**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **ІКНІ**

Кафедра **ПЗ**



### ЗВІТ

До лабораторної роботи № 12

**На тему:** *“* *Виняткові ситуації в мові програмування С++”*

**З дисципліни:** *“Об’єктно-орієнтоване програмування”*

**Лектор:**

доцент каф.ПЗ

Коротєєва Т. О.

**Виконав:**

ст. гр. ПЗ-16

Шеремета А.І.

**Прийняв:**

асист. каф. ПЗ

Дивак І.В.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑= \_\_\_\_ .

Львів – 2022

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

**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**

У програмах на C++ можуть виникати помилки. Розрізняють три типи помилок, які можуть виникати у програмах:

синтаксичні. Це помилки в синтаксисі мови C++. Вони можуть зустрічатись в іменах операторів, функцій, розділювачів і т.д. У цьому випадку компілятор визначає наявність синтаксичної помилки і видає відповідне повідомлення. У результаті виконавчий (\*.exe) файл не створюється і програма не виконується;

логічні. Це помилки програміста, які важко виявити на етапі розробки програми. Ці помилки виявляються на етапі виконання під час тестування роботи програми. Логічні помилки можна виявити тільки за результатами роботи програми. Прикладом логічних помилок може бути неправильна робота з покажчиками у випадках виділення/звільнення пам’яті;

помилки часу виконання. Такі помилки виникають під час роботи програми. Помилки часу виконання можуть бути логічними помилками програміста, помилками зовнішніх подій (наприклад, нехватка оперативної пам’яті), невірним введенням даних користувачем тощо. У результаті виникнення помилки часу виконання, програма призупиняє свою роботу. Тому, важливо перехопити цю помилку і правильно обробити її для того, щоб програма продовжила свою роботу без зупинки.

Виключна ситуація – це подія, що призвела до збою в роботі програми. У результаті виникнення виключної ситуації програма не може коректно продовжити своє виконання.

Приклади дій у програмі, що можуть призвести до виникнення виключних ситуацій:

ділення на нуль;

нехватка оперативної пам’яті при застосуванні оператора new для її виділення (або іншої функції);

доступ до елементу масиву за його межами (помилковий індекс);

переповнення значення для деякого типу;

взяття кореня з від’ємного числа;

інші ситуації.

Поняття виключення

У мові C++ виключення – це спеціальний об’єкт класу або значення базового типу, що описує (визначає) конкретну виключну ситуацію і відповідним чином обробляється.

При написанні програми система опису виключних ситуацій вибирається програмістом на власний розсуд. Можна створити свою кваліфікацію помилок, які можуть виникати у програмі. Наприклад, програміст може кваліфікувати різні типи помилок числовим (цілочисельним) значенням або побудувати власну ієрархію класів що описують виключні ситуації. Крім того, можна використовувати можливості класів C++, які є похідними від класу exception.

Мова програмування C++ дає можливість перехоплювати виключні ситуації та відповідним чином їх обробляти.

Механізм перехоплення виключень C++ дозволяє генерувати виключення у тому місці, в якому воно виникає – це є дуже зручно. Не потрібно “видумувати” власні способи обробки виключень, які виникли у функціях нижчого рівня, для того щоб передати їх у функції вищого рівня.

Для перехоплення та обробки виключних ситуацій у мові C++ введено конструкцію try…catch, яка має наступну загальну форму:

try {

// тіло блоку try

// ...

// генерування виключення оператором throw

}

catch(type1 argument1)

{

// тіло блоку catch

}

catch(type2 argument2)

{

// тіло блоку catch

}

...

catch(typeN argumentN)

{

// тіло блоку catch

}

Код, який потрібно проконтролювати, повинен виконуватись всередниі блоку try. Виключні ситуації перехоплюються оператором catch, який слідує безпосередньо за блоком try в якому вони виникли.

