 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №7**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Виняткові ситуації»

Варіант №15

**Виконав:**

студент гр. БС-71

Орлівський С.П.

**Перевірив:**

ac. Рисін С.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2019

**🞏 Практичне заняття без зауважень**

**🞏 Практичне заняття має зауваження:**

**🞏 несвоєчасний захист**

**🞏 присутні зауваження до UML діаграми:**

**🞏 діаграма класу не відповідає коду**

**🞏 виконані не за стандартом:**

**🞏 атрибути**

**🞏 відношення**

**🞏 потужність**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 хибно задані специфікатори доступу**

**🞏 помилки у визначенні конструкторів / деструкторів**

**🞏 відсутні списки ініціалізації в конструкторах**

**🞏 константні методи**

**🞏 використано глобальні змінні**

**🞏 статичні змінні при роботі з масивами**

**🞏 оформлення коду**

**🞏 присутні зайві символи «{» та «}»**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 невірні відповіді на запитання:**

**🞏 №1 🞏 №2 🞏 №3 🞏 №4 🞏 №5**

**🞏 №6 🞏 №7 🞏 №8 🞏 №9 🞏 №10**

**🞏 маються інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

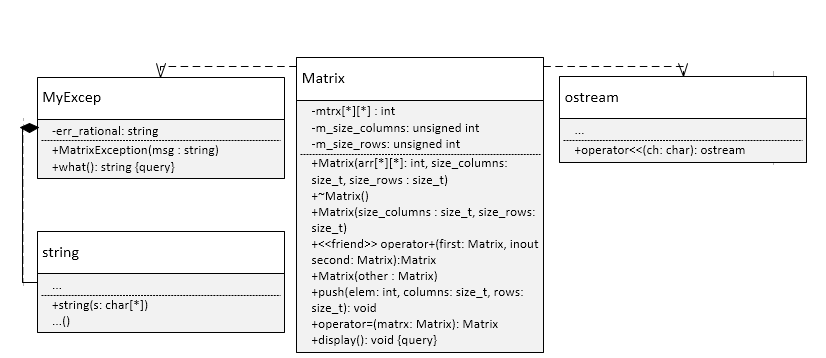
**Завдання:**

1. Ознайомитися з теоретичними відомостями обробки виняткових ситуацій в мові С++.
2. Створити клас Exception, що дозволяє обробляти зазначену в завданні виняткову ситуацію.
3. Розробити діаграму класів в нотації UML та програмний застосунок для демонстрації роботи програми в нормальному режимі та під час виникнення виняткової ситуації відповідно до свого варіанту (визначення класів та їх реалізації мають бути розташовані у файлах \*.h та \*.cpp відповідно; для ініціалізації даних класів використовувати введення з клавіатури; обробка виняткової ситуації має включати виведення на екран відповідного повідомлення з текстом помилки):

***Завдання: Розробити клас для роботи із матрицею цілих чисел, для якого реалізовано дружню функцію-оператор + знаходження суми двох матриць. Виключити ситуацію несумісності розмірів матриць для додавання.***

1. Скласти і захистити звіт по роботі.

**UML діаграма:**



**Код:**

**MyExcp.h**

#pragma once

#include <string>

#include <cstring>

class MyExcp

{

    std::string err\_rational;

public:

    MyExcp(std::string);

    std::string what() const;

};

**MyExcp.cpp**

#include "MyExcp.h"

MyExcp::MyExcp(std::string s)

{

    err\_rational = s;

}

std::string MyExcp::what() const

{

    return err\_rational;

}

**Matrix.h**

#pragma once

#include <iostream>

#include "MyExcp.h"

class Matrix

{

    int \*\*mtrx;

    size\_t m\_size\_columns;

    size\_t m\_size\_rows;

public:

    Matrix(int \*\*arr, size\_t size\_columns, size\_t size\_rows);

    Matrix(size\_t size\_columns, size\_t size\_rows);

