Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

**Лабораторна робота №5**

**Тема:** «Перезавантаження операторів»

|  |
| --- |
| Виконав: ст. гр. КН-23 |
| Кобзар М. В. |
| Перевірив: Козірова Н. Л. |
|  |

Кропивницький

2024

**Тема: Перезавантаження операторів**

**Мета:** Ознайомитись з поняттям перевантаження операторів та навчитись їх програпно реалізовувати мовою С++.

**Завдання:**

***Завдання 1:***

Перевантажте оператори для завдання № 2 з лабораторної роботи №2.

***Завдання 2:***

Продовжить розробку гри «Спіймай муху», тепер ваша муха має стати класом, реалізуйте декілька рівнів гри, на кожному наступному рівні, має з’являтися нова муха, яка буде класом наслідником. Також, з другого рівня на вашій формі має з’явитися вікно, муха має намагатися вилетіти в це вікно, задача гравця відігнати муху від вікна і загнати її в пастку.

***Завдання 3:***

**Варіант 8**

Створіть клас Rectangle, який представляє прямокутник з заданими шириною та висотою. Ширина та висота повинні бути додатними цілими числами. В класі Rectangle перевантажте наступні оператори:

1. Оператор + для об'єднання двох прямокутників. Результатом має бути новий прямокутник, який охоплює обидва початкових прямокутники.
2. Оператор - для віднімання одного прямокутника від іншого. Результатом має бути новий прямокутник, який представляє різницю між початковими прямокутниками.
3. Оператор \* для масштабування прямокутника. Передавайте ціле число як аргумент, і результатом має бути новий прямокутник, який має ширину та висоту, помножені на це число.
4. Оператор / для зменшення прямокутника. Передавайте ціле число як аргумент, і результатом має бути новий прямокутник, який має ширину та висоту, поділені на це число.
5. Оператор == для порівняння двох прямокутників на рівність.
6. Оператор != для порівняння двох прямокутників на нерівність.
7. Оператор << для виводу прямокутника у зрозумілому форматі.

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з прямокутниками.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Rectangle і виконайте з ними операції об'єднання, віднімання, масштабування, зменшення, порівняння на рівність та виведення на екран.

**Варіант 8**

**Лістинг :**

**Завдання 1:**

**Set.h**

#ifndef SET\_H

#define SET\_H

#include <iostream>

#include <vector>

#include <string>

class Set {

private:

std::vector<std::string> elements;

public:

Set(); // Конструктор

void addElement(const std::string& element);

bool contains(const std::string& element) const;

void inputElements();

void display() const;

Set intersection(const Set& other) const;

Set unionSet(const Set& other) const;

Set difference(const Set& other) const;

Set symmetricDifference(const Set& other) const;

// Перевантажені оператори

Set operator+(const Set& other) const; // Об'єднання

Set operator-(const Set& other) const; // Різниця

Set operator\*(const Set& other) const; // Перетин

Set operator^(const Set& other) const; // Симетрична різниця

bool operator==(const Set& other) const; // Перевірка на рівність

bool operator!=(const Set& other) const; // Перевірка на нерівність

};

#endif // SET\_H

**Set.cpp**

#include "Set.h"

Set::Set() {}

Set Set::operator+(const Set& other) const {

return this->unionSet(other);

}

Set Set::operator-(const Set& other) const {

return this->difference(other);

}

Set Set::operator\*(const Set& other) const {

return this->intersection(other);

}

Set Set::operator^(const Set& other) const {

return this->symmetricDifference(other);

}

bool Set::operator==(const Set& other) const {

if (elements.size() != other.elements.size()) {

return false;

}

for (const auto& elem : elements) {

if (!other.contains(elem)) {

return false;

}

}

return true;

}

bool Set::operator!=(const Set& other) const {

return !(\*this == other);

}

void Set::addElement(const std::string& element) {

if (!contains(element)) {

elements.push\_back(element);

}

}

bool Set::contains(const std::string& element) const {

for (const auto& elem : elements) {

if (elem == element) {

return true;

}

}

return false;

}

void Set::inputElements() {

std::string input;

while (true) {

std::getline(std::cin, input);

if (input == "done") break;

addElement(input);

}

}

void Set::display() const {

for (const auto& elem : elements) {

std::cout << elem << " ";

}

std::cout << std::endl;

}

Set Set::intersection(const Set& other) const {

Set result;

for (const auto& elem : elements) {

if (other.contains(elem)) {

result.addElement(elem);

