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 С++»

з курсу «Об’єктно-орієнтоване програмування»

Виконала:

Ст. групи ПЗ-11

Солтисюк Д.А.

Перевірила:

доц. Коротєєва Т.О.

Львів

2022

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:

У ході виконання програми можуть виявитися різні помилки. Вони можуть бути позв'язані з неправильним програмуванням (наприклад, вихід індексу масиву за межі припустимого чи переповнення пам'яті), а іноді їхня причина не

залежить від програміста (скажемо, розрив зв'язку при мережевому з'єднанні). У кожній з цих ситуацій реакція програми непередбачена. Іноді вона завершує виконання, і лише після закінчення деякого інтервалу часу починають позначатися наслідку помилки, а частіше програма негайно припиняє роботу, піддаючи ризику дані, що знаходяться в пам'яті чи у файлі. Якщо не передбачити акуратне завершення роботи, використовуючи обробку виняткових ситуацій, результати можуть виявитися неприємними.

В подальшому ми будемо називати винятковою ситуацією будь-яку подію, що вимагає особливої обробки. При цьому зовсім неважливо, чи є ця подія фатальною чи простою помилкою. Перевірка умов, що описують виняткову ситуацію, і реакція на її виникнення називається обробкою виняткової ситуації. Ця задача покладається на оброблювача виняткової ситуації.

В основі обробки виняткових ситуацій у мові C++ 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.

Завдання для лабораторної роботи:

1. Ознайомитися з основними поняттями та синтаксисом мови C++, з метою передбачення та оброблення виняткових ситуацій.
2. Провести аналіз завдання (індивідуальні варіанти), визначити можливі виняткові ситуації, які можуть виникнути в процесі роботи програмного забезпечення.
3. Розробити програмне забезпечення для реалізації поставленої задачі.
4. Оформити і здати звіт про виконання лабораторної роботи. Звіт має містити варіант завдання, код розробленої програми, результати роботи програми (скріншоти), висновок.

Варіант 3.

Реалізувати калькулятор для дробів. Калькулятор повинен вміти виконувати додавання, віднімання, множення та ділення дробів, а також повинен вміти знаходити обернений дріб. Роботу з дробами необхідно здійснювати за допомогою розробленого класу Drib. Програма повинна перехоплювати та опрацьовувати такі виняткові ситуації: а) ділення на нуль, б) помилкове введення користувачем літерного символу замість числа при введенні дробу, в) переповнення, г) ще дві виняткові ситуації передбачити самостійно.

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

Результати:

fraction.h

```
#ifndef FRACTION_H
#define FRACTION_H
#include <QLineEdit>
#include <string>
#include <QRegularExpression>
using std::string;

class Fraction
{
    int nominator;
    int denominator;
public:
    Fraction() throw()
    {
        nominator=1;
        denominator=1;
    }
    Fraction(int n, int d)
    {
        nominator=n;
        denominator=d;
    }
    void Input(QLineEdit* nomin, QLineEdit* denom) noexcept(false)
    {
        if(nomin==nullptr||denom==nullptr)
            throw string("Error with QLineEdit* pointers!");
        if(nomin->text().contains(QRegularExpression("[^0-9.,]")))
            throw string("Nominator cannot contain non-number symbols");
        if(denom->text().contains(QRegularExpression("[^0-9.,]")))
            throw string("Denominator cannot contain non-number symbols");
        if(nomin->text().contains(QRegularExpression("[,.]")))
            throw 0.1;
        if(denom->text().contains(QRegularExpression("[,.]")))
            throw 0.1;
        if(nomin->text().toLongLong()>INT_MAX||nomin-
>text().toLongLong()<INT_MIN)
            throw QString("int memory overflow in nominator");
        if(denom->text().toLongLong()>INT_MAX||denom-
>text().toLongLong()<INT_MIN)
            throw QString("int memory overflow in denominator");
        nominator = nomin->text().toInt();
        denominator = denom->text().toInt();
        if(denominator==0)
            throw 0;
    }
    void Print(QLineEdit* nomin, QLineEdit* denom) noexcept(false)
    {
        if(nomin==nullptr||denom==nullptr)
            throw string("Error with QLineEdit* pointers!");
        nomin->setText(QString::number(nominator));
        denom->setText(QString::number(denominator));
    }
    Fraction operator+(Fraction& f) throw()
    {
```

```
        return Fraction(nominator*f.denominator+denominator*f.nominator,
denominator*f.denominator);
    }
    Fraction operator-(Fraction& f) throw()
    {
        return Fraction(nominator*f.denominator-denominator*f.nominator,
denominator*f.denominator);
    }
    Fraction operator*(Fraction& f) throw()
    {
        return Fraction(nominator*f.nominator, denominator*f.denominator);
    }
    Fraction operator/(Fraction& f) throw()
    {
        return Fraction(nominator*f.denominator, denominator*f.nominator);
    }
    Fraction Flip() noexcept(false)
    {
        if(nominator==0)
            throw 0;
        return Fraction(denominator, nominator);
    }
};

#endif // FRACTION_H
```