У блоці try можуть бути розміщені оператори і функції. Якщо у блоці try генерується відповідна виключна ситуація, то вона перехоплюється відповідним блоком catch. Вибір того чи іншого блоку catch здійснюється в залежності від типу виключної ситуації. Після виникнення виключної ситуації певного типу, викликається блок catch з таким самим типом аргументу. Аргумент приймає деяке значення, яке відповідним чином обробляється (виводиться на екран повідомлення про помилку тощо).

Якщо у блоці try виникне виключна ситуація, яка не передбачена блоком catch, то викликається стандартна функція terminate(), яка за замовчуванням викликає функцію abort(). Ця стандартна функція припиняє виконання програми.

**Лабораторне завдання**

1. Ознайомитися з основними поняттями та синтаксисом мови С++, 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 5**

Реалізувати програму для обчислення виразу

(a11 + …+ a1n)\*√x + (a21 +…+a2m)\*log(100-x) + (a31 + …+a3k)/(x-10)

Коефіцієнти зчитуються з клавіатури. Роботу з виразом потрібно здійснювати за допомогою класу Vyraz. Програма повинна перехоплювати та опрацьовувати такі виняткові ситуації: а) ділення на нуль, б) помилкове введення користувачем літерного символу замість числа при введенні коефіцієнтів, в) переповнення, г)вихід за межі масиву коефіцієнтів, д) ще дві виняткові ситуації передбачити самостійно.

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

**Хід роботи**

1. Реалізовую програму:

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

//#include <exception>

#include <QMessageBox>

#include <vyraz.h>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->NVec->hide();

ui->MVec->hide();

ui->KVec->hide();

}

MainWindow::~MainWindow()

{

delete ui;

}

Vyraz MyVyraz;

void MainWindow::on\_pushButton\_clicked()

{

try{

ui->label\_2->setText(QString::number(MyVyraz.CalculateResult()));

}catch(QString& E){

QMessageBox Error;

Error.setText(E);

Error.exec();

}

}

void MainWindow::on\_setSizeN\_clicked()

{

ui->NVec->show();

ui->NVec->setColumnCount(ui->textEditN->toPlainText().toInt());

MyVyraz.setN(ui->textEditN->toPlainText().toInt());

}

void MainWindow::on\_setSizeM\_clicked()

{

ui->MVec->show();

ui->MVec->setColumnCount(ui->textEditM->toPlainText().toInt());

MyVyraz.setM(ui->textEditM->toPlainText().toInt());

}

void MainWindow::on\_setSizeK\_clicked()

{

ui->KVec->show();

ui->KVec->setColumnCount(ui->textEditK->toPlainText().toInt());

MyVyraz.setK(ui->textEditK->toPlainText().toInt());

}

void MainWindow::on\_setNVector\_clicked()

{

try{

std::vector <double>Nvector;

for(int i = 0; i < ui->NVec->columnCount(); i++){

QString checkN = ui->KVec->item(0, i)->text();

for(int j = 0; j < checkN.size(); j++){

if(checkN[j].isLetter())

throw(QString("Not a number in N col: ") + QString::number(i));

}

Nvector.push\_back(ui->NVec->item(0, i)->text().toDouble());

}

MyVyraz.setNVector(Nvector);

}catch(QString& E){

QMessageBox Error;

Error.setText(E);

Error.exec();

}

}

void MainWindow::on\_setMVector\_clicked()

{

try{

std::vector <double>Mvector;

for(int i = 0; i < ui->MVec->columnCount(); i++){

QString checkM = ui->KVec->item(0, i)->text();

for(int j = 0; j < checkM.size(); j++){

if(checkM[j].isLetter())

throw(QString("Not a number in M, col: ") + QString::number(i));

}

Mvector.push\_back(ui->MVec->item(0, i)->text().toDouble());

}

MyVyraz.setMVector(Mvector);

}catch(QString& E){

QMessageBox Error;

Error.setText(E);

Error.exec();

}

}

void MainWindow::on\_setKVector\_clicked()

{

try{

std::vector <double>Kvector;

for(int i = 0; i < ui->KVec->columnCount(); i++){

QString checkK = ui->KVec->item(0, i)->text();

for(int j = 0; j < checkK.size(); j++){

if(checkK[j].isLetter())

throw(QString("Not a number in K, col: ") + QString::number(i));

