Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №5

з дисципліни «Об’єктно-орієнтоване програмування»

**Тема:** «Перевантаження операторів»

|  |
| --- |
| Виконав: ст. гр. КН-23 |
| Гребенюк Д.О. |
| Перевірив: асистент  Козірова Н.Л. |
|  |

Кропивницький

2024

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

**Варіант 4**

**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**

Створіть клас String, який представляє рядок символів. В класі String перевантажте наступні оператори:

1. Оператор + для конкатенації двох рядків.
2. Оператор == для порівняння двох рядків на рівність.
3. Оператор != для порівняння двох рядків на нерівність.
4. Оператор [] для доступу до символу за індексом.
5. Оператор << для виводу рядка на екран.

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

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу String і виконайте з ними операції конкатенації, порівняння на рівність, доступу до символу за індексом та виведення на екран.

**Завдання 1**

*Лістинг bankaccount.cpp*

#include "bankaccount.h"

#include <iostream>

BankAccount::BankAccount(int number, const std::string &owner, long int balance)

{

setNumber(number);

setOwner(owner);

setBalance(balance);

}

int BankAccount::getNumber() const

{

return number;

}

void BankAccount::setNumber(int newNumber)

{

if (newNumber <= 0) {

std::cout << "The account number cannot be less than or equal to zero." << std::endl;

return;

}

number = newNumber;

}

std::string BankAccount::getOwner() const

{

return owner;

}

void BankAccount::setOwner(const std::string &newOwner)

{

if (newOwner.empty()) {

std::cout << "You have not entered the name of the account holder." << std::endl;

return;

}

owner = newOwner;

}

long int BankAccount::getBalance() const

{

return balance;

}

void BankAccount::setBalance(long int newBalance)

{

balance = newBalance;

}

*Лістинг bankaccount.h*

#ifndef BANKACCOUNT\_H

#define BANKACCOUNT\_H

#include <string>

class BankAccount

{

public:

BankAccount(int number, const std::string &owner, long int balance);

virtual BankAccount& operator+(const BankAccount& other) = 0;

virtual BankAccount& operator-(const BankAccount& other) = 0;

virtual BankAccount& operator+=(double amount) = 0;

virtual BankAccount& operator-=(double amount) = 0;

int getNumber() const;

void setNumber(int newNumber);

std::string getOwner() const;

void setOwner(const std::string &newOwner);

long int getBalance() const;

void setBalance(long int newBalance);

virtual void withdraw(double amount) = 0;

virtual void deposit(double amount) = 0;

virtual double calculateInterest() const = 0;

protected:

int number;

std::string owner;

long int balance;

};

#endif // BANKACCOUNT\_H

*Лістинг interestaccount.cpp*

#include "interestaccount.h"

#include <iostream>

InterestAccount::InterestAccount(int number, const std::string &owner, long int balance, int interestRate) : BankAccount(number, owner, balance)

{

setInterestRate(interestRate);

}

InterestAccount& InterestAccount::operator+(const BankAccount &other)

{

deposit(other.getBalance());

return \*this;

}

InterestAccount& InterestAccount::operator-(const BankAccount &other)

{

withdraw(other.getBalance());

return \*this;

}

InterestAccount& InterestAccount::operator+=(double amount)

{

deposit(amount);

return \*this;

}

InterestAccount& InterestAccount::operator-=(double amount)

{

withdraw(amount);

return \*this;

}

int InterestAccount::getInterestRate() const

{

return interestRate;

}

void InterestAccount::setInterestRate(int newInterestRate)

{

if (newInterestRate <= 0) {

std::cout << "The interest rate cannot be equal to or less than zero." << std::endl;

return;

}

interestRate = newInterestRate;

}

void InterestAccount::withdraw(double amount) {

if (amount > balance) {

std::cout << "There are not enough money in your account." << std::endl;

return;

}

balance -= amount;

}

void InterestAccount::deposit(double amount) {

if (amount <= 0) {

std::cout << "The deposit amount cannot be less than or equal to zero." << std::endl;

return;

}

balance += amount;

balance += calculateInterest();

