**Міністерство освіти і науки України**

**Національний університет “Львівська політехніка”**

**Інститут комп’ютерних наук та інформаційних технологій**

***Кафедра ПЗ***

**Звіт**

до лабораторної роботи №12

на тему «Виняткові ситуації в мові програмування C++»

з дисципліни “Об’єктно-орієнтоване програмування

**Виконав:**

                                       студент групи ПЗ-11

                     Симан Василь

**Перевірив**:

                                                                                                                            доц. Коротєєва Т.О.

Львів

**2022**

**Лабораторна робота №12**

**Виняткові ситуації в мові програмування C++**

**Тема**: Виняткові ситуації в мові програмування С++.

**Мета:** Ознайомитися з синтаксисом та принципами використання винятків, нав­читися передбачати виняткові ситуації, які можуть виникнути в процесі роботи програмного забезпечення, а також навчитися їх перехоплювати та опрацьовувати.

**Теоретичні відомості**

В основі обробки виняткових ситуацій у мові С++ лежать три ключових слова: try, catch і throw.

Якщо програміст підозрює, що визначений фрагмент програми може спровокувати помилку, він повинний занурити цю частину коду в блок try. Необхідно мати на увазі, що зміст помилки (за винятком стандартних ситуацій) визначає сам програміст. Це значить, що програміст може задати будь-яку умову, що приведе до створення виняткової ситуації. Після цього необхідно вказати, у яких умовах варто генерувати виняткову ситуацію. Для цієї мети призначене ключове слово throw. І нарешті, виняткову ситуацію потрібно перехопити й обробити в блоці catch. Ось як виглядає ця конструкція.

try

{

*//* *Тіло блоку try*

if(*умова*) throw *виняткова ситуація*

}

catch(*тип1 аргумент*)

{

*// Тіло блоку catch*

}

catch(*тип2 аргумент*)

{

*// Тіло блоку catch*

}

.

.

.

catch(*тип N аргумент*)

{

*// Тіло блоку catch*

}

Розмір блоку try не обмежений. У нього можна занурити як один оператор, так і цілу програму. Один блок try можна зв'язати з довільною кількістю блоків catch. Оскільки кожен блок catch відповідає окремому типу виняткової ситуації, програма сама визначить, який з них виконати. У цьому випадку інші блоки catch не виконуються. Кожен блок catch має аргумент, що приймає визначене значення. Цей аргумент може бути об'єктом будь-якого типу.

Якщо програма виконана правильно й у блоці try не виникло жодної виняткової ситуації, усі блоки catch будуть зігноровані. Якщо в програмі виникла подія, що програміст вважає небажаним, оператор throw генерує виняткову ситуацію. Для цього оператор throw повинний знаходитися усередині блоку try або усередині функції, викликуваної усередині блоку try.

**Індивідуальне завдання**

**Варіант 1:**

Реалізувати калькулятор для матриць довільного розміру (розмір задається користувачем). Калькулятор повинен вміти виконувати додавання, віднімання та множення матриць між собою, а також повинен вміти знаходити обернену матрицю. Роботу з матрицями необхідно здійснювати за допомогою розробленого класу MyMatrix, а місце під елементи матриці повинно виділятися в  динамічній пам’яті. Програма повинна перехоплювати та опрацьовувати такі виняткові ситуації: а) вихід за межі масиву,  б) помилкове введення користувачем літерного символу замість числа при введенні матриці, в) некоректна розмірність матриць,   г) ще дві виняткові ситуації передбачити самостійно.

Всі функції повинні містити список винятків, які вони можуть генерувати.