}

}

return result;

}

Set Set::unionSet(const Set& other) const {

Set result;

for (const auto& elem : elements) {

result.addElement(elem);

}

for (const auto& elem : other.elements) {

result.addElement(elem);

}

return result;

}

Set Set::difference(const Set& other) const {

Set result;

for (const auto& elem : elements) {

if (!other.contains(elem)) {

result.addElement(elem);

}

}

return result;

}

Set Set::symmetricDifference(const Set& other) const {

Set result = this->unionSet(other).difference(this->intersection(other));

return result;

}

**Main.cpp**

#include <iostream>

#include "Set.h"

using namespace std;

int main() {

Set set1, set2;

set1.addElement("apple");

set1.addElement("banana");

set1.addElement("cherry");

set2.addElement("banana");

set2.addElement("dragonfruit");

set2.addElement("elderberry");

// Перевірка операцій

Set unionSet = set1 + set2;

Set intersectionSet = set1 \* set2;

Set differenceSet = set1 - set2;

Set symDifferenceSet = set1 ^ set2;

cout << "Set 1: ";

set1.display();

cout << "Set 2: ";

set2.display();

cout << "Union: ";

unionSet.display();

cout << "Intersection: ";

intersectionSet.display();

cout << "Difference (Set 1 - Set 2): ";

differenceSet.display();

cout << "Symmetric Difference: ";

symDifferenceSet.display();

bool areEqual = (set1 == set2);

cout << "Are Set 1 and Set 2 equal? " << (areEqual ? "Yes" : "No") << endl;

bool areNotEqual = (set1 != set2);

cout << "Are Set 1 and Set 2 not equal? " << (areNotEqual ? "Yes" : "No") << endl;

return 0;

}

**Завдання 2:**

**Main.cpp**

#include "muha.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

muha w;

w.show();

return a.exec();

}

**Muha.cpp**

#include "muha.h"

#include "ui\_muha.h"

#include <QPixmap>

#include <QDebug>

#include <QRandomGenerator>

#include <QMessageBox>

muha::muha(QWidget \*parent)

: QMainWindow(parent), ui(new Ui::muha), level(1)

{

ui->setupUi(this);

qDebug() << "Constructor called";

setupLevel(level);

}

muha::~muha()

{

qDebug() << "Destructor called";

delete ui;

}

bool muha::event(QEvent \*e)

{

if (e->type() == QEvent::MouseMove) {

QMouseEvent \*mouseEvent = static\_cast<QMouseEvent\*>(e);

qDebug() << "Mouse move event at position:" << mouseEvent->pos();

moveFlyAway(mouseEvent->pos());

if (checkCollision()) {

QMessageBox::information(this, "Win", "піймався");

level++;

if (level > 10) level = 1; // Повторне встановлення рівня на 1 після 10

setupLevel(level);

} else if (level >= 2 && checkWindowCollision()) {

QMessageBox::information(this, "Loss", "муха втекла через вікно");

qApp->quit(); // Закриття гри після програшу

}

}

return QMainWindow::event(e);

}

void muha::moveFlyAway(const QPoint &mousePos)

{

if (!fly) {

qDebug() << "Fly label is null!";

return;

}

int flyX = fly->x();

int flyY = fly->y();

int deltaX = flyX - mousePos.x();

int deltaY = flyY - mousePos.y();

int newX = flyX;

int newY = flyY;

if (qAbs(deltaX) < 50 && qAbs(deltaY) < 50) {

newX = flyX + (deltaX > 0 ? 1 : -1);

newY = flyY + (deltaY > 0 ? 1 : -1);

} else if (level >= 2) {

// Додаємо логіку для руху мухи до вікна

int windowX = window->x();

int windowY = window->y();

if (flyX < windowX)

newX = flyX + 1;

else if (flyX > windowX)

newX = flyX - 1;

if (flyY < windowY)

newY = flyY + 1;

else if (flyY > windowY)

newY = flyY - 1;

}

newX = qBound(0, newX, this->width() - fly->width());

newY = qBound(0, newY, this->height() - fly->height());

qDebug() << "New fly position:" << newX << newY;

fly->move(newX, newY);

}

bool muha::checkCollision()

{

if (!fly || !trap) {

qDebug() << "Fly or trap label is null!";

return false;

}

QRect flyRect = fly->geometry();