}

double InterestAccount::calculateInterest() const {

return balance \* interestRate / 100;

}

*Лістинг interestaccount.h*

#ifndef INTERESTACCOUNT\_H

#define INTERESTACCOUNT\_H

#include "bankaccount.h"

class InterestAccount : public BankAccount

{

public:

InterestAccount(int number, const std::string &owner, long int balance, int interestRate);

virtual InterestAccount& operator+(const BankAccount& other) override;

virtual InterestAccount& operator-(const BankAccount& other) override;

virtual InterestAccount& operator+=(double amount) override;

virtual InterestAccount& operator-=(double amount) override;

int getInterestRate() const;

void setInterestRate(int newInterestRate);

void withdraw(double amount) override;

void deposit(double amount) override;

double calculateInterest() const override;

private:

int interestRate;

};

#endif // INTERESTACCOUNT\_H

*Лістинг regularaccount.cpp*

#include "regularaccount.h"

#include <iostream>

RegularAccount::RegularAccount(int number, const std::string &owner, long int balance, long int minimumAllowedBalance) : BankAccount(number, owner, balance)

{

setMinimumAllowedBalance(minimumAllowedBalance);

}

RegularAccount& RegularAccount::operator+(const BankAccount &other)

{

deposit(other.getBalance());

return \*this;

}

RegularAccount& RegularAccount::operator-(const BankAccount &other)

{

withdraw(other.getBalance());

return \*this;

}

RegularAccount& RegularAccount::operator+=(double amount)

{

deposit(amount);

return \*this;

}

RegularAccount& RegularAccount::operator-=(double amount)

{

withdraw(amount);

return \*this;

}

long int RegularAccount::getMinimumAllowedBalance() const

{

return minimumAllowedBalance;

}

void RegularAccount::setMinimumAllowedBalance(long int newMinimumAllowedBalance)

{

if (newMinimumAllowedBalance < 0) {

std::cout << "The minimum allowed balance cannot be less than zero." << std::endl;

return;

}

minimumAllowedBalance = newMinimumAllowedBalance;

}

void RegularAccount::withdraw(double amount) {

if (amount > balance) {

std::cout << "There are not enough money in your account." << std::endl;

return;

}

if (balance - amount < minimumAllowedBalance) {

std::cout << "Balance cannot be less than the minimum allowed balance." << std::endl;

return;

}

balance -= amount;

}

void RegularAccount::deposit(double amount) {

if (amount <= 0) {

std::cout << "The deposit amount cannot be less than or equal to zero." << std::endl;

return;

}

balance += amount;

}

double RegularAccount::calculateInterest() const {

return 0.0;

}

*Лістинг regularaccount.h*

#ifndef REGULARACCOUNT\_H

#define REGULARACCOUNT\_H

#include "bankaccount.h"

class RegularAccount : public BankAccount

{

public:

RegularAccount(int number, const std::string &owner, long int balance, long int minimumAllowedBalance);

virtual RegularAccount& operator+(const BankAccount& other) override;

virtual RegularAccount& operator-(const BankAccount& other) override;

virtual RegularAccount& operator+=(double amount) override;

virtual RegularAccount& operator-=(double amount) override;

long int getMinimumAllowedBalance() const;

void setMinimumAllowedBalance(long int newMinimumAllowedBalance);

void withdraw(double amount) override;

void deposit(double amount) override;

double calculateInterest() const override;

private:

long int minimumAllowedBalance;

};

#endif // REGULARACCOUNT\_H

*Лістинг main.cpp*

#include <iostream>

#include <vector>

#include "bankaccount.h"

#include "regularaccount.h"

#include "interestaccount.h"

void showMenu() {

std::cout << "\n1. Create a new regular account\n";

std::cout << "2. Create a new interest account\n";

std::cout << "3. Deposit to account\n";

std::cout << "4. Withdraw from account\n";

std::cout << "5. Switch account\n";

std::cout << "6. Show balance\n";

std::cout << "7. Combine accounts using + operator\n";

std::cout << "8. Subtract accounts using - operator\n";

std::cout << "9. Add amount using += operator\n";

std::cout << "10. Subtract amount using -= operator\n";

std::cout << "11. Exit\n";

std::cout << "Choose an option: ";