    Matrix(const Matrix &other);

    void display() const;

    friend Matrix operator+(const Matrix &matr\_first, const Matrix &matr\_second);

    Matrix &operator=(const Matrix &other);

    void push(int elem, size\_t columns, size\_t rows);

    ~Matrix();

};

**Matrix.cpp**

#include "Matrix.h"

Matrix::Matrix(int \*\*matrix, size\_t size\_columns, size\_t size\_row) : m\_size\_columns(size\_columns), m\_size\_rows(size\_row)

{

    matrix = new int \*[size\_columns];

    for (size\_t i = 0; i < size\_columns; ++i)

    {

        matrix[i] = new int[size\_row];

        for (size\_t j = 0; j < size\_row; ++j)

        {

            matrix[i][j] = matrix[i][j];

        }

    }

}

Matrix operator+(const Matrix &first\_matr, const Matrix &second\_matr)

{

    if (first\_matr.m\_size\_columns != second\_matr.m\_size\_columns || first\_matr.m\_size\_rows != second\_matr.m\_size\_rows)

        throw MatrixException("Sizes of matrixes mismatch");

    Matrix m(first\_matr.m\_size\_columns, first\_matr.m\_size\_rows);

    for (size\_t i = 0; i < first\_matr.m\_size\_columns; ++i)

    {

        for (size\_t j = 0; j < first\_matr.m\_size\_rows; ++j)

        {

            m.mtrx[i][j] = first\_matr.mtrx[i][j] + second\_matr.mtrx[i][j];

        }

    }

    return m;

}

Matrix::Matrix(size\_t size\_columns, size\_t size\_rows) : m\_size\_columns(size\_columns), m\_size\_rows(size\_rows)

{

    mtrx = new int \*[m\_size\_columns];

    for (size\_t i = 0; i < m\_size\_columns; ++i)

    {

        mtrx[i] = new int[m\_size\_rows];

    }

}

Matrix::Matrix(const Matrix &alien) : m\_size\_columns(alien.m\_size\_columns), m\_size\_rows(alien.m\_size\_rows)

{

    mtrx = new int \*[m\_size\_columns];

    for (unsigned int i = 0; i < m\_size\_columns; ++i)

    {

        mtrx[i] = new int[m\_size\_rows];

        for (size\_t j = 0; j < m\_size\_rows; ++j)

        {

            mtrx[i][j] = alien.mtrx[i][j];

        }

    }

}

void Matrix::display() const

{

    for (size\_t i = 0; i < m\_size\_columns; ++i)

    {

        for (size\_t j = 0; j < m\_size\_rows; ++j)

        {

            std::cout << this->mtrx[i][j] << " ";

        }

        std::cout << "\n";

    }

}

Matrix &Matrix::operator=(const Matrix &other)

{

    delete[] mtrx;

    m\_size\_rows = other.m\_size\_rows;

    m\_size\_columns = other.m\_size\_columns;

    mtrx = new int \*[m\_size\_columns];

    for (unsigned int i = 0; i < m\_size\_columns; ++i)

    {

        mtrx[i] = new int[m\_size\_rows];

        for (unsigned int j = 0; j < m\_size\_rows; ++j)

        {

            mtrx[i][j] = other.mtrx[i][j];

        }

    }

    return \*this;

}

void Matrix::push(int element, size\_t indx\_columns, size\_t indx\_row)

{

    if (indx\_columns >= m\_size\_columns || indx\_row >= m\_size\_rows)

        throw MatrixException("wrong index was provided");

    mtrx[indx\_columns][indx\_row] = element;

}

Matrix::~Matrix()

{

    for (size\_t i = 0; i < m\_size\_columns; ++i)

    {

        delete[] mtrx[i];

    }

    delete[] mtrx;

}

**Source.cpp**

#include "matrix.h"

#include <iostream>

#include <vector>

int main()

{

    bool ok = false;

    int temp;

    size\_t n, m;

    const size\_t pair = 2;

    while (!ok)