**Код програми**

Файл mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTableWidget>

#include <QTableWidgetItem>

#include <QMessageBox>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*private*:

Ui::MainWindow \*ui;

*private* slots:

void **SetSizePressed**();

void **AddMatrices**();

void **SubtractMatrices**();

void **MultiplyMatrices**();

void **TransponateMatrix1**();

void **TransponateMatrix2**();

};

#endif *//* *MAINWINDOW\_H*

Файл mymatrix.h:

#ifndef MYMATRIX\_H

#define MYMATRIX\_H

#include <QMainWindow>

*using* *namespace* std;

*class* **MyMatrix**{

*private*:

int rowNum;

int columnNum;

vector<vector<int>> Matrix;

*public*:

**MyMatrix**();

**MyMatrix**(vector<vector<int>> mat);

**MyMatrix**(int rows, int columns);

int **GetRowNum**(){*return* rowNum;}

int **GetColumnNum**(){*return* columnNum;}

int **GetNum**(int r, int c){*return* Matrix[r][c];}

void **SetNum**(int r, int c, int n){Matrix[r][c] = n;}

void **Transponate**();

MyMatrix *operator*+(MyMatrix &left);

MyMatrix *operator*-(MyMatrix &left);

MyMatrix *operator*\*(MyMatrix &left);

};

*class* **MyException**

{

*private*:

QString myError;

*public*:

**MyException**(QString error) : myError(error){};

QString **getError**(){*return* myError;}

};

#endif *//* *MYMATRIX\_H*

Файл main.cpp:

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, *argv*);

MainWindow w;

w.show();

*return* a.exec();

}

Файл mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "mymatrix.h"

int m1, m2, n1, n2;

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

connect(ui->SetSize, SIGNAL (released()), *this*, SLOT (SetSizePressed()));

connect(ui->SumButt, SIGNAL (released()), *this*, SLOT (AddMatrices()));

connect(ui->DiffButt, SIGNAL (released()), *this*, SLOT (SubtractMatrices()));

connect(ui->MultiplyButt, SIGNAL (released()), *this*, SLOT (MultiplyMatrices()));

connect(ui->Trans1, SIGNAL (released()), *this*, SLOT (TransponateMatrix1()));

connect(ui->Trans2, SIGNAL (released()), *this*, SLOT (TransponateMatrix2()));

}

void MainWindow::**TransponateMatrix2**(){

*try*{

QTableWidget \*mt1 = ui->Matrx2;

MyMatrix mat1(m2, n2);

*for* (int i = 0; i < mat1.GetRowNum(); ++i){

*for* (int j = 0; j < mat1.GetColumnNum(); ++j){

*if*(!mt1->item(i, j)){

mat1.SetNum(i, j, 0);

} *else* {

bool it;

mat1.SetNum(i, j, mt1->item(i, j)->text().toInt(*&it*));

*if*(!it){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

}

}

}

mat1.Transponate();

ui->ResultMatrx->setRowCount(mat1.GetColumnNum());

ui->ResultMatrx->setColumnCount(mat1.GetRowNum());

*for* (int i = 0; i < mat1.GetColumnNum(); ++i){

*for* (int j = 0; j < mat1.GetRowNum(); ++j){

QTableWidgetItem \*itm = *new* QTableWidgetItem(QString::number(mat1.GetNum(i, j)));

ui->ResultMatrx->setItem(i, j, *itm*);

}

}

}

*catch*(MyException &exception){

QMessageBox error;

error.setText("Error. " + exception.getError());

error.*exec*();

}

}

void MainWindow::**TransponateMatrix1**(){

*try*{

QTableWidget \*mt1 = ui->Matrx1;

MyMatrix mat1(m1, n1);

*for* (int i = 0; i < mat1.GetRowNum(); ++i){

*for* (int j = 0; j < mat1.GetColumnNum(); ++j){

*if*(!mt1->item(i, j)){

mat1.SetNum(i, j, 0);

} *else* {

bool it;

mat1.SetNum(i, j, mt1->item(i, j)->text().toInt(*&it*));

*if*(!it){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

}

}

}

mat1.Transponate();

ui->ResultMatrx->setRowCount(mat1.GetColumnNum());

ui->ResultMatrx->setColumnCount(mat1.GetRowNum());

*for* (int i = 0; i < mat1.GetColumnNum(); ++i){

*for* (int j = 0; j < mat1.GetRowNum(); ++j){

QTableWidgetItem \*itm = *new* QTableWidgetItem(QString::number(mat1.GetNum(i, j)));

ui->ResultMatrx->setItem(i, j, *itm*);