QRect trapRect = trap->geometry();

bool collided = flyRect.intersects(trapRect);

qDebug() << "Checking collision:" << collided;

return collided;

}

bool muha::checkWindowCollision()

{

if (!fly || !window) {

qDebug() << "Fly or window label is null!";

return false;

}

QRect flyRect = fly->geometry();

QRect windowRect = window->geometry();

bool collided = flyRect.intersects(windowRect);

qDebug() << "Checking window collision:" << collided;

return collided;

}

void muha::setupLevel(int lvl)

{

qDebug() << "Setting up level:" << lvl;

level = lvl;

if (fly) fly->deleteLater();

if (trap) trap->deleteLater();

if (window) window->deleteLater();

fly = new QLabel(this);

if (!fly) {

qDebug() << "Failed to create fly label!";

return;

}

QPixmap pix;

if (!pix.load("C://Users/kobza/OneDrive/Рабочий стол/PROJECT\_KNTU/OOP\_lab/lab\_5/Завдання 2/lab5\_2/muha.png")) {

qDebug() << "Failed to load muha.png";

return;

}

fly->setPixmap(pix);

fly->setScaledContents(true);

fly->setGeometry(50, 50, 50, 50);

fly->show();

trap = new QLabel(this);

if (!trap) {

qDebug() << "Failed to create trap label!";

return;

}

QPixmap trapPix;

if (!trapPix.load("C://Users/kobza/OneDrive/Рабочий стол/PROJECT\_KNTU/OOP\_lab/lab\_5/Завдання 2/lab5\_2/trap.png")) {

qDebug() << "Failed to load trap.png";

return;

}

trap->setPixmap(trapPix);

trap->setScaledContents(true);

trap->setGeometry(300, 300, 100, 100);

trap->show();

if (level >= 2) {

window = new QLabel(this);

if (!window) {

qDebug() << "Failed to create window label!";

return;

}

QPixmap windowPix;

if (!windowPix.load("C://Users/kobza/OneDrive/Рабочий стол/PROJECT\_KNTU/OOP\_lab/lab\_5/Завдання 2/lab5\_2/window.png")) {

qDebug() << "Failed to load window.png";

return;

}

window->setPixmap(windowPix);

window->setScaledContents(true);

window->setGeometry(500, 50, 100, 100);

window->show();

}

randomizeTrapAndWindow(); // Випадкова зміна положення капкана і вікна

if (!levelLabel) {

levelLabel = new QLabel(this);

levelLabel->setGeometry(10, 10, 100, 30);

}

updateLevelLabel();

qDebug() << "Level setup complete.";

}

void muha::randomizeTrapAndWindow()

{

int newX, newY;

// Випадкове положення для капкана

newX = QRandomGenerator::global()->bounded(this->width() - trap->width());

newY = QRandomGenerator::global()->bounded(this->height() - trap->height());

trap->move(newX, newY);

qDebug() << "Trap moved to:" << newX << newY;

// Випадкове положення для вікна

if (window) {

newX = QRandomGenerator::global()->bounded(this->width() - window->width());

newY = QRandomGenerator::global()->bounded(this->height() - window->height());

window->move(newX, newY);

qDebug() << "Window moved to:" << newX << newY;

}

}

void muha::updateLevelLabel()

{

levelLabel->setText("Level: " + QString::number(level));

qDebug() << "Level label updated to:" << level;

}

**Muha.h**

#ifndef MUHA\_H

#define MUHA\_H

#include <QMainWindow>

#include <QEvent>

#include <QMouseEvent>

#include <QLabel>

QT\_BEGIN\_NAMESPACE

namespace Ui {

class muha;

}

QT\_END\_NAMESPACE

class muha : public QMainWindow

{

Q\_OBJECT

public:

muha(QWidget \*parent = nullptr);

~muha();

protected:

bool event(QEvent \*e) override;

private:

Ui::muha \*ui;

QLabel \*fly = nullptr;

QLabel \*trap = nullptr;

QLabel \*window = nullptr;

QLabel \*levelLabel = nullptr;

int level;

void moveFlyAway(const QPoint &mousePos);

void moveFlyToWindow();

bool checkCollision();

bool checkWindowCollision();

void setupLevel(int lvl);

void updateLevelLabel();

void randomizeTrapAndWindow(); // Нова функція для випадкової зміни положення

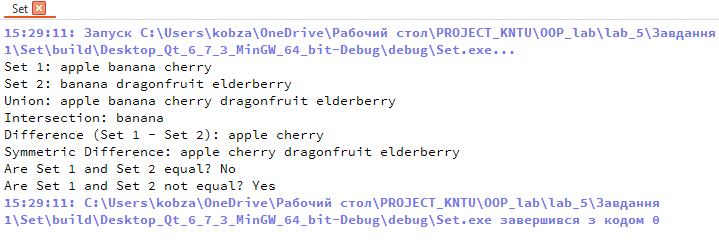
};