widget.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_Plus_clicked();

    void on_Minus_clicked();

    void on_Divide_clicked();

    void on_Product_clicked();

    void on_Flip_clicked();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

widget.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "Fraction.h"
#include <QMessageBox>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_Plus_clicked()
{
    try
    {
        Fraction f1;
        f1.Input(ui->nomin1, ui->denom1);
        Fraction f2;
        f2.Input(ui->nomin2, ui->denom2);
        ui->operation->setText("+");
        (f1+f2).Print(ui->nomin3, ui->denom3);
    }
    catch(string ex)
    {
        QMessageBox msgBox;
        msgBox.setText(QString::fromStdString(ex));
        msgBox.exec();
    }
    catch(int ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot divide by zero!");
        msgBox.exec();
    }
    catch(double ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot input non-integer numbers");
        msgBox.exec();
    }
    catch(QString ex)
    {
        QMessageBox msgBox;
        msgBox.setText(ex);
        msgBox.exec();
    }
    catch(...){}
}


void MainWindow::on_Minus_clicked()
{
    try
    {
```

```
        Fraction f1;
        f1.Input(ui->nomin1, ui->denom1);
        Fraction f2;
        f2.Input(ui->nomin2, ui->denom2);
        ui->operation->setText("-");
        (f1-f2).Print(ui->nomin3, ui->denom3);
    }
    catch(string ex)
    {
        QMessageBox msgBox;
        msgBox.setText(QString::fromStdString(ex));
        msgBox.exec();
    }
    catch(int ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot divide by zero!");
        msgBox.exec();
    }
    catch(double ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot input non-integer numbers");
        msgBox.exec();
    }
    catch(QString ex)
    {
        QMessageBox msgBox;
        msgBox.setText(ex);
        msgBox.exec();
    }
    catch(...){}
}


void MainWindow::on_Divide_clicked()
{
    try
    {
        Fraction f1;
        f1.Input(ui->nomin1, ui->denom1);
        Fraction f2;
        f2.Input(ui->nomin2, ui->denom2);
        ui->operation->setText("/");
        (f1/f2).Print(ui->nomin3, ui->denom3);
    }
    catch(string ex)
    {
        QMessageBox msgBox;
        msgBox.setText(QString::fromStdString(ex));
        msgBox.exec();
    }
    catch(int ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot divide by zero!");
        msgBox.exec();
    }
    catch(double ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot input non-integer numbers");
```