}

int main() {

std::vector<BankAccount\*> accounts;

int currentAccount = -1;

while (true) {

showMenu();

int choice;

std::cin >> choice;

switch (choice) {

case 1: {

int number;

std::string owner;

long int balance, minimumBalance;

std::cout << "Enter account number: ";

std::cin >> number;

std::cout << "Enter owner name: ";

std::cin >> owner;

std::cout << "Enter starting balance: ";

std::cin >> balance;

std::cout << "Enter minimum allowed balance: ";

std::cin >> minimumBalance;

accounts.push\_back(new RegularAccount(number, owner, balance, minimumBalance));

currentAccount = accounts.size() - 1;

break;

}

case 2: {

int number;

std::string owner;

long int balance;

int interestRate;

std::cout << "Enter account number: ";

std::cin >> number;

std::cout << "Enter owner name: ";

std::cin >> owner;

std::cout << "Enter starting balance: ";

std::cin >> balance;

std::cout << "Enter interest rate: ";

std::cin >> interestRate;

accounts.push\_back(new InterestAccount(number, owner, balance, interestRate));

currentAccount = accounts.size() - 1;

break;

}

case 3: {

if (currentAccount < 0) {

std::cout << "No account selected.\n";

break;

}

double amount;

std::cout << "Enter amount to deposit: ";

std::cin >> amount;

accounts[currentAccount]->deposit(amount);

std::cout << "Deposit successful.\n";

break;

}

case 4: {

if (currentAccount < 0) {

std::cout << "No account selected.\n";

break;

}

double amount;

std::cout << "Enter amount to withdraw: ";

std::cin >> amount;

accounts[currentAccount]->withdraw(amount);

std::cout << "Withdrawal attempt completed.\n";

break;

}

case 5: {

int accountIndex;

std::cout << "Enter account index to switch (0 - " << accounts.size() - 1 << "): ";

std::cin >> accountIndex;

if (accountIndex >= 0 && accountIndex < accounts.size()) {

currentAccount = accountIndex;

std::cout << "Switched to account " << currentAccount << ".\n";

} else {

std::cout << "Invalid account index.\n";

}

break;

}

case 6: {

if (currentAccount < 0) {

std::cout << "No account selected.\n";

break;

}

std::cout << "Current balance: " << accounts[currentAccount]->getBalance() << "\n";

break;

}

case 7:

case 8: {

if (currentAccount < 0 || accounts.size() < 2) {

std::cout << "Two accounts are required for this operation.\n";

break;

}

int otherAccount;

std::cout << "Enter index of another account: ";

std::cin >> otherAccount;

if (otherAccount >= 0 && otherAccount < accounts.size() && otherAccount != currentAccount) {

if (choice == 7) {

\*accounts[currentAccount] + \*accounts[otherAccount];

std::cout << "Accounts combined.\n";

} else {

\*accounts[currentAccount] - \*accounts[otherAccount];

std::cout << "Accounts subtracted.\n";

}

} else {

std::cout << "Invalid account selection.\n";

}

break;

}

case 9: {

if (currentAccount < 0) {

std::cout << "No account selected.\n";

break;

}

double amount;

std::cout << "Enter amount to add: ";

std::cin >> amount;

\*accounts[currentAccount] += amount;

std::cout << "Amount added.\n";

break;

}

case 10: {

if (currentAccount < 0) {

std::cout << "No account selected.\n";

break;

}

double amount;

std::cout << "Enter amount to subtract: ";

std::cin >> amount;

\*accounts[currentAccount] -= amount;

std::cout << "Amount subtracted.\n";

break;

}

case 11:

for (BankAccount\* account : accounts) {

delete account;

}

return 0;

default:

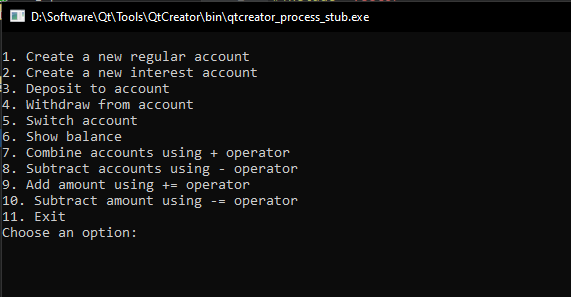
std::cout << "Invalid choice.\n";

