOperation(const MatrixOperation& mat) {

this->staticMatrix = mat.staticMatrix;

this->setOrder(mat.lines, mat.colums);

matrix = new int\* [lines];

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

matrix[i] = new int[colums];

}

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

{

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

matrix[i][j] = mat.matrix[i][j];

}

}

MatrixOperation::~MatrixOperation() {}

ostream& operator<<(ostream& stream, MatrixOperation& matrix) {

cout << "Matrix out: " << endl;

cout.setf(ios::showpos);

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

{

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

{

stream << " " << matrix.matrix[i][j] << " ";

}

cout << endl;

}

cout.unsetf(ios::showpos);

cout << endl;

return stream;

}

istream& operator>>(istream& stream, MatrixOperation &matrix) {

cout << "Matrix enter: " << endl;

if (matrix.getChangeStatus()) {

throw MatrixException((char\*)"This matrix can't be changed! ", (char\*)"operator >>, string 57.");

}

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

{

delete[]matrix.matrix[i];

}

delete[]matrix.matrix;

int m, n;

cout << "Enter number of lines: " << endl;

cin >> m;

cout << "Enter number of colums: " << endl;

cin >> n;

matrix.setOrder(m, n);

matrix.matrix = new int\* [matrix.lines];

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

{

matrix.matrix[i] = new int[matrix.colums];

}

matrix.fillMatrix();

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

{

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

{

stream >> matrix.matrix[i][j];

}

cout << endl;

}

cout << endl;

return stream;

}

MatrixOperation MatrixOperation::operator+(const MatrixOperation& matr) {

if (this->colums != matr.colums || this->lines != matr.lines)

{

throw MatrixException((char\*)"invalid matrix dimension", (char\*)"operator +, string 97.");

}

MatrixOperation\* result = new MatrixOperation(matr.lines, matr.colums, matr.staticMatrix);

for (int i = 0; i < result->lines; i++)

{

for (int j = 0; j < result->colums; j++)

{

result->matrix[i][j] = matr.matrix[i][j] + this->matrix[i][j];

}

cout << endl;

}

return \*result;

}

MatrixOperation MatrixOperation::operator-(const MatrixOperation& matr) {

if (this->colums != matr.colums || this->lines != matr.lines)

{

throw MatrixException((char\*)"invalid matrix dimension", (char\*)"operator -, string 116.");

}

MatrixOperation\* result = new MatrixOperation(matr.lines, matr.colums, matr.staticMatrix);

for (int i = 0; i < result->lines; i++)

{

for (int j = 0; j < result->colums; j++)

{

result->matrix[i][j] = matr.matrix[i][j] - this->matrix[i][j];

}

cout << endl;

}

return \*result;

}

MatrixOperation& MatrixOperation::operator=(const MatrixOperation& matr) {

if (this->getChangeStatus()) {

throw MatrixException((char\*)"This matrix can't be changed! ", (char\*)"operator =, string 134.");

}

for (int i = 0; i < this->lines; i++)

{

delete[]this->matrix[i];

}

delete[]this->matrix;

this->staticMatrix = matr.staticMatrix;

this->setOrder(matr.lines, matr.colums);

this->matrix = new int\* [this->lines];

for (int i = 0; i < this->lines; i++)

{

this->matrix[i] = new int[this->colums];

}

for (int i = 0; i < this->lines; i++)

{

for (int j = 0; j < this->colums; j++)

{

this->matrix[i][j] = matr.matrix[i][j];

}

}

return \*this;

}

MatrixOperation MatrixOperation::operator\*(const MatrixOperation& matrix) {

cout << " Operator '\*' called" << endl;

if (this->colums != matrix.lines)

{

throw MatrixException((char\*)"Тot suitable dimensions of matrices, multiplication is impossible! ", (char\*)"operator \*, string 164.");

}

MatrixOperation\* result = new MatrixOperation(this->lines, matrix.colums, false);

for (int i = 0; i < result->lines; i++)

{

for (int j = 0; j < result->colums; j++)

{

result->matrix[i][j] = 0;

for (int k = 0; k < this->colums; k++) {

result->matrix[i][j] += this->matrix[i][k] \* matrix.matrix[k][j];

}

}

}

return \*result;

}

MatrixOperation MatrixOperation::transposition() {

MatrixOperation\* result = new MatrixOperation(this->colums, this->lines, false);

for (int i = 0; i < result->lines; i++)

{

for (int j = 0; j < result->colums; j++)

{

result->matrix[i][j] = this->matrix[j][i];

}

}

return \*result;

}

**MatrixException.h**

#pragma once

#include <exception>

#include <iostream>

using namespace std;

class MatrixException : public std::exception

{

public:

MatrixException(char\*, char\* str);

~MatrixException();

char\* message;

char\* unit;

void showMessage();

};

**MatrixException.cpp**

#include "MatrixException.h"

#include <exception>

#include <iostream>

using namespace std;

void MatrixException::showMessage() {

cout << "ATENTION, exception catched: " << endl;

cout << this->message << " from operation : " << this->unit << endl;

}

MatrixException::MatrixException(char\* str, char\* place)

{

message = str;

unit = place;

}

MatrixException::~MatrixException()

{

}