```
        msgBox.exec();
    }
    catch(QString ex)
    {
        QMessageBox msgBox;
        msgBox.setText(ex);
        msgBox.exec();
    }
    catch(...){}
}


void MainWindow::on_Product_clicked()
{
    try
    {
        Fraction f1;
        f1.Input(ui->nomin1, ui->denom1);
        Fraction f2;
        f2.Input(ui->nomin2, ui->denom2);
        ui->operation->setText("*");
        (f1*f2).Print(ui->nomin3, ui->denom3);
    }
    catch(string ex)
    {
        QMessageBox msgBox;
        msgBox.setText(QString::fromStdString(ex));
        msgBox.exec();
    }
    catch(int ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot divide by zero!");
        msgBox.exec();
    }
    catch(double ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot input non-integer numbers");
        msgBox.exec();
    }
    catch(QString ex)
    {
        QMessageBox msgBox;
        msgBox.setText(ex);
        msgBox.exec();
    }
    catch(...){}
}


void MainWindow::on_Flip_clicked()
{
    try
    {
        Fraction f1;
        f1.Input(ui->nomin1, ui->denom1);
        ui->operation->setText("");
        ui->nomin2->clear();
        ui->denom2->clear();
        (f1.Flip()).Print(ui->nomin3, ui->denom3);
    }
```