}

Kvector.push\_back(ui->KVec->item(0, i)->text().toDouble());

}

MyVyraz.setKVector(Kvector);

}catch(QString& E){

QMessageBox Error;

Error.setText(E);

Error.exec();

}

}

void MainWindow::on\_setX\_clicked()

{

try{

QString check = ui->setX->text();

for(int i = 0; i < check.size(); i++){

if(check[i].isLetter())

throw(QString("X should be a number"));

}

MyVyraz.setX(ui->setX->text().toDouble());

}catch(QString& E){

QMessageBox Error;

Error.setText(E);

Error.exec();

}

}

**Vyraz.h**

#ifndef VYRAZ\_H

#define VYRAZ\_H

#include <math.h>

#include <QMessageBox>

#include <QMainWindow>

#include <QTableWidget>

#include <vector>

#include <string>

class Exception{

public:

Exception(const QString& msg):msg(msg){}

QString msg;

};

class EqualZero : public Exception{

public:

EqualZero(const QString& msg):Exception(msg){}

};

class Vyraz

{

public:

double CalculateResult();

void setN(int n);

void setM(int m);

void setK(int k);

void setX(double x);

void setNVector(std::vector<double> nVector);

void setMVector(std::vector<double> mVector);

void setKVector(std::vector<double> kVector);

Vyraz();

private:

double X;

int n;

int m;

int k;

std::vector<double> nVector;

std::vector<double> mVector;

std::vector<double> kVector;

};

#endif // VYRAZ\_H

**Vyraz.cpp**

#include "vyraz.h"

Vyraz::Vyraz()

{

}

double Vyraz::CalculateResult(){

if(X < 0){

throw(QString("You cant find root of negative number."));

}

if(100 - X <= 0){

throw(QString("You cant find natural logarithm of 100-" + QString::number(X)));

}

if(X-10 == 0){

throw(QString("Cant divide by zero"));

}

double overflowTest = 0.0;

double nVectorsum = 0.0, mVectorsum = 0.0, kVectorsum = 0.0;

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

if(i >= nVector.size()) throw(QString("Sum count went beyond in N vector"));

nVectorsum += \*(nVector.begin() + i);

overflowTest += \*(nVector.begin() + i);

}

overflowTest \*= sqrt(X);

nVectorsum \*= sqrt(X);

if(overflowTest != nVectorsum){

throw(QString("Overflow occured in calculation with " + QString::number(overflowTest)));

}

overflowTest = 0.0;

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

if(i >= mVector.size())throw(QString("Sum count went beyond in M vector"));

mVectorsum += \*(mVector.begin() + i);

overflowTest += \*(mVector.begin() + i);

}

mVectorsum \*= log(100 - X);

overflowTest \*= log(100 - X);

if(overflowTest != mVectorsum){

throw(QString("Overflow occured in calculation with " + QString::number(overflowTest)));

}

overflowTest = 0.0;

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

if(i >= kVector.size())throw(QString("Sum count went beyond in K vector"));

kVectorsum += \*(kVector.begin() + i);

overflowTest += \*(kVector.begin() + i);

}

overflowTest /= X - 10;

kVectorsum /= X - 10;

if(overflowTest != kVectorsum)throw("Overflow occured in calculation with " + QString::number(overflowTest));

overflowTest = 0.0;

overflowTest += mVectorsum + kVectorsum;

nVectorsum += mVectorsum + kVectorsum;

if(overflowTest != nVectorsum)throw("Overflow occured in calculation with " + QString::number(overflowTest));

return nVectorsum;

}

void Vyraz::setN(int n)

{

if(n <= 0){

throw(QString("N is negative or zero"));

}

this->n = n;

}

void Vyraz::setM(int m)

{

if(n <= 0){

throw Exception("M is negative or zero");

}

this->m = m;

}

void Vyraz::setK(int k)