#endif // MUHA\_H

**Завдання 3:**

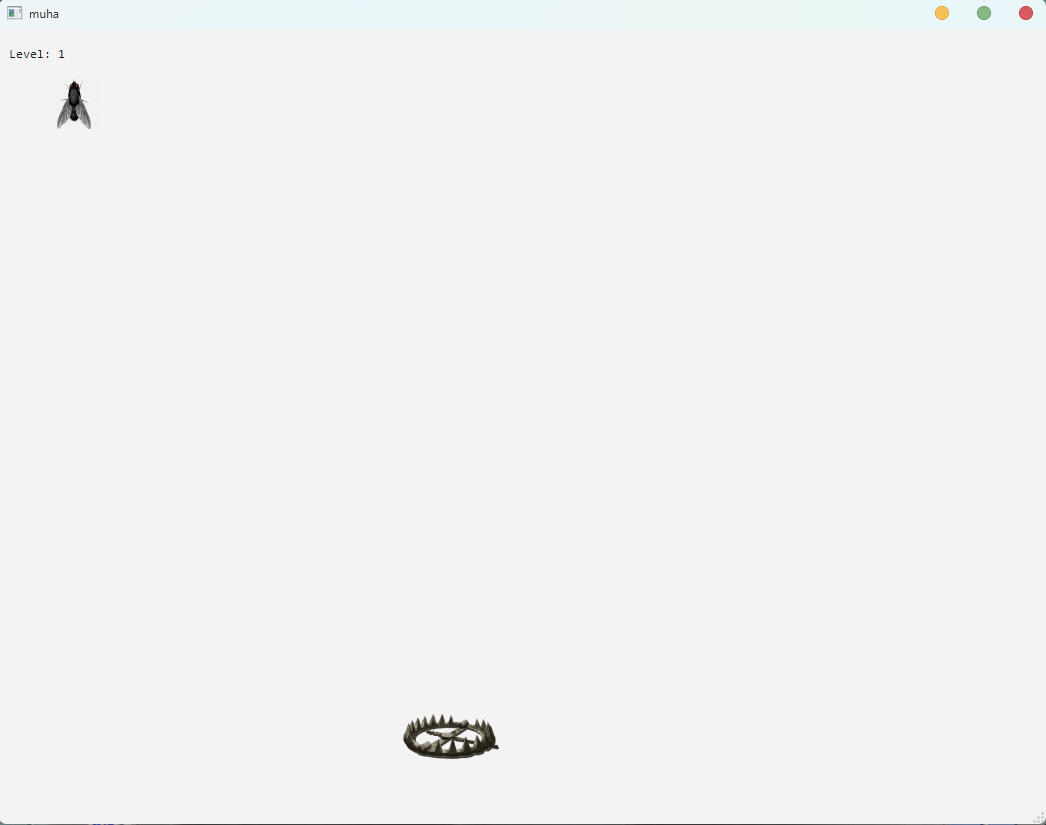
**Результат:**

**Завдання 1**

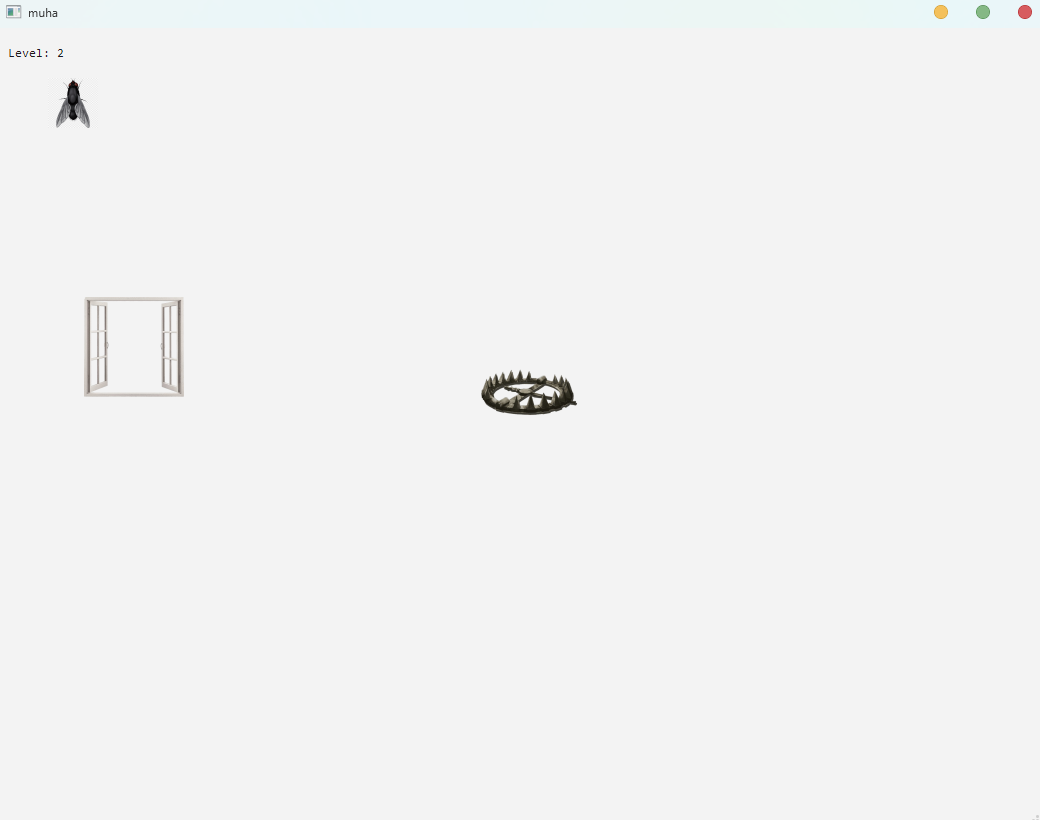
****

**Завдання 2**

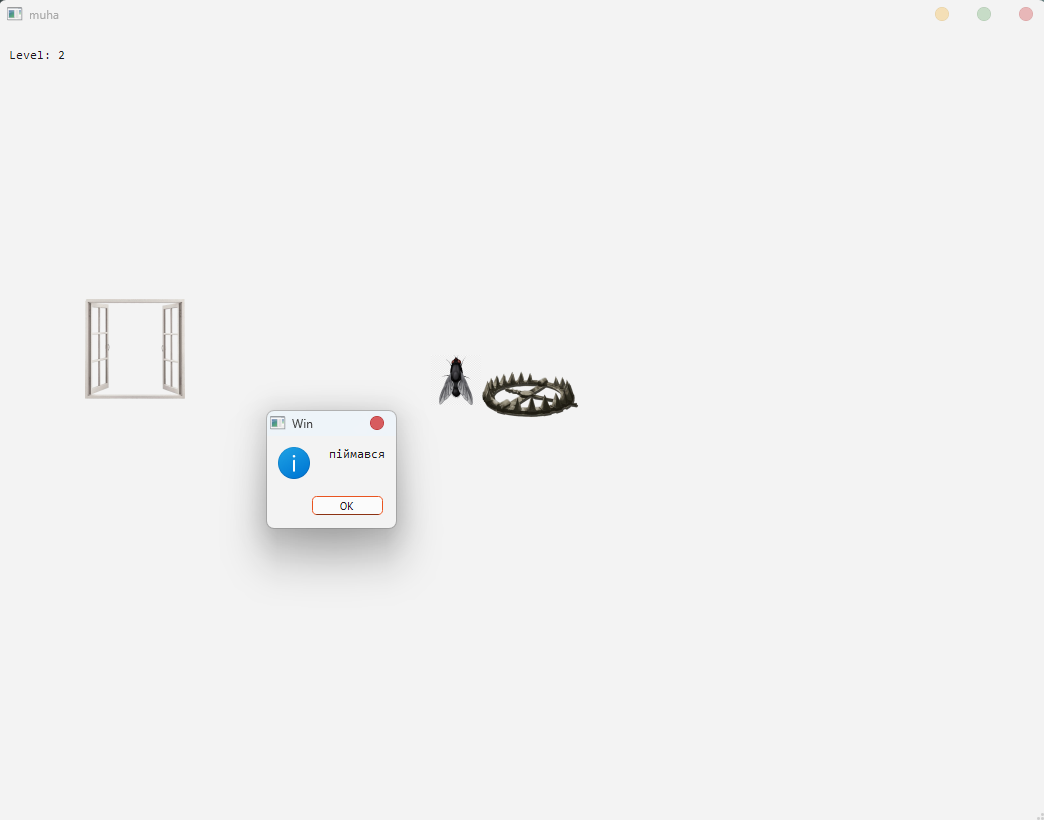
1 рівень



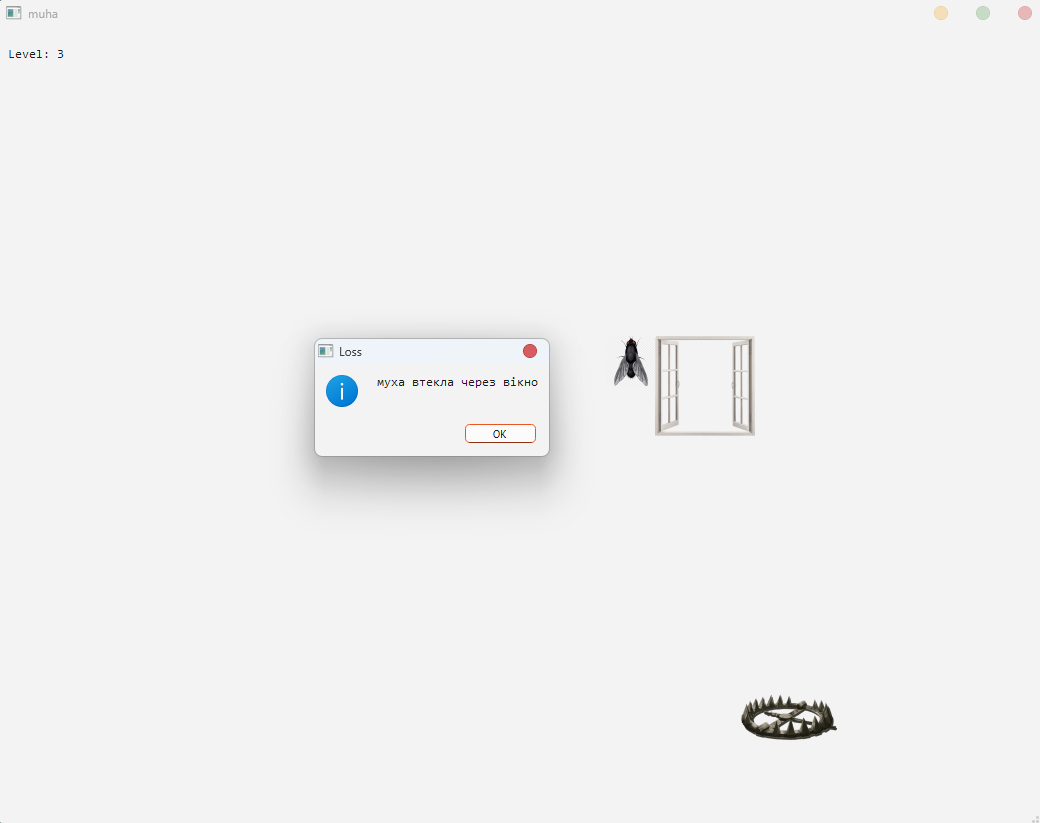
2 рівень



Якщо загнали у капкан



3 рівень та якщо муха втекла



**Частина 3**

Rectangle.h

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

class Rectangle {

public:

Rectangle(int width, int height);

~Rectangle();

Rectangle operator+(const Rectangle& other) const;

Rectangle operator-(const Rectangle& other) const;

Rectangle operator\*(int factor) const;

Rectangle operator/(int divisor) const;

bool operator==(const Rectangle& other) const;

bool operator!=(const Rectangle& other) const;

friend std::ostream& operator<<(std::ostream& os, const Rectangle& rect);

private:

int width;

int height;

};

#endif // RECTANGLE\_H

Rectangle.cpp

#include "rectangle.h"

#include <algorithm>

Rectangle::Rectangle(int width, int height)

: width(width), height(height)

{

if (width <= 0 || height <= 0) {

throw std::invalid\_argument("Width and height must be positive integers.");

}

}

Rectangle::~Rectangle() {}

Rectangle Rectangle::operator+(const Rectangle& other) const {

int newWidth = std::max(width, other.width);

int newHeight = std::max(height, other.height);

return Rectangle(newWidth, newHeight);

