Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторних робіт №12

На тему:

«Виняткові ситуації в мові програмування С++»

Лектор:
доцент каф. ПЗ
Коротєєва Т.О.
Виконав:
ст. гр. ПЗ-11
Морозов О. Р.
Прийняла:
доцент каф. ПЗ
Коротєєва Т.О.

« __ » _____ 2022 р.

∑ = _____ _____ .

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

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(тип 1 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 ігноровані. 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, викликаної всередині блоку try.

ЗАВДАННЯ ДО 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.

Завдання згідно варіанту

Варіант 2

Розробити програмне забезпечення, яке дозволяло б працювати з числами в різних форматах. Числа представляти, як об’єкти класу Chyslo. Користувач задає ціле число з клавіатури в одному з трьох форматів: двійковий, десятковий, шістнадцятковий (формат користувач теж задає при вводі). Клас повинен містити функціонал, який дозволяв би конвертувати число з одного формату в інший, а також додавати, віднімати і множити числа. Програма повинна перехоплювати та опрацьовувати такі виняткові ситуації: а) випадковий ввід користувачем символу замість цифри, б) переповнення, в) введення завеликого числа, г) ще дві виняткові ситуації передбачити самостійно.

Код програми

number.h

```
#ifndef NUMBER_H
#define NUMBER_H

#include <QRegularExpression>

#include <string>
#include <sstream>
#include <iostream>
#include <bitset>

class Number
{
     private:
```

```
        int m_data;

        QRegularExpression m_CheckBinaryData;
        QRegularExpression m_CheckDecimalData;
        QRegularExpression m_CheckHeximalData;

    public:
        Number() : m_data(),
        m_CheckBinaryData("[A-Z2-9]"),
        m_CheckDecimalData("[A-Z]") {};


        Number(int num) : m_data(num),
              m_CheckBinaryData("[A-Z2-9]"),
              m_CheckDecimalData("[A-Z]") {};

        Number(Number &n) :
              m_CheckBinaryData("[A-Z2-9]"),
              m_CheckDecimalData("[A-Z]]")
        {
              this->m_data = n.m_data;
        }


        std::string toHex();
        std::string toDec();
        int getDec();
        std::string toBin();

        void fromHex(const std::string &str);
        void fromDec(const std::string &str);
        void fromDec(int &num);
        void fromBinary(const std::string &str);

        friend Number operator+(const Number a, const Number b);
        friend Number operator-(const Number a, const Number b);
        friend Number operator*(const Number a, const Number b);
        friend Number operator/(const Number a, const Number b);

        Number & operator+=(const Number &a);
        Number & operator-=(const Number &a);
        Number & operator*=(const Number &a);
        Number & operator/=(const Number &a);

        Number & operator=(const Number &a);
        Number & operator=(int &num);
};

#endif // NUMBER_H
```