}

}

}

*catch*(MyException &exception){

QMessageBox error;

error.setText("Error. " + exception.getError());

error.*exec*();

}

}

void MainWindow::**MultiplyMatrices**(){

*try*{

QTableWidget \*mt1 = ui->Matrx1;

QTableWidget \*mt2 = ui->Matrx2;

MyMatrix mat1(m1, n1), mat2(m2, n2);

*for* (int i = 0; i < mat1.GetRowNum(); ++i){

*for* (int j = 0; j < mat1.GetColumnNum(); ++j){

*if*(!mt1->item(i, j)){

mat1.SetNum(i, j, 0);

} *else* {

bool it;

mat1.SetNum(i, j, mt1->item(i, j)->text().toInt(*&it*));

*if*(!it){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

}

}

}

*for* (int i = 0; i < mat2.GetRowNum(); ++i){

*for* (int j = 0; j < n2; ++j){

*if*(!mt2->item(i, j)){

mat2.SetNum(i, j, 0);

} *else* {

bool it;

mat2.SetNum(i, j, mt2->item(i, j)->text().toInt(*&it*));

*if*(!it){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

}

}

}

MyMatrix result = mat1 \* mat2;

ui->ResultMatrx->setRowCount(result.GetRowNum());

ui->ResultMatrx->setColumnCount(result.GetColumnNum());

*for* (int i = 0; i < result.GetRowNum(); ++i){

*for* (int j = 0; j < result.GetColumnNum(); ++j){

QTableWidgetItem \*itm = *new* QTableWidgetItem(QString::number(result.GetNum(i, j)));

ui->ResultMatrx->setItem(i, j, *itm*);

}

}

}

*catch*(MyException &exception){

QMessageBox error;

error.setText("Error. " + exception.getError());

error.*exec*();

}

}

void MainWindow::**SubtractMatrices**(){

*try*{

QTableWidget \*mt1 = ui->Matrx1;

QTableWidget \*mt2 = ui->Matrx2;

MyMatrix mat1(m1, n1), mat2(m2, n2);

*for* (int i = 0; i < mat1.GetRowNum(); ++i){

*for* (int j = 0; j < mat1.GetColumnNum(); ++j){

*if*(!mt1->item(i, j)){

mat1.SetNum(i, j, 0);

} *else* {

bool it;

mat1.SetNum(i, j, mt1->item(i, j)->text().toInt(*&it*));

*if*(!it){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

}

}

}

*for* (int i = 0; i < mat2.GetRowNum(); ++i){

*for* (int j = 0; j < n2; ++j){

*if*(!mt2->item(i, j)){

mat2.SetNum(i, j, 0);

} *else* {

bool it;

mat2.SetNum(i, j, mt2->item(i, j)->text().toInt(*&it*));

*if*(!it){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

}

}

}

MyMatrix result = mat1 - mat2;

ui->ResultMatrx->setRowCount(result.GetRowNum());

ui->ResultMatrx->setColumnCount(result.GetColumnNum());

*for* (int i = 0; i < result.GetRowNum(); ++i){

*for* (int j = 0; j < result.GetColumnNum(); ++j){

QTableWidgetItem \*itm = *new* QTableWidgetItem(QString::number(result.GetNum(i, j)));

ui->ResultMatrx->setItem(i, j, *itm*);

}

}

}

*catch*(MyException &exception){

QMessageBox error;

error.setText("Error. " + exception.getError());

error.*exec*();

}

}

void MainWindow::**AddMatrices**(){

*try*{

QTableWidget \*mt1 = ui->Matrx1;

QTableWidget \*mt2 = ui->Matrx2;

MyMatrix mat1(m1, n1), mat2(m2, n2);

*for* (int i = 0; i < mat1.GetRowNum(); ++i){

*for* (int j = 0; j < mat1.GetColumnNum(); ++j){

*if*(!mt1->item(i, j)){

mat1.SetNum(i, j, 0);

