МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

До лабораторної роботи № 12
На тему: «Виняткові ситуації в мові програмування С++»
З дисципліни: «Об'єктно-орієнтоване програмування»

Лектор:
доц. кафедри ПЗ
Коротєєва Т. О.

Виконав:
ст. гр. ПЗ-16
Чаус О. М.

Прийняв:
асист. кафедри ПЗ
Дивак І. В.

« ____ » _____ 2022 р.

∑= _____ .....:

Львів-2022

Тема роботи: Виняткові ситуації в мові програмування C++

Мета роботи: 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 C++ винятки – це об’єкт, який передається з тої області коду, де виникла проблема, в ту область коду, де проблема обробляється. Тип винятка визначає, яка область коду буде обробляти проблему, а вміст об’єкту може використовуватись для зворотного зв’язку з користувачем.

Основні етапи застосування винятків

1. Знайти ті ділянки програми, в яких знаходяться операції, що можуть привести до виникнення винятків, та помістити їх в блоки try.

2. У відповідному місці коду згенерувати можливу помилку з допомогою ключового слова throw.

3. Для обробки винятків створити блоки catch, які виправляють ситуацію та видають повідомлення користувачу.

Блок try – це набір операторів у фігурних дужках, всередині яких може бути перехоплено виняток. Блок catch – це набір блоків операторів, що знаходиться безпосередньо за блоком try, в яких відбувається обробка винятка.

Розмір блоку try не обмежений. У нього можна розмістити 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 підходящих обробників catch немає, то виконання програми переходить до наступного блоку try. Якщо до кінця програми не знайдені відповідні обробники catch, то програма завершує своє виконання з помилкою винятку.

Оператор throw генерує виняткову ситуацію. Для цього оператор throw повинний знаходитися усередині блоку try або усередині функції, що викликається усередині блоку try. За ключовим словом throw вказується значення будь-якого типу даних, яке ви хочете задіяти, щоб сигналізувати про помилку. Як правило, цим значенням є код помилки, опис проблеми або спеціальний клас-виняток.

Клас винятків

Виняткова ситуація — це об'єкт, що може мати будь-який тип, як вбудований, так і користувацький. Клас-виняток — це звичайний клас, який викидається в якості винятку. Створення класів користувацьких ситуацій дозволяє програмісту точніше визначати реакцію програми на небажану ситуацію.

Винятки і успадкування

Оскільки виняткова ситуація може бути об'єктом класу, у мові C++ існує можливість створювати ієрархію виняткових ситуацій. У цьому випадку блок catch перехоплює об'єкти не тільки базового, але і похідних класів. Щоб задіяти виконання обробки помилки похідного типу, необхідно розмістити блок catch, що відповідає похідному

класу, вище блоку, призначеного для перехоплення об'єктів базового класу. Інакше блок catch похідного типу ніколи не спрацює. Його завжди буде перехоплювати блок catch базового типу.

Обробка винятків

Виняток, що переданий, попадає в стек викликів. Стек викликів – це список функцій, що викликаються. В ньому реєструються звернення до функцій. Переданий в стек виняток проходить всі його блоки, поки не знайде свій блок, який обробить даний виняток. Це називається «**прокруткою стеку**» або **розкручуванням стеку**. За кожним блоком **try** знаходиться один або декілька блоків **catch.** Якщо переданий виняток відповідає одному з блоків **catch**, то він обробляється операторами цього блоку. Якщо відповідного блоку **catch** не знайдено, то викликається вбудований обробник, який завершить програму.

Прокрутка стеку винятком це процес незворотній. По мірі прокрутки стеку об’єкти руйнуються. Повернутись неможливо. Програма поновить роботу після того оператора **catch,** який обробив виняток, що був переданий блоком **try.**

Можлива ситуація, коли причин виникнення винятку декілька. В цьому випадку оператори **catch** можна розмістити один за одним як в операторі switch. Еквівалентом оператора default буде оператор **catch(...) ,** що означає «обробити все**».**

Специфікації винятків

Специфікації винятків — це механізм оголошення функцій із зазначенням того, чи генеруватиме функція винятки (і які саме) чи ні. Це може бути корисно при визначенні необхідності розміщення виклику функції в блоці try.