number.cpp

```
#include "number.h"
```

```
std::string Number::toHex(){
     std::stringstream hexstr;
     hexstr << std::hex << this->m_data;
     return hexstr.str();
}
std::string Number::toDec(){
     return std::to_string(m_data);
}
std::string Number::toBin(){
     return (std::bitset<8>{static_cast<unsigned long long>(m_data)}).to_string();
}

void Number::fromHex(const std::string &str){
            std::stringstream hexstr;
            hexstr << str;
            hexstr >> std::hex >> this->m_data;
}
void Number::fromDec(const std::string &str){
     if(QString::fromStdString(str).contains(m_CheckDecimalData)) {
            throw 1;
     }
     this->m_data = std::stoi(str);
}
void Number::fromDec(int &num){
     this->m_data = num;
}

unsigned long fromBin(unsigned long n)
{
     unsigned long factor = 1;
     unsigned long total = 0;

     while (n != 0)
     {
            total += (n%10) * factor;
            n /= 10;
            factor *= 2;
     }

     return total;
}

void Number::fromBinary(const std::string &str){
     if(QString::fromStdString(str).contains(m_CheckBinaryData)){
            throw 1;
     }
     unsigned long ui = std::stoi(str);
     qDebug() << ui;
     this->m_data = fromBin(ui);
     qDebug() << this->m_data;
```

```
}


Number operator+(const Number a, const Number b){
	return Number(a.m_data + b.m_data);
}
Number operator-(const Number a, const Number b){
	return Number(a.m_data - b.m_data);
}
Number operator*(const Number a, const Number b){
	return Number(a.m_data * b.m_data);
}
Number operator/(const Number a, const Number b){
	if(b.m_data == 0) {
		throw 1.1;
	}
	return Number(a.m_data / b.m_data);
}

Number & Number::operator+=(const Number &a){
	this->m_data = this->m_data + a.m_data;
	return *this;
}
Number & Number::operator-=(const Number &a){
	this->m_data = this->m_data - a.m_data;
	return *this;
}
Number & Number::operator*=(const Number &a){
	this->m_data = this->m_data * a.m_data;
	return *this;
}
Number & Number::operator/=(const Number &a){
	if(a.m_data == 0) {
		throw 1;
	}
	this->m_data = this->m_data / a.m_data;
	return *this;
}

Number & Number::operator=(const Number &a){
	this->m_data = a.m_data;
	return *this;
}
Number & Number::operator=(int &num){
	m_data = num;
	return *this;
}
```

widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
```

```
#include <QWidget>
#include <QLayout>

#include <QMessageBox>
#include <QRadioButton>
#include <QPushButton>
#include <QLabel>
#include <QString>

#include <iostream>
#include "number.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
            Q_OBJECT

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

            QPushButton * createAndConnectNumberAlphaButton(const QString
&strButtonName, const QString &strButtonStyle);
            QPushButton * createAndConnectOperatorButton(const QString &strButtonName,
const QString &strButtonStyle);

            void Convert(Number * num);

     private slots:
            void ButtonNumberAlphaClicked(); //slot for numbers and letters buttons
            void ButtonOperatorClicked(); //slot for operator buttons
            void ButtonSetChecked(); // slot for type set button

     private:
            Number *m_a = new Number();
            Number *m_b = new Number();
            Number *m_c = new Number();

            std::string m_OperatorArray[6] = {"÷","×","−","+","=","⌫"};
            int m_CurrentTask;

            QString m_CurrentCheckedButon;
            QLabel *m_UpperLabel, *m_BottomLabel;
            QPushButton *m_HexButton, *m_DecButton, *m_BinButton;
};
#endif // WIDGET_H
```

widget.cpp

```
#include "widget.h"
#include "ui_widget.h"

void Widget::ButtonNumberAlphaClicked() {
	QString str = ((QPushButton*)sender())->text();

	if(this->m_BinButton->isChecked()) {
		this->m_BottomLabel->setText(this->m_BottomLabel->text() + str);
		return;
	} else if(this->m_BottomLabel->text() == "0"){
		this->m_BottomLabel->clear();
		this->m_BottomLabel->setText(this->m_BottomLabel->text() + str);
		return;
	} else {
		this->m_BottomLabel->setText(this->m_BottomLabel->text() + str);
		return;
	}
}

void Widget::Convert(Number * num) {
	if(this->m_CurrentCheckedButon == "Bin") {
		try {
			num->fromBinary(this->m_BottomLabel->text().toStdString());
		} catch(int err){
			qDebug() << "Convertation error";
			throw;
		}
	}
	if(m_CurrentCheckedButon == "Dec") {
		try {
			num->fromDec(this->m_BottomLabel->text().toStdString());
		} catch(int err){
			qDebug() << "Convertation error";
			throw;
		}
	}
	if(m_CurrentCheckedButon == "Hex") {
		try {
			num->fromHex(this->m_BottomLabel->text().toStdString());
		} catch(int err){
			qDebug() << "Convertation error";
			throw;
		}
	}
}

void Widget::ButtonOperatorClicked() {
	std::string strButton = ((QPushButton*)sender())->text().toStdString();

	int i = 0;
```

```
	while(m_OperatorArray[i] != strButton){
		++i;
	}

	//0 "÷" //1 "×"	//2 "−"	//3 "+"	//4 "="	//5 "⌫"