} *else* {

bool it;

mat1.SetNum(i, j, mt1->item(i, j)->text().toInt(*&it*));

*if*(!it){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

}

}

}

*for* (int i = 0; i < mat2.GetRowNum(); ++i){

*for* (int j = 0; j < n2; ++j){

*if*(!mt2->item(i, j)){

mat2.SetNum(i, j, 0);

} *else* {

bool it;

mat2.SetNum(i, j, mt2->item(i, j)->text().toInt(*&it*));

*if*(!it){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

}

}

}

MyMatrix result = mat1 + mat2;

ui->ResultMatrx->setRowCount(result.GetRowNum());

ui->ResultMatrx->setColumnCount(result.GetColumnNum());

*for* (int i = 0; i < result.GetRowNum(); ++i){

*for* (int j = 0; j < result.GetColumnNum(); ++j){

QTableWidgetItem \*itm = *new* QTableWidgetItem(QString::number(result.GetNum(i, j)));

ui->ResultMatrx->setItem(i, j, *itm*);

}

}

}

*catch*(MyException &exception){

QMessageBox error;

error.setText("Error. " + exception.getError());

error.*exec*();

}

}

void MainWindow::**SetSizePressed**(){

*try*{

QString m\_1 = ui->m1->text();

QString m\_2 = ui->m2->text();

QString n\_1 = ui->n1->text();

QString n\_2 = ui->n2->text();

*if*(m\_1.isEmpty() || m\_2.isEmpty() || n\_1.isEmpty() || n\_2.isEmpty()){

*throw* MyException("Empty field. Enter all values and try again");

}

bool M1, M2, N1, N2;

m1 = m\_1.toInt(*&M1*);

m2 = m\_2.toInt(*&M2*);

n1 = n\_1.toInt(*&N1*);

n2 = n\_2.toInt(*&N2*);

*if*(!M1 || !M2 || !N1 || !N2){

*throw* MyException("Letter detected. Please, enter a number and try again");

}

ui->Matrx1->setRowCount(m1);

ui->Matrx1->setColumnCount(n1);

ui->Matrx2->setRowCount(m2);

ui->Matrx2->setColumnCount(n2);

ui->m1->setText("");

ui->m2->setText("");

ui->n1->setText("");

ui->n2->setText("");

}

*catch* (MyException &exception){

QMessageBox error;

error.setText("Error. " + exception.getError());

error.*exec*();

}

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

Файл mymatrix.cpp:

#include "mymatrix.h"

MyMatrix::**MyMatrix**()

{

}

MyMatrix::**MyMatrix**(int rows, int columns){

rowNum = rows;

columnNum = columns;

Matrix.resize(rowNum);

*for* (int i = 0; i < rowNum; ++i){

Matrix[i].resize(columnNum);

}

}

void MyMatrix::**Transponate**(){

*//генерація* *винятку* *при* *неправильному* *розмірі* *матриці*

*if*(rowNum < 0 || columnNum < 0){

*throw* MyException("Bad matrix size. Please, try again.");

}

vector<vector<int>> temp;

temp.resize(columnNum);

*for* (int i = 0; i < columnNum; ++i){

temp[i].resize(rowNum);

}

*for*(int i = 0; i < rowNum; ++i){

*for*(int j = 0; j < columnNum; ++j){

*//генерація* *винятку* *при* *виході* *за* *межі* *масиву*

*if*(i == rowNum || j == columnNum){

*throw* MyException("Going out the matrix");

}

temp[j][i] = Matrix[i][j];

}

}

Matrix = temp;

}

MyMatrix MyMatrix::*operator*\*(MyMatrix &left){

*//генерація* *винятку* *при* *неправильному* *розмірі* *матриці*

*if*(columnNum != left.rowNum || rowNum < 0 || left.rowNum < 0 || columnNum < 0 || left.columnNum < 0){

*throw* MyException("Bad matrix size. Please, try again.");

}

*static* MyMatrix result(rowNum, left.columnNum);

*for* (int i = 0; i < rowNum; i++) {

*for* ( int j = 0; j < columnNum; j++) {

int res = 0;