Існують три типи специфікації винятків.

По-перше, ми можемо використовувати порожній оператор throw для позначення того, що функція не генерує ніяких винятків, які виходять за її межі
По-друге, ми можемо використовувати оператор throw із зазначенням типу винятку, який може генерувати ця функція
Третій варіант функція може генерувати різні типи винятків
Оголошуючи **функцію**, можна перелічити типи виняткових ситуацій, що їй дозволено генерувати. Типи виняткових ситуацій, не включені в список, функція генерувати не зможе. Спроба порушити це обмеження приведе до негайного припинення роботи програми за допомогою ланцюжка викликів стандартних функцій: unexpected() — abort().

Повторна генерація винятків

Іноді ви можете зіткнутися з ситуацією, коли потрібно зловити виняток, але обробляти його в даний момент часу не потрібно (або немає можливості). Наприклад, ви можете записати помилку в лог-файл, а потім передати її назад в caller для виконання фактичної обробки.

Мова C++ надає спосіб повторної генерації одного і того ж винятку. Для цього потрібно використати ключове слово throw всередині блоку catch без вказівки будь-якого ідентифікатора.

Інтерфейсний клас std::exception

exception є інтерфейсний клас, який використовується в якості батьківського класу для будь-якого винятку, який генерується в Стандартній бібліотеці C++.

Завдяки std::exception ми можемо налаштувати обробник винятків типу std::exception, який ловитиме і оброблятиме як std::exception, так і всі дочірні йому класи-винятки. Прикладами дочірніх є такі класи-винятки : 1) Невдале динамічне приведення типів за допомогою оператора dynamic_cast генерує виняток std::bad_cast; 2) оператор new та std::string можуть генерувати std::bad_alloc при нестачі пам'яті.

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.

Хід роботи

Файл **MyException.h :**

```
#ifndef MYEXCEPTION_H
#define MYEXCEPTION_H

#include <string>

class MyException {
protected:
    std::string m_message;
public:
    MyException(std::string mess) : m_message(mess) {};
    std::string what() const{
        return m_message;
    };
};

class DivisionByZeroException : public MyException{
public:
    DivisionByZeroException() : MyException("Error: division by 0 is impossible") {};
};

class NotANumberException : public MyException {
public:
    NotANumberException() : MyException("Error: input string is not a number") {};
};

class OverflowException : public MyException {
public:
    OverflowException() : MyException("Error: overflow") {};
};

class OutOfRangeException : public MyException {
public:
    OutOfRangeException() : MyException("Error: one of the indices is out of range") {};
};

class EmptyTextException : public MyException {
public:
    EmptyTextException() : MyException("Error: some lineedits are empty") {};
};

class WrongXException : public MyException {
```

```
public:
    WrongXException() : MyException("Error: this value of x can't be input") {};
};

class WrongMatrixException : public MyException {
public:
    WrongMatrixException() : MyException("Error: this matrix is of wrong size") {};
};

#endif // MYEXCEPTION_H
```

Файл **Vyraz.h**

```
#ifndef VYRAZ_H
#define VYRAZ_H

#include <array>
#include <vector>
#include <QTableWidget>
#include "MyException.h"
class Vyraz {
private:
    std::array<std::array<int, 10>, 3> matrix;
public:
    Vyraz();
    Vyraz(QTableWidget * table);
    double calculate(int n, int m, int k, int x);
};


#endif // VYRAZ_H
```