	QString OperatorString[5] = {"÷", "×", "-", "+", "="};

	try{
	switch(i){
		case 0 :{
			try {
				this->Convert(this->m_a);
			}
			catch(int){
				throw;
			}
			this->m_CurrentTask = 1;
			this->m_UpperLabel->setText(this->m_BottomLabel->text() + " " +
((QPushButton*)sender())->text());
			this->m_BottomLabel->setText("0");
			break;
		}
		case 1 :{
			try {
				this->Convert(this->m_a);
			}
			catch(int){
				throw;
			}
			this->m_CurrentTask = 2;
			this->m_UpperLabel->setText(this->m_BottomLabel->text() + " " +
((QPushButton*)sender())->text());
			this->m_BottomLabel->setText("0");
			break;
		}
		case 2 :{
			try {
				this->Convert(this->m_a);
			}
			catch(int){
				throw;
			}
			this->m_CurrentTask = 3;
			this->m_UpperLabel->setText(this->m_BottomLabel->text() + " " +
((QPushButton*)sender())->text());
			this->m_BottomLabel->setText("0");
			break;
		}
		case 3 :{
			try {
```

```
                        this->Convert(this->m_a);
                }
                catch(int){
                        throw;
                }
                this->m_CurrentTask = 4;
                this->m_UpperLabel->setText(this->m_BottomLabel->text() + " " +
((QPushButton*)sender())->text());
                this->m_BottomLabel->setText("0");
                break;
            }
            case 4 :{
                if(!m_CurrentTask) {
                        throw "Number not entered";
                }

                try {
                        this->Convert(this->m_b);
                }
                catch(int){
                        throw;
                }

                switch(this->m_CurrentTask) {
                        case 1: {
                                *m_c = *m_a / *m_b;
                                break;
                        }
                        case 2: {
                                *m_c = *m_a * *m_b;
                                break;
                        }
                        case 3: {
                                *m_c = *m_a - *m_b;
                                break;
                        }
                        case 4: {
                                *m_c = *m_a + *m_b;
                                break;
                        }
                }

                this->m_UpperLabel->setText(this->m_UpperLabel->text() + " " +
this->m_BottomLabel->text());

                if(this->m_CurrentCheckedButon == "Bin") {

this->m_BottomLabel->setText(QString::fromStdString(m_c->toBin()));
                }
                if(m_CurrentCheckedButon == "Dec") {
```

```
this->m_BottomLabel->setText(QString::fromStdString(m_c->toDec()));
                }
                if(m_CurrentCheckedButon == "Hex") {

this->m_BottomLabel->setText(QString::fromStdString(m_c->toHex()));
                }

                break;
            }
            case 5 :{
                if(this->m_BottomLabel->text() != "0") {

this->m_BottomLabel->setText(this->m_BottomLabel->text().chopped(1));
                        if(this->m_BottomLabel->text() == "") {
                            this->m_BottomLabel->setText("0");
                        }
                }
                break;
            }
      }
      }
      catch(double dnum) {
            QMessageBox * msg = new QMessageBox(this);
            msg->setText("You cannot divide by 0");
            msg->show();
      }
      catch(const char *str) {
            QMessageBox * msg = new QMessageBox(this);
            msg->setText("Numbers not entered!");
            msg->show();
      }
      catch(int num) {
            QMessageBox * msg = new QMessageBox(this);
            msg->setText("Convertation error\nChoose another type");
            msg->show();
      }
}

void Widget::ButtonSetChecked(){
      QString strButton = ((QPushButton*)sender())->text();
      if(strButton == "Hex") {
            this->m_BinButton->setChecked(false);
            this->m_DecButton->setChecked(false);
            this->m_CurrentCheckedButon = "Hex";
      } else if(strButton == "Dec") {
            this->m_BinButton->setChecked(false);
            this->m_HexButton->setChecked(false);
            this->m_CurrentCheckedButon = "Dec";
      } else if(strButton == "Bin") {
            this->m_HexButton->setChecked(false);
```