*for* (int k = 0; k < columnNum; k++) {

*//генерація* *винятку* *при* *виході* *за* *межі* *масиву*

*if* ( i == rowNum || j == columnNum || k == columnNum){

*throw* MyException("Going out the matrix");

}

res += *this*->GetNum(i, k) \* left.GetNum(k, j);

}

result.SetNum(i, j, res);

}

}

*return* result;

}

MyMatrix MyMatrix::*operator*+(MyMatrix &left){

*//генерація* *винятку* *при* *неправильному* *розмірі* *матриці*

*if*(rowNum != left.rowNum || columnNum != left.columnNum || rowNum < 0 || left.rowNum < 0 || columnNum < 0 || left.columnNum < 0){

*throw* MyException("Bad matrix size. Please, try again.");

}

*static* MyMatrix result(*this*->GetRowNum(), *this*->GetColumnNum());

*for* (int i = 0; i < *this*->GetRowNum(); ++i){

*for* (int j = 0; j < *this*->GetColumnNum(); ++j){

*//генерація* *винятку* *при* *виході* *за* *межі* *масиву*

*if* ( i == rowNum || j == columnNum){

*throw* MyException("Going out the matrix");

}

result.SetNum(i, j, (*this*->GetNum(i, j) + left.GetNum(i, j)));

}

}

*return* result;

}

MyMatrix MyMatrix::*operator*-(MyMatrix &left){

*//генерація* *винятку* *при* *неправильному* *розмірі* *матриці*

*if*(rowNum != left.rowNum || columnNum != left.columnNum || rowNum < 0 || left.rowNum < 0 || columnNum < 0 || left.columnNum < 0){

*throw* MyException("Bad matrix size. Please, try again.");

}

*static* MyMatrix result(*this*->GetRowNum(), *this*->GetColumnNum());

*for* (int i = 0; i < *this*->GetRowNum(); ++i){

*for* (int j = 0; j < *this*->GetColumnNum(); ++j){

*//генерація* *винятку* *при* *виході* *за* *межі* *масиву*

*if* ( i == rowNum || j == columnNum){

*throw* MyException("Going out the matrix");

}

result.SetNum(i, j, (*this*->GetNum(i, j) - left.GetNum(i, j)));

}

}

*return* result;

}

Результат виконання програми:

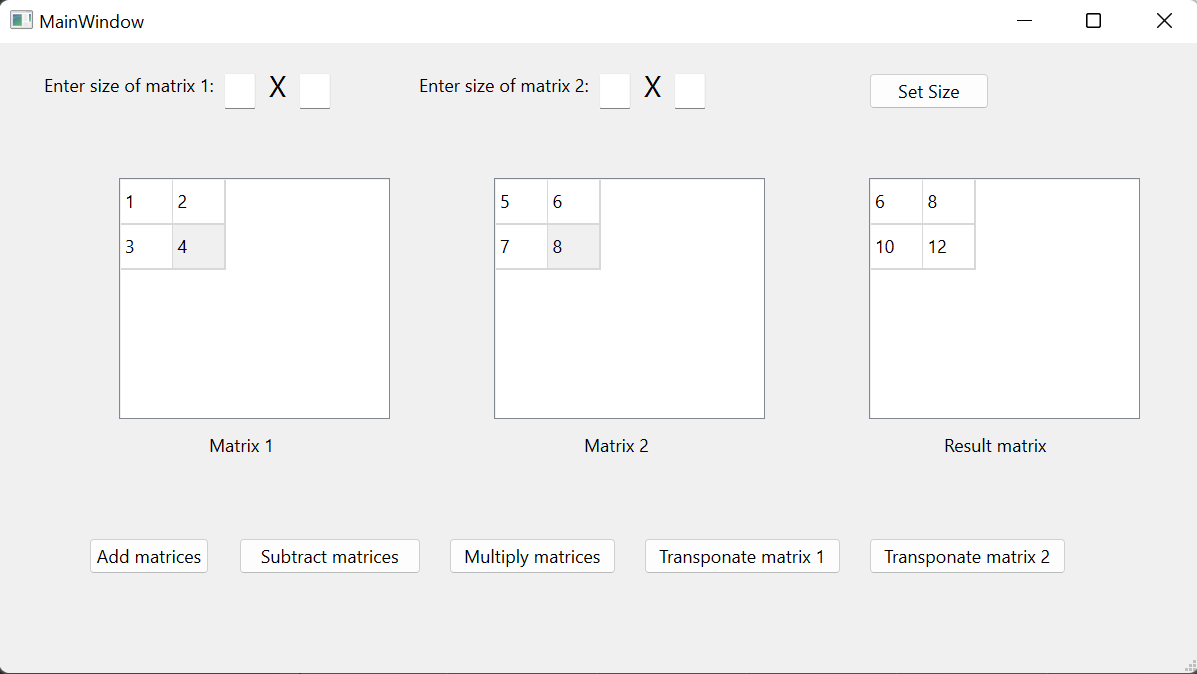


Рис. 1 результат суми двох матриць

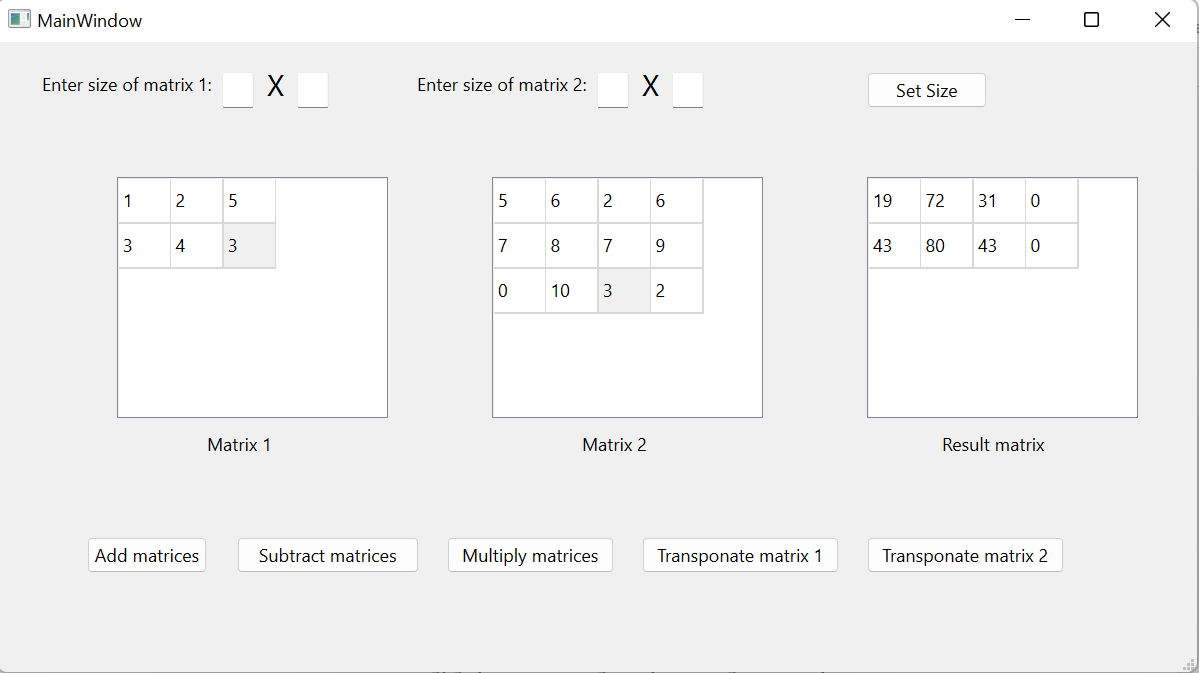


Рис. 2 результат множення матриць

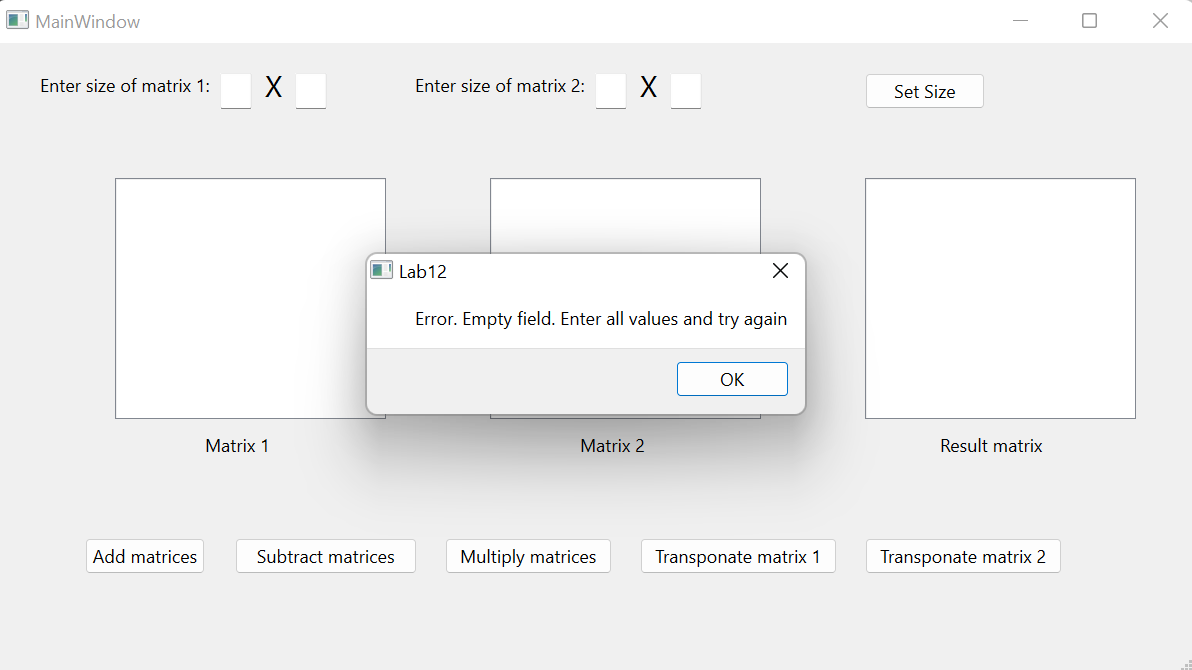


Рис. 3 результат виконання винятку про незаповненість поля розміру матриці