    {

        std::vector<Matrix> mv;

        mv.reserve(2);

        for (size\_t i = 0; i < pair; ++i)

        {

            std::cout << "Input m: ";

            std::cin >> m;

            std::cout << "Input n: ";

            std::cin >> n;

            Matrix matr\_ft(m, n);

            for (size\_t j = 0; j < m; ++j)

            {

                for (size\_t k = 0; k < n; ++k)

                {

                    std::cout << "Input val for matr: ";

                    std::cin >> temp;

                    matr\_ft.push(temp, j, k);

                }

            }

            matr\_ft.display();

            std::cout << "\n";

            mv.push\_back(matr\_ft);

        }

        Matrix matr\_res(\*mv.cbegin());

        for (std::vector<Matrix>::const\_iterator it = ++mv.cbegin(); it != mv.cend(); ++it)

        {

            try

            {

                matr\_res = matr\_res + \*it;

                ok = true;

            }

            catch (const MyExcp &e)

            {

                std::cout << e.what() << std::endl;

            }

        }

        if (ok)

        {

            matr\_res.display();

        }

    }

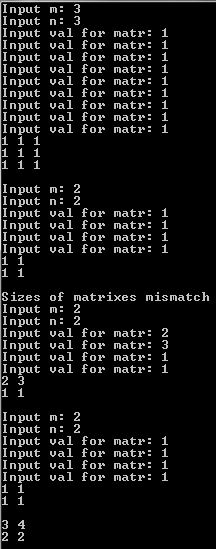
    std::cin.get();

    std::cin.get();

    return 0;

}

**Робота програми:**



**Контрольні запитання**

1. **Що називається винятковою ситуацією?**

Виняткова ситуація – це ситуація, що виникла в певний момент виконання програми, що унеможливлює подальшу коректну роботу програми і потребує особливої уваги та обробки.

1. **Що називається обробкою виняткової ситуації?**

Перевірка умов, що описують виняткову ситуацію, і реакція на її виникнення.

1. **Як здійснюється обробка виняткової ситуації в мові С++?**

Обробка виняткових ситуації в мові С++ використовує три ключових слова: *try, throw, catch.*

Загальний формат цих блоків:

try

{

// код, що може генерувати виняткові ситуації

throw[expression]

}

[catch (exception-declaration)

{

//код, який виконується у випадку відповідності exception-declaration до

//типу згенерованого в блоці *try* винятку

}

[catch (exception-declaration)

{

//код, що обробляє інший тип винятку

}]…]

Як операнд виразу *throw* можна використовувати об’єкт будь-якого типу. Для обробки винятків, необхідно реалізувати один або кілька блоків *catch* відразу після блоку *try.* Саме код всередині *catch* обробляє виняткову ситуацію.

Якщо замість типу оператор *catch* задати через три крапки (…), то такий блок *catch* обробляє усі типи винятків. Але такий блок повинен бути розміщений останнім, оскільки блоки *catch* обробляються в порядку коду програми.

1. **Чи можна помістити блок *try* у функцію, залишивши блок *catch* для обробки виняткових ситуацій у функції *main()*? Відповідь пояснити.**

Ні, блок try обов’язково повинен містити після себе блок обробки винятків *catch*. В інакшому випадку компілятор не зрозуміє як обробити ,створений у блоці *try,* виняток.

1. **До чого призведе передавання винятку в блок *catch* за посиланням?**

Уникнення створення копії об’єкта винятку, та можливості його зміни (див. пит. 6), що залишиться навіть після виходу з блоку обробки.

1. **Для чого використовують вираз *throw*, що не містить операндів?**

class MyException {

public:

// ...

void addInfo(const std::string& info);

// ...

};

void f()

{

try {

// ...