```
            this->m_DecButton->setChecked(false);
            this->m_CurrentCheckedButon = "Bin";
      }
}

QPushButton * Widget::createAndConnectNumberAlphaButton(const QString &strButtonName,
const QString &strButtonStyle) {
      QPushButton * button = new QPushButton(strButtonName);

      button->setStyleSheet(strButtonStyle);
      button->setShortcut(QKeySequence(strButtonName));

      connect(button, SIGNAL(clicked()), SLOT(ButtonNumberAlphaClicked()));

      return button;
}

QPushButton * Widget::createAndConnectOperatorButton(const QString &strButtonName,
const QString &strButtonStyle) {
      QPushButton * button = new QPushButton(strButtonName);

      button->setStyleSheet(strButtonStyle);
      if(strButtonName.contains("⌫")) {
            button->setShortcut(QKeySequence(Qt::Key_Backspace));
      }

      connect(button, SIGNAL(clicked()), SLOT(ButtonOperatorClicked()));

      return button;
}

Widget::Widget(QWidget *parent)
      : QWidget(parent) {
      this->setFixedSize(250,400);
      this->setWindowIcon(QIcon("://icon//keys.png"));
      this->setWindowTitle("Calculator");
      this->setStyleSheet(
                        "QWideget { background-color: #ebedff;}"
                        "QPushButton {max-height: 50px; max-width: 50px;}"
                                    );


      QString PushButtonsName[5][5] = {{"A","Hex","Dec","Bin","÷"},
                                                      {"B","7","8","9","×"},
                                                      {"C","4","5","6","−"},
                                                      {"D","1","2","3","+"},
                                                      {"E","F","0","⌫","="}
                                                     };
      this->m_UpperLabel = new QLabel;
      this->m_BottomLabel = new QLabel("0");
```

```
    m_UpperLabel->setStyleSheet("QLabel {background-color: #fff; font-family: Source
Code Pro; font: bold 25px; }");
    m_BottomLabel->setStyleSheet("QLabel {background-color: #fff; font-family: Source
Code Pro; font: bold 30px; }");
    m_UpperLabel->setAlignment(Qt::AlignRight);
    m_BottomLabel->setAlignment(Qt::AlignRight);
    m_BottomLabel->setCursor(QCursor(Qt::IBeamCursor));

    QString NumAlphaButtonStyle = " QPushButton {"
                                                    "background-color: #fff;"
                                                    "border-radius: 15px;"
                                                    "color: #000;"
                                                    "font-family: Source Code
Pro;"
                                                    "font: bold 17px;"
                                                    "outline: none;"
                                                    "padding: .75rem;"
                                                    "border-bottom-left-radius:
15px 255px;"
                                                    "border-bottom-right-radius:
255px 15px;"
                                                    "border-top-left-radius:
255px 15px;"
                                                    "border-top-right-radius:
15px 255px;}"
                                                "QPushButton::pressed {"
                                                "border: 2px solid #0087f3;}";
    QString OperatorButtonStyle = " QPushButton {              "
                                                    "background-color: #fff;"
                                                    "border-radius: 15px;"
                                                    "color: #0087f3;"
                                                    "font-family: Source Code
Pro;"
                                                    "font: bold 20px;"
                                                    "outline: none;"
                                                    "padding: .75rem;"
                                                    "border-bottom-left-radius:
15px 255px;"
                                                    "border-bottom-right-radius:
255px 15px;"
                                                    "border-top-left-radius:
255px 15px;"
                                                    "border-top-right-radius:
15px 255px;}"
                                                "QPushButton::pressed {"
                                                "border: 2px solid #0087f3;}"
                                                "QPushButton::checked {"
                                                "background-color: #ebedff;}";