Файл **Vyraz.cpp**

```
#include "Vyraz.h"
#include "MyException.h"
#include <cmath>

Vyraz::Vyraz() {};

Vyraz::Vyraz(QTableWidget * table) {
    if(matrix.size() < table->rowCount() || matrix[0].size() < table->columnCount())
        throw WrongMatrixException();
    for(int i = 0; i < table->rowCount(); i++)
        for(int j = 0; j < table->columnCount(); j++)
            matrix[i][j] = table->item(i, j)->text().toInt();
};

double Vyraz::calculate(int n, int m, int k, int x) {
    double result = 0;
    double currentResult = 0;
    if(x == 10)
        throw(DivisionByZeroException());
    if(x > 100 || x < 0)
        throw WrongXException();
    if(n > matrix[0].size())
        throw OutOfRangeException();
    else if(m > matrix[1].size())
        throw OutOfRangeException();
    else if(k > matrix[2].size())
        throw OutOfRangeException();
    for(std::size_t i = 0; i < n; i++)
        currentResult += matrix[0][i] * sqrt(x);
    result += currentResult;
    currentResult = 0;
    for(std::size_t i = 0; i < m; i++)
        currentResult += matrix[1][i] * log(100 - x);
    result += currentResult;
    currentResult = 0;
    for(std::size_t i = 0; i < k; i++)
```

```
        currentResult += matrix[2][i] / (x - 10);
    result += currentResult;
    return result;
}
```

Файл **mainwindow.cpp**

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include "MyException.h"
#include <QMessageBox>
#include <climits>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    srand(time(NULL));
    ui->tableWidget->blockSignals(true);
    for(int i = 0; i < ui->tableWidget->rowCount(); i++)
        for(int j = 0; j < ui->tableWidget->columnCount(); j++)
        {
            ui->tableWidget->setItem(i, j, new QTableWidgetItem(QString::number(rand() %
200 - 100)));
        }
    ui->tableWidget->blockSignals(false);
}

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


void MainWindow::on_tableWidget_cellChanged(int row, int column)
{
    ui->tableWidget->blockSignals(true);
    try{
    for(std::size_t i = 0; i < ui->tableWidget->item(row, column)->text().size(); i++)
    {
        if(!isdigit(ui->tableWidget->item(row, column)->text().at(i).toLatin1()))
        {
            if(i != 0 || ui->tableWidget->item(row, column)->text().at(i).toLatin1() !=
'-')
            {
                ui->tableWidget->setItem(row, column, new QTableWidgetItem("0"));
                throw NotANumberException();
            }
        }
    }
    if(ui->tableWidget->item(row, column)->text().toDouble() > INT_MAX || ui-
>tableWidget->item(row, column)->text().toDouble() < INT_MIN)
    {
        ui->tableWidget->setItem(row, column, new QTableWidgetItem("0"));
        throw OverflowException();
    }
    } catch(const MyException& e)
    {
        QMessageBox::critical(this, "Error", QString::fromStdString(e.what()));
    }
    ui->tableWidget->blockSignals(false);
}


void MainWindow::on_NlineEdit_textEdited(const QString &arg1)
{
        try {
        for(std::size_t i = 0; i < ui->NlineEdit->text().size(); i++)
```

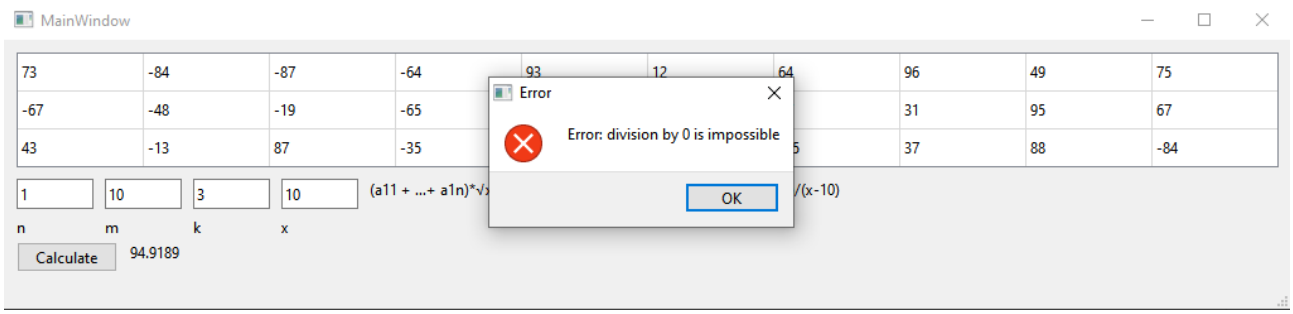
```
        {
            if(!isdigit(ui->NlineEdit->text().at(i).toLatin1()))
            {
                ui->NlineEdit->setText("");
                throw NotANumberException();
            }
            if (ui->NlineEdit->text().toDouble() > INT_MAX || ui->NlineEdit-
>text().toDouble() < INT_MIN)
            {
                ui->NlineEdit->setText("");
                throw OverflowException();
            }
        }
        } catch (const MyException &e)
        {
            QMessageBox::critical(this, "Error", QString::fromStdString(e.what()));
        }
}