```
    catch(string ex)
    {
        QMessageBox msgBox;
        msgBox.setText(QString::fromStdString(ex));
        msgBox.exec();
    }
    catch(int ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot divide by zero!");
        msgBox.exec();
    }
    catch(double ex)
    {
        QMessageBox msgBox;
        msgBox.setText("You cannot input non-integer numbers");
        msgBox.exec();
    }
    catch(QString ex)
    {
        QMessageBox msgBox;
        msgBox.setText(ex);
        msgBox.exec();
    }
    catch(...){}
}
```

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Результати виконання програми:

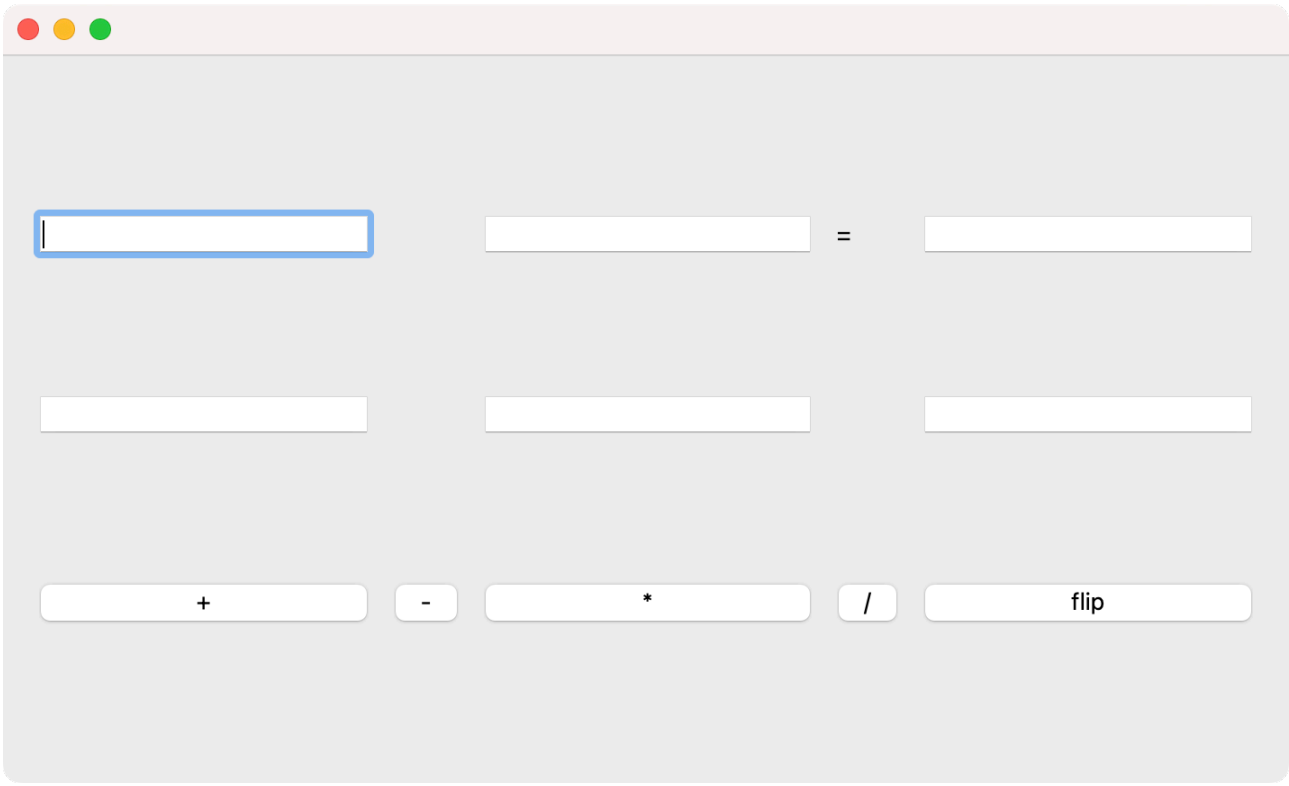

Рис. 1. Запуск програми