}

}

}

*Результат виконання*



**Завдання 2**

*Лістинг blackfly.cpp*

#include "blackfly.h"  
  
BlackFly::BlackFly() : Fly(QPixmap(":/assets/black\_fly.png")) {}

*Лістинг blackfly.h*

#ifndef BLACKFLY\_H  
#define BLACKFLY\_H  
  
#include <QObject>  
#include "fly.h"  
  
class BlackFly : public Fly  
{  
public:  
 BlackFly();  
};  
  
#endif // BLACKFLY\_H

*Лістинг bluefly.cpp*

#include "bluefly.h"  
  
BlueFly::BlueFly() : Fly(QPixmap(":/assets/blue\_fly.png")) {}

*Лістинг bluefly.h*

#ifndef BLUEFLY\_H  
#define BLUEFLY\_H  
  
#include <QObject>  
#include "fly.h"  
  
class BlueFly : public Fly  
{  
public:  
 BlueFly();  
};  
  
#endif // BLUEFLY\_H

*Лістинг classicfly.cpp*

#include "classicfly.h"  
  
ClassicFly::ClassicFly() : Fly(QPixmap(":/assets/fly.png")) {}

*Лістинг classicfly.h*

#ifndef CLASSICFLY\_H  
#define CLASSICFLY\_H  
  
#include <QObject>  
#include "fly.h"  
  
class ClassicFly : public Fly  
{  
public:  
 ClassicFly();  
  
};  
  
#endif // CLASSICFLY\_H

*Лістинг fly.cpp*

#include "fly.h"  
#include "mainwindow.h"  
  
#include <QIcon>  
  
Fly::Fly(QPixmap flyPixmap)  
{  
 setFlyPixmap(flyPixmap);  
}  
  
Fly::~Fly()  
{  
  
}  
  
QPixmap Fly::getFlyPixmap() const  
{  
 return flyPixmap;  
}  
  
void Fly::setFlyPixmap(QPixmap newFlyPixmap)  
{  
 flyPixmap = newFlyPixmap;  
}

*Лістинг fly.h*

#ifndef FLY\_H  
#define FLY\_H  
  
#include <QObject>  
#include <QPushButton>  
  
class Fly  
{  
public:  
 Fly(QPixmap flyPixmap);  
 ~Fly();  
  
 QPixmap getFlyPixmap() const;  
 void setFlyPixmap(QPixmap newFlyPixmap);  
  
private:  
 QPixmap flyPixmap;  
  
};  
  
#endif // FLY\_H

*Лістинг greenfly.cpp*

#include "greenfly.h"  
  
GreenFly::GreenFly() : Fly(QPixmap(":/assets/green\_fly.png")) {}

*Лістинг greenfly.h*

#ifndef GREENFLY\_H  
#define GREENFLY\_H  
  
#include <QObject>  
#include "fly.h"  
  
class GreenFly : public Fly  
{  
public:  
 GreenFly();  
  
};  
  
#endif // GREENFLY\_H

*Лістинг main.cpp*

#include "mainwindow.h"  
  
#include <QApplication>  
  
int main(int argc, char \*argv[])  
{  
 QApplication a(argc, argv);  
 MainWindow w;  
 w.show();  
 return a.exec();  
}

*Лістинг mainwindow.cpp*

#include "mainwindow.h"  
#include "ui\_mainwindow.h"  
  
#include <cmath>  
#include <QRandomGenerator>  
#include <QMessageBox>  
  
#include "classicfly.h"  
#include "greenfly.h"  
#include "blackfly.h"  
#include "bluefly.h"  
  
MainWindow::MainWindow(QWidget \*parent)  
 : QMainWindow(parent)  
 , ui(new Ui::MainWindow)  
{  
 ui->setupUi(this);  
  
 flies.push\_back(ClassicFly());  
 flies.push\_back(GreenFly());  
 flies.push\_back(BlackFly());  
 flies.push\_back(BlueFly());  
  
 flyInitialPos = ui->flyObject->pos();  
 flyLevel = -1;  
 updateFlyLevel();  
  