}

Rectangle Rectangle::operator-(const Rectangle& other) const {

int newWidth = std::max(1, width - other.width); // Мінімальна ширина 1

int newHeight = std::max(1, height - other.height); // Мінімальна висота 1

return Rectangle(newWidth, newHeight);

}

Rectangle Rectangle::operator\*(int factor) const {

return Rectangle(width \* factor, height \* factor);

}

Rectangle Rectangle::operator/(int divisor) const {

if (divisor == 0) {

throw std::invalid\_argument("Division by zero is not allowed.");

}

return Rectangle(width / divisor, height / divisor);

}

bool Rectangle::operator==(const Rectangle& other) const {

return width == other.width && height == other.height;

}

bool Rectangle::operator!=(const Rectangle& other) const {

return !(\*this == other);

}

std::ostream& operator<<(std::ostream& os, const Rectangle& rect) {

os << "Rectangle(width: " << rect.width << ", height: " << rect.height << ")";

return os;

}

Main.cpp

#include <iostream>

#include "rectangle.h"

int main() {

try {

Rectangle rect1(10, 20);

Rectangle rect2(15, 25);

Rectangle rect3 = rect1 + rect2;

std::cout << "rect1 + rect2 = " << rect3 << std::endl;

Rectangle rect4 = rect1 - rect2;

std::cout << "rect1 - rect2 = " << rect4 << std::endl;

Rectangle rect5 = rect1 \* 2;

std::cout << "rect1 \* 2 = " << rect5 << std::endl;

Rectangle rect6 = rect2 / 2;

std::cout << "rect2 / 2 = " << rect6 << std::endl;

bool isEqual = rect1 == rect2;

std::cout << "rect1 == rect2: " << (isEqual ? "true" : "false") << std::endl;

bool isNotEqual = rect1 != rect2;

std::cout << "rect1 != rect2: " << (isNotEqual ? "true" : "false") << std::endl;

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

}

return 0;

}

**Висновок :**

Завдання 1: Перевантаження операторів (Лабораторна робота №2)

У першому завданні ми реалізували клас `Set`, що представляє множину рядкових елементів, і перевантажили такі оператори:

1. Оператор + для об'єднання двох множин.

2. Оператор - для різниці між двома множинами.

3. Оператор \* для перетину двох множин.

4. Оператор ^ для симетричної різниці двох множин.

5. Оператор == для порівняння двох множин на рівність.

6. Оператор != для порівняння двох множин на нерівність.

Результати тестування показали коректну роботу перевантажених операторів, що дозволяє виконувати базові операції з множинами.

**Завдання** 2: Гра з мухою

У другому завданні ми створили інтерактивну програму з мухою, яка намагається втекти від курсору. Гра включала наступні функціональні можливості:

1. Рух мухи: Муха уникає курсору миші.

2. Різні рівні: На кожному рівні змінюються позиції капкану та вікна.

3. Притягнення мухи до вікна: Починаючи з другого рівня, муха намагається втекти через вікно.

4. Закриття гри при поразці: Якщо муха втікає через вікно, гра закривається.

5. Перемога на 10 рівні: Виводиться повідомлення "Ви перемогли" і гра закривається, якщо гравець ловить муху на 10 рівні.

Ця реалізація продемонструвала використання Qt для створення інтерактивних програм з графічним інтерфейсом.

**Завдання** 3: Клас Rectangle з перевантаженими операторами

У третьому завданні ми створили клас `Rectangle`, що представляє прямокутник із заданими шириною та висотою. Ми перевантажили такі оператори:

1. Оператор + для об'єднання двох прямокутників.

2. Оператор - для віднімання одного прямокутника від іншого.

3. Оператор \* для масштабування прямокутника.

4. Оператор / для зменшення прямокутника.

5. Оператор == для порівняння двох прямокутників на рівність.

6. Оператор != для порівняння двох прямокутників на нерівність.

7. Оператор << для виведення прямокутника у зрозумілому форматі.

Результати тестування підтвердили правильність реалізації та коректну роботу перевантажених операторів.

Загальний висновок

Успішна реалізація всіх трьох завдань демонструє навички роботи з класами, перевантаження операторів та створення інтерактивних програм за допомогою бібліотеки Qt. Завдання допомогли закріпити знання та вміння з об'єктно-орієнтованого програмування в C++, включаючи обробку множин, роботу з геометричними фігурами та створення графічних інтерфейсів.