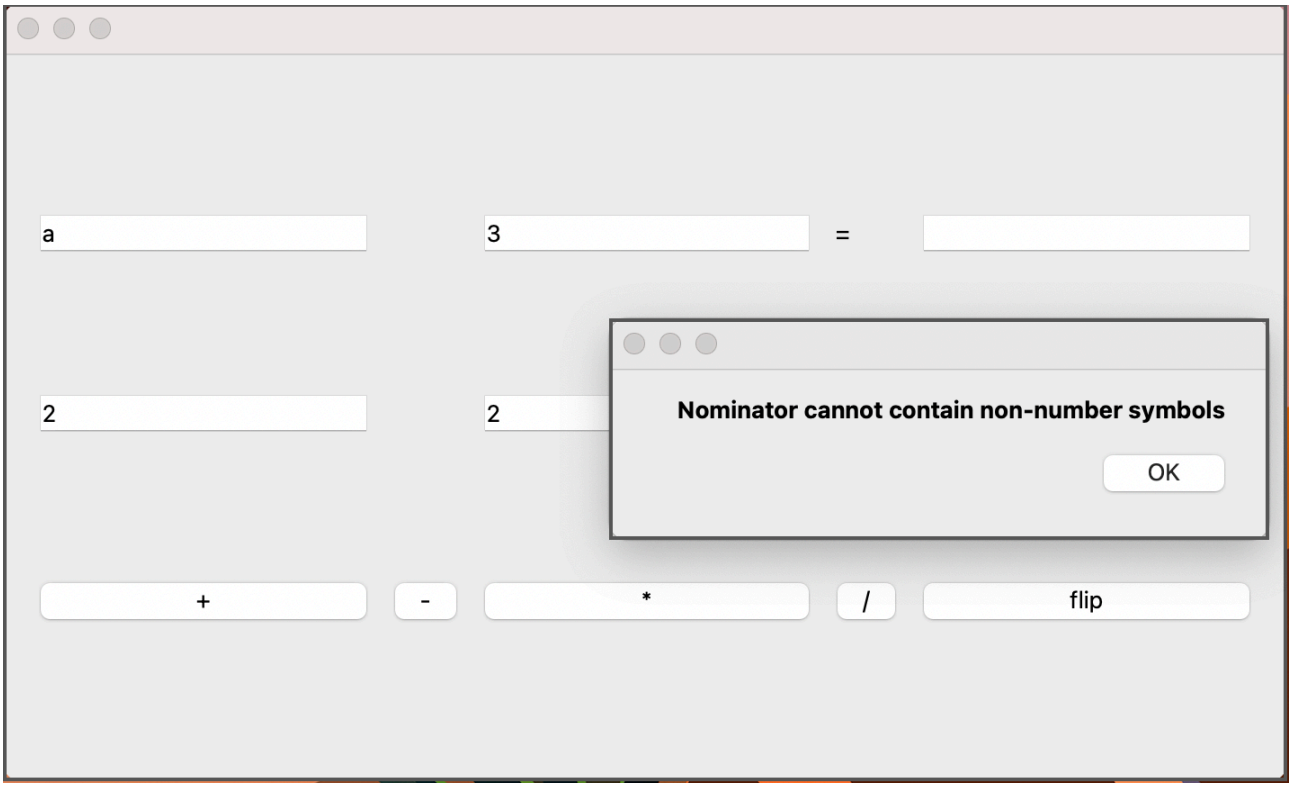

Рис. 2. Виняткова ситуація помилкового введення літерного символу

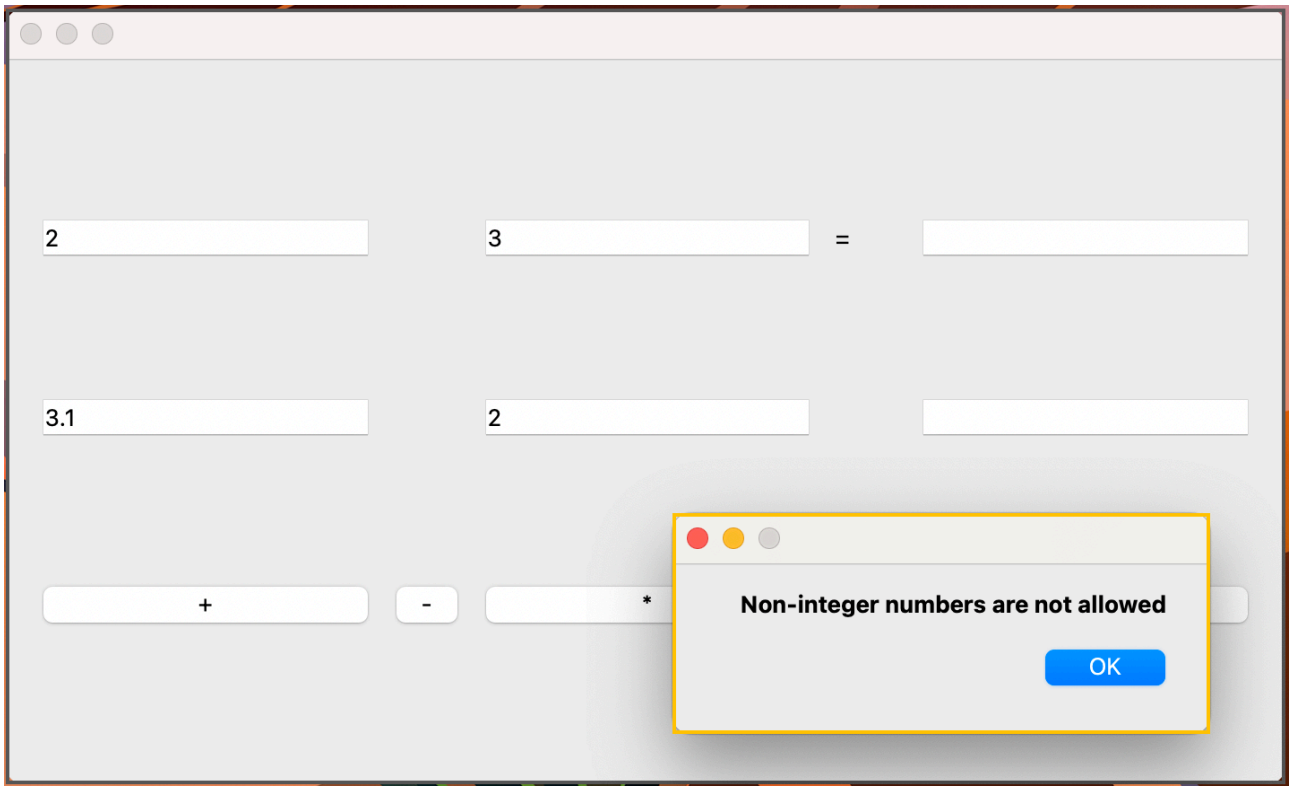

Рис. 3. Виняткова ситуація введення дробового числа

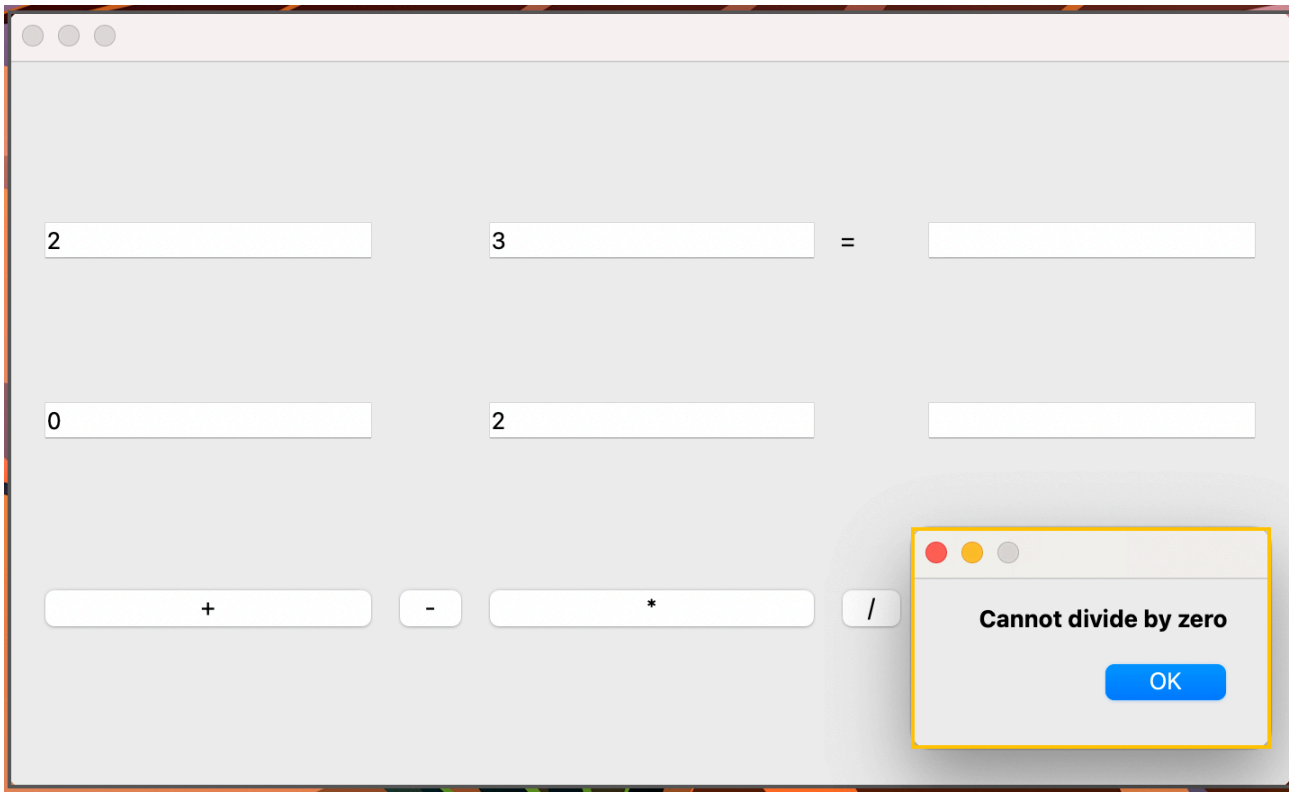

Рис. 4. Виняткова ситуація ділення на нуль

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.