    QGridLayout * mainLayout = new QGridLayout;
```

```
	for(int i = 0; i < 5; i++) {
		QPushButton *pButton =
createAndConnectNumberAlphaButton(PushButtonsName[i][0], NumAlphaButtonStyle);
		mainLayout->addWidget(pButton, i+2, 0);
		pButton = createAndConnectOperatorButton(PushButtonsName[i][4],
OperatorButtonStyle);
		mainLayout->addWidget(pButton, i+2, 4);
	}

	this->m_HexButton = new QPushButton("Hex");
	this->m_HexButton->setStyleSheet(OperatorButtonStyle);
	this->m_HexButton->setCheckable(true);
	connect(this->m_HexButton, SIGNAL(clicked()), SLOT(ButtonSetChecked()));
	mainLayout->addWidget(m_HexButton, 2, 1);

	this->m_DecButton = new QPushButton("Dec");
	this->m_DecButton->setStyleSheet(OperatorButtonStyle);
	this->m_DecButton->setCheckable(true);
	this->m_DecButton->setChecked(true);
	connect(this->m_DecButton, SIGNAL(clicked()), SLOT(ButtonSetChecked()));
	mainLayout->addWidget(m_DecButton, 2, 2);

	this->m_BinButton = new QPushButton("Bin");
	this->m_BinButton->setStyleSheet(OperatorButtonStyle);
	this->m_BinButton->setCheckable(true);
	connect(this->m_BinButton, SIGNAL(clicked()), SLOT(ButtonSetChecked()));
	mainLayout->addWidget(m_BinButton, 2, 3);

	this->m_CurrentCheckedButon = "Dec";
	this->m_CurrentTask = 0;

	for(int i = 1; i < 5; i++) {
		for(int j = 1; j < 4; j++) {
			if(PushButtonsName[i][j] != "⌫") {
			QPushButton *pButton =
createAndConnectNumberAlphaButton(PushButtonsName[i][j], NumAlphaButtonStyle);
			mainLayout->addWidget(pButton, i+2, j);
			} else {
				QPushButton *pButton =
createAndConnectOperatorButton(PushButtonsName[i][j], OperatorButtonStyle);
				mainLayout->addWidget(pButton, i+2, j);
			}
		}
	}


	mainLayout->setSpacing(0);
	mainLayout->setContentsMargins(0,0,0,0);

	mainLayout->addWidget(this->m_UpperLabel, 0, 0, 1, 5);
	mainLayout->addWidget(this->m_BottomLabel, 1, 0, 1, 5);
```

```
	setLayout(mainLayout);
}

Widget::~Widget() {
	delete m_a;
	delete m_b;
	delete m_c;

	delete m_UpperLabel;
	delete m_BottomLabel;

	delete m_HexButton;
	delete m_DecButton;
	delete m_BinButton;
}
```

main.cpp

```
/*
* icon taked from:
* https://www.flaticon.com/
*/

#include "widget.h"

#include <QApplication>

int main(int argc, char *argv[])
{
	QApplication a(argc, argv);
	Widget w;

	try {
	w.setWindowIcon(QIcon(":/math.png"));
	w.setWindowTitle("Calculator");
	w.show();
	}
	catch(...){
		qDebug() << "Undefined error";
	}