void MainWindow::on_MlineEdit_textEdited(const QString &arg1)
{
    try {
    for(std::size_t i = 0; i < ui->MlineEdit->text().size(); i++)
    {
        if(!isdigit(ui->MlineEdit->text().at(i).toLatin1()))
        {
            ui->MlineEdit->setText("");
            throw NotANumberException();
        }
        if (ui->MlineEdit->text().toDouble() > INT_MAX || ui->MlineEdit-
>text().toDouble() < INT_MIN)
        {
            ui->MlineEdit->setText("");
            throw OverflowException();
        }
    }
    } catch (const MyException &e)
    {
        QMessageBox::critical(this, "Error", QString::fromStdString(e.what()));
    }
}


void MainWindow::on_KlineEdit_textEdited(const QString &arg1)
{
    try {
    for(std::size_t i = 0; i < ui->KlineEdit->text().size(); i++)
    {
        if(!isdigit(ui->KlineEdit->text().at(i).toLatin1()))
        {
            ui->KlineEdit->setText("");
            throw NotANumberException();
        }
        if (ui->KlineEdit->text().toDouble() > INT_MAX || ui->KlineEdit-
>text().toDouble() < INT_MIN)
        {
            ui->KlineEdit->setText("");
            throw OverflowException();
        }
    }
    } catch (const MyException &e)
    {
        QMessageBox::critical(this, "Error", QString::fromStdString(e.what()));
    }
}
```

```
void MainWindow::on_XlineEdit_textEdited(const QString &arg1)
{
    try {
    for(std::size_t i = 0; i < ui->XlineEdit->text().size(); i++)
    {
        if(!isdigit(ui->XlineEdit->text().at(i).toLatin1()))
        {
            if(i != 0 || ui->XlineEdit->text().at(i) != '-')
            {
                ui->XlineEdit->setText("");
                throw NotANumberException();
            }
        }
        if (ui->XlineEdit->text().toDouble() > INT_MAX || ui->XlineEdit-
>text().toDouble() < INT_MIN)
        {
            ui->XlineEdit->setText("");
            throw OverflowException();
        }
    }
    } catch (const MyException &e)
    {
        QMessageBox::critical(this, "Error", QString::fromStdString(e.what()));
    }
}


void MainWindow::on_pushButton_clicked()
{
    try {
    if(ui->MlineEdit->text().isEmpty() || ui->NlineEdit->text().isEmpty()
            || ui->KlineEdit->text().isEmpty() || ui->XlineEdit->text().isEmpty())
        throw EmptyTextException();
    Vyraz equation(ui->tableWidget);
    ui->labelRes->setText(QString::number(equation.calculate(ui->NlineEdit-
>text().toInt(), ui->MlineEdit->text().toInt(),
                                                              ui->KlineEdit-
>text().toInt(), ui->XlineEdit->text().toInt())));
    } catch(const MyException &e) {
        QMessageBox::critical(this, "Error", QString::fromStdString(e.what()));
    }
}
```



Зображення програми, приклад виведеної помилки

Висновок: 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.