}

catch (MyException& error) {

error.addInfo("f() failed");

throw;

}

}

В даному прикладі вираз *throw* повторно «викидає» попередній виняток ‘**error**’. Функція «зловила» виняток, змінила цей виняток(додавши до нього інформацію за допомогою addInfo()), і повторно «викинула» цей виняток за допомогою виразу *throw* без аргументів. Так *throw* використовується для створення простої форми stack-trace, додаючи відповідні *catch* випадки у важливих місцях програми. Вираз *throw* ззовні блоку *catch* призведе до виклику функції *terminate().*

1. **До чого призведе генерація похідної виняткової ситуації за умови наявності блоку *catch,* що призначений для обробки базової виняткової ситуації? Відповідь пояснити.**

*catch* зможе "спіймати" винятки похідного класу і базового. Наприклад:

#include <iostream>

class ExceptionBase {

};

class MyException : public ExceptionBase {

};

int main()

{

try

{

throw MyException();

}

catch (MyException const& e) {

std::cout << "catch 1" << std::endl;

}

catch (ExceptionBase const& e) {

std::cout << "should not catch 1" << std::endl;

}

////////

try

{

throw MyException();

}

catch (ExceptionBase const& e) {

std::cout << "catch 2" << std::endl;

}

catch (...) {

std::cout << "should not catch 2" << std::endl;

}

return 0;

}

В результаті буде наступний вивід на екран:

catch 1  
catch 2

1. **Назвіть і опишіть стандартні типи винятків з бібліотеки *std*?**

std::logic\_error – повертає помилку, що є наслідком неправильної логіки програми, наприклад порушення логічних умов або інваріантів класу.

std::runtime\_error – повертає помилку, що є наслідком процесів за межами програми.

std::bad\_typeid – повертає помилку у випадку, коли typeid оператор був використаний з разіменованною змінною, що має значення nullptr та є поліморфним типом.

std::bad\_alloc – тип об’єкта що викликається як виняток функціями виділення пам’яті при неправильному виділенні пам’яті.

std::bad\_cast – повертає помилку, що є наслідком неправильної операції переведення динамічного типу.

std::bad\_exception – повертає помилку коли std::current\_exception не може зробити копію об’єкта-винятку.

std::overflow\_error – повертає помилку коли результат арифметичної операції перевищує допустиме значення для певного типу.

1. **Для чого під час обробки виняткових ситуації використовують призначені функції *terminate(), unexpected(), abort()*?**

Не усі винятки можуть бути «спіймані» блоком *catch,* в деяких випадках найкращим способом обробити виняток є припинення роботи програми. Для цього використовуються 2 функції бібліотеки: ***terminate()і unexpected().***

1. Коли функція зі специфікацією виключення викидає виняток, що не був передбачений в специфікації, то викликається функція **void unexpected().**
2. Функція **void\_unexpecter()** викликає функцію, що була вказана в функції **set\_unexpected(),** якщо ж така функція не була вказана, то за замовчуванням викликається функція **terminate().**
3. Функція **terminate()** викликає функцію, зазначену у функції **set\_terminate()**, якщо така функція не була зазначена, то за замовчуванням викликається функція **abort()**
4. **abort()** аварійно завершує роботу програми викликавши функцію **\_exit().**
5. **\_exit()** завершує роботу програми без закриття файлів, очищення буферів, та визову функцій.
6. **Навести приклад коду, за допомогою якого можна вивести на екран текстове повідомлення з номером строчки, під час виконання якої сталася виняткова ситуація, та назвою файлу, що містить цю строчку.**

#include <iostream>

#include <stdexcept>

#include <string>

using namespace std;

inline const string

Exception\_helper(const string msg,

const char\* file\_name,

const size\_t line\_number) {

return msg + "in " + file\_name + ":" +

to\_string(line\_number);

}

void Foo() {

throw logic\_error(Exception\_helper("There is an error ",

\_\_FILE\_\_,

\_\_LINE\_\_));

}

int main() {

try

{

Foo();

}

catch (std::exception & e)

{

cout << e.what() << endl;

return 1;

}

return 0;

}