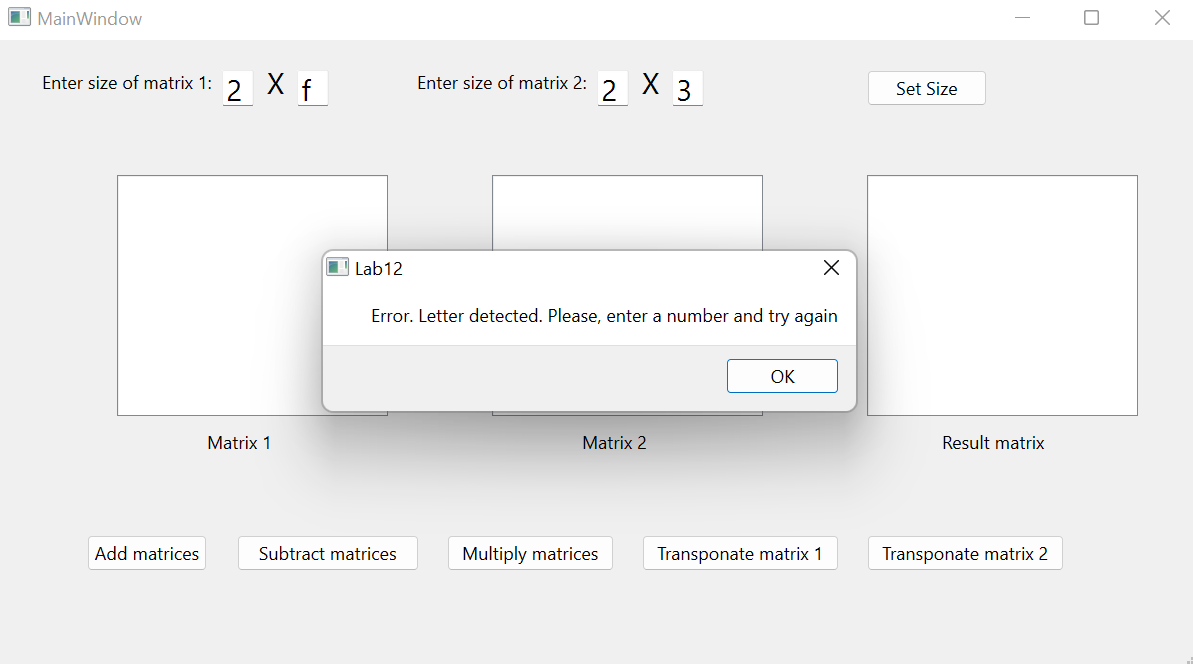


Рис.4 результат виконання винятку про введення літери замість числа в поле задання розміру матриці

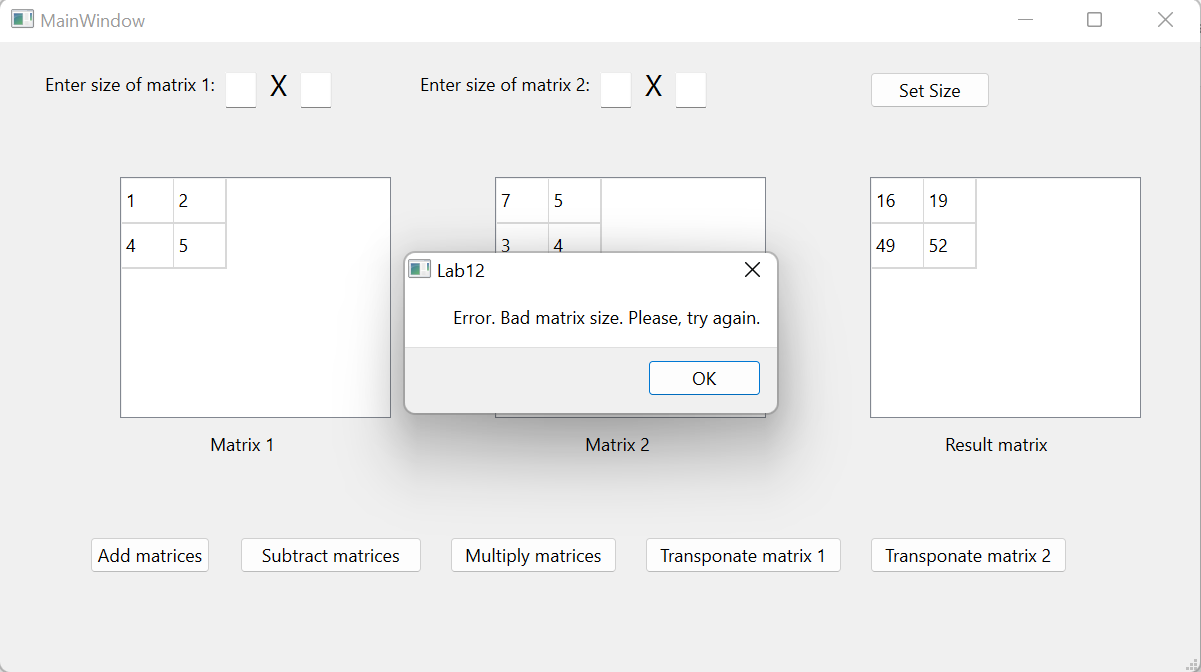


Рис. 5 результат виконання винятку про непідходящі розміри матриць для виконання операції множення

**Висновок:** Виконуючи лабораторну роботу №12 я ознайомився з синтаксисом та принципами використання винятків, нав­чився передбачати виняткові ситуації, які можуть виникнути в процесі роботи програмного забезпечення, а також навчився їх перехоплювати та опрацьовувати.