	return a.exec();
}
```

Вигляд програми

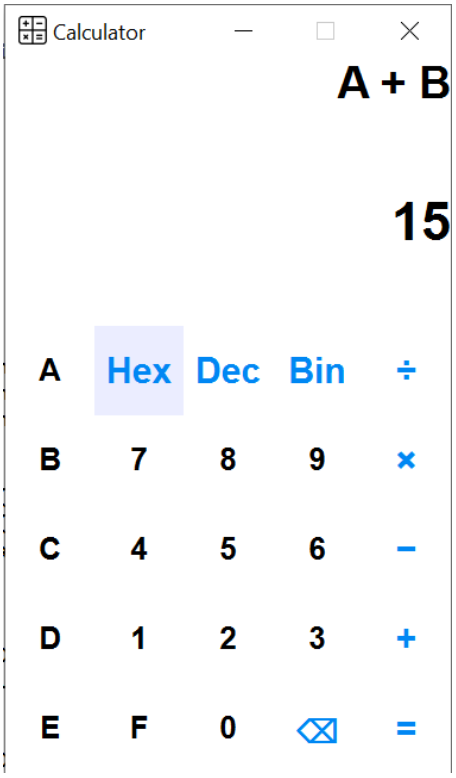

Рис 1. Загальний вигляд програми

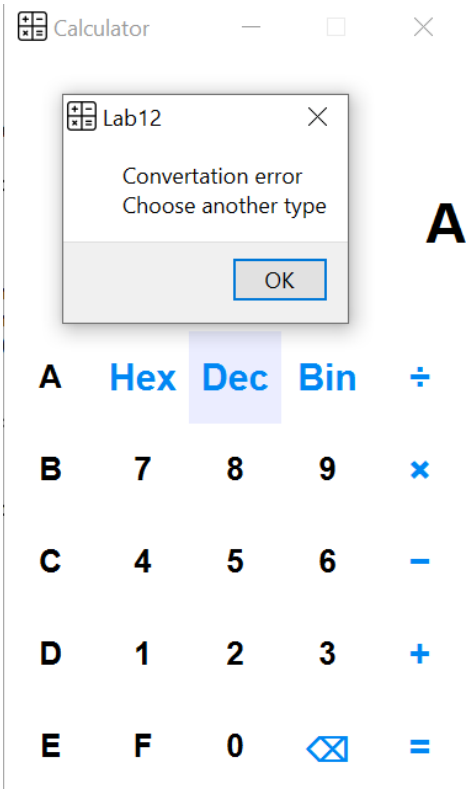

Рис 2. Помилка конвертації

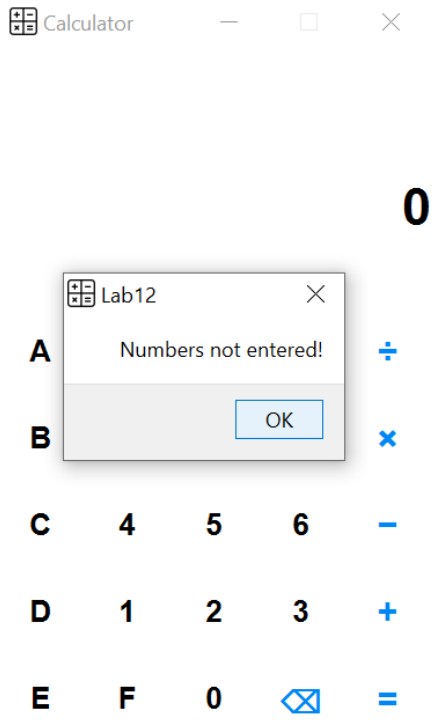

Рис 3. Помилка числа не введені

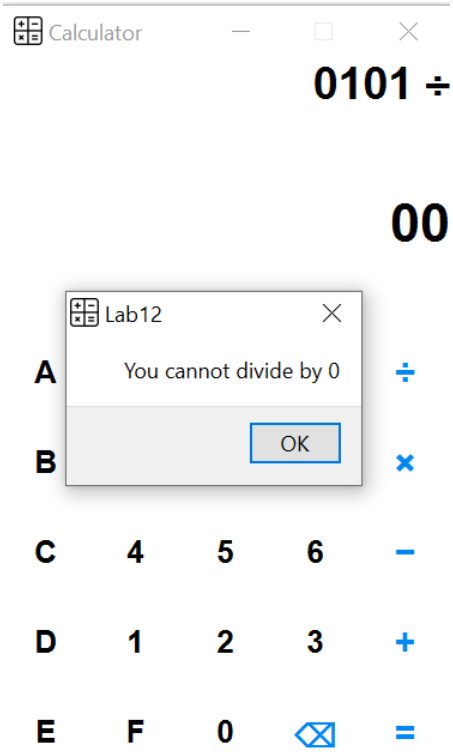

Рис 4. Помилка при спробі поділити на 0

ВИСНОВКИ

Виконуючи лабораторну роботу №12, я навчився користуватися винятковими ситуаціями в мові програмування C++, 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.