{

if(n == 0){

throw EqualZero("K is equal zero");

}

if(n <= 0){

throw(QString("K is negative"));

}

this->k = k;

}

void Vyraz::setX(double x){

this->X = x;

}

void Vyraz::setNVector(std::vector<double> nVector){

if(nVector.size() > n){

throw(QString("Size n doesnt match N"));

}

this->nVector = nVector;

}

void Vyraz::setMVector(std::vector<double> mVector){

if(mVector.size() > m){

throw(QString("Size m doesnt match M"));

}

this->mVector = mVector;

}

void Vyraz::setKVector(std::vector<double> kVector){

if(kVector.size() > k){

throw(QString("Size k doesnt match K"));

}

this->kVector = kVector;

}

**Виконання лабораторної роботи**

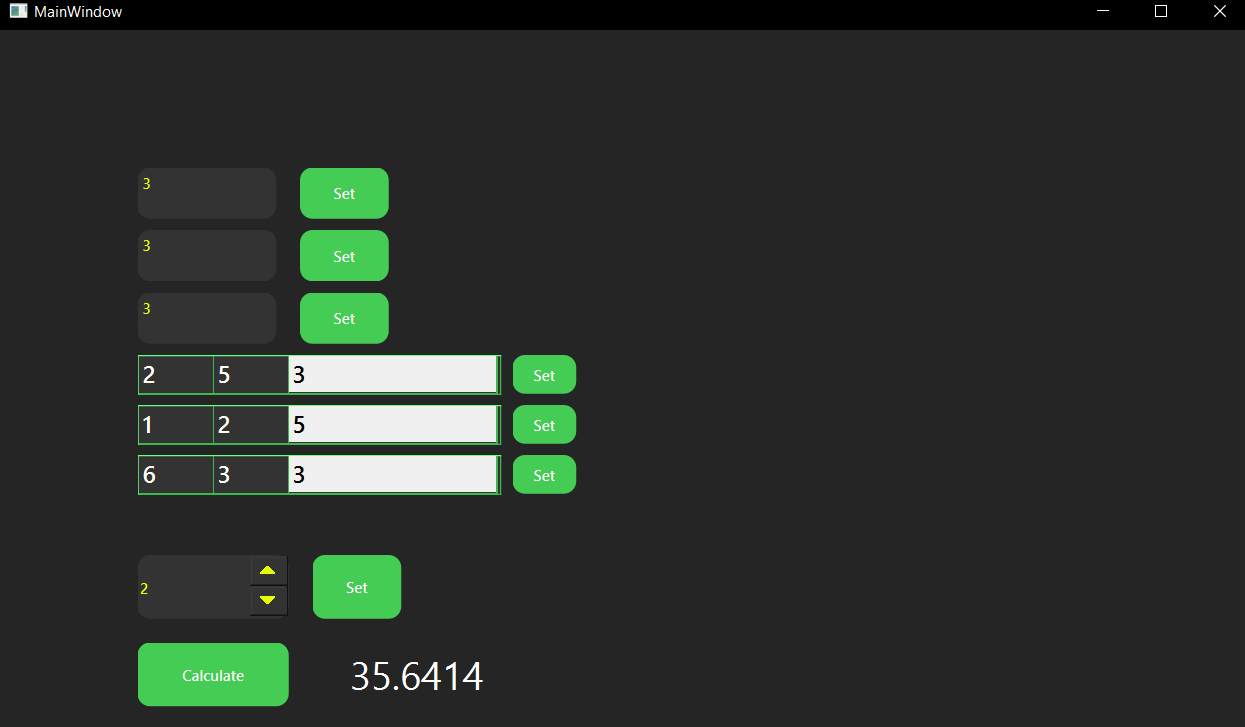


Рис. 1 Виконання програми

**Висновки**

На цій лабораторній роботі я ознайомився з синтаксисом та принципами використання винятків, навчивсяя передбачати виняткові ситуації, які можуть виникнути в процесі роботи програмного забезпечення, а також навчився їх перехоплювати та опрацьовувати..