 QPixmap trapPlantPixmap(":/assets/plant\_trap.png");  
 ui->trapPlantObject->setIcon(QIcon(trapPlantPixmap));  
 ui->trapPlantObject->setIconSize(QSize(256, 256));  
 ui->trapPlantObject->stackUnder(ui->flyObject);  
  
 QPixmap windowPixmap(":/assets/window.png");  
 ui->windowObject->setIcon(QIcon(windowPixmap));  
 ui->windowObject->setIconSize(QSize(128, 128));  
 ui->windowObject->stackUnder(ui->flyObject);  
  
 this->setMouseTracking(true);  
 ui->centralwidget->setMouseTracking(true);  
 ui->flyObject->setMouseTracking(true);  
}  
  
MainWindow::~MainWindow()  
{  
 delete ui;  
}  
  
bool MainWindow::event(QEvent \*event)  
{  
 if (event->type() == QEvent::MouseMove) {  
 QPoint mousePos = ui->centralwidget->mapFromGlobal(QCursor::pos());  
 QPoint flyPos = ui->flyObject->pos();  
  
 double mouseDistance = calculateDistance(flyPos, mousePos);  
 if (mouseDistance <= FLY\_MOVE\_DIST\_CONDITION) {  
 QPoint newFlyPos(0, 0);  
  
 while (true) {  
 int randomX = generateRandomXY();  
 int randomY = generateRandomXY();  
  
 newFlyPos.setX(flyPos.x() + randomX);  
 newFlyPos.setY(flyPos.y() + randomY);  
  
 int windowMoveLimitForX = this->size().width() - ui->flyObject->iconSize().width();  
 if (newFlyPos.x() < 0 || newFlyPos.x() > windowMoveLimitForX)  
 continue;  
  
 int windowMoveLimitForY = this->size().height() - ui->flyObject->iconSize().height();  
 if (newFlyPos.y() < 0 || newFlyPos.y() > windowMoveLimitForY)  
 continue;  
  
 break;  
 }  
  
 ui->flyObject->move(newFlyPos);  
 }  
  
 if (ui->trapPlantObject->geometry().contains(flyPos)) {  
 QMessageBox msgBox;  
 msgBox.setIcon(QMessageBox::Information);  
 msgBox.setText("Муха спіймана, спробуйте ще раз :)");  
 msgBox.exec();  
  
 ui->flyObject->move(flyInitialPos);  
 }  
  
 if (ui->windowObject->geometry().contains(flyPos)) {  
 if (flyLevel + 1 == MAX\_LEVEL) {  
 QMessageBox msgBox;  
 msgBox.setIcon(QMessageBox::Information);  
 msgBox.setText("Вау, вітаю! Ти пройшов гру \"Піймай муху\" 😍");  
 msgBox.exec();  
  
 flyLevel = -1;  
 } else {  
 QMessageBox msgBox;  
 msgBox.setIcon(QMessageBox::Information);  
 msgBox.setText(QString("Побіг вдався 😁\nВи пройшли: %1/%2.").arg(QString::number(flyLevel + 1), QString::number(MAX\_LEVEL)));  
 msgBox.exec();  
 }  
  
 ui->flyObject->move(flyInitialPos);  
 updateFlyLevel();  
 }  
 }  
  
 return QMainWindow::event(event);  
}  
  
int MainWindow::generateRandomXY() {  
 return QRandomGenerator::global()->bounded(-FLY\_MOVE\_DIST\_CONDITION, FLY\_MOVE\_DIST\_CONDITION);  
}  
  
double MainWindow::calculateDistance(QPoint &firstPoint, QPoint &secondPoint) {  
 return sqrt(pow(secondPoint.x() - firstPoint.x(), 2) + pow(secondPoint.y() - firstPoint.y(), 2));  
}  
  
void MainWindow::updateFlyLevel()  
{  
 flyLevel += 1;  
 ui->flyObject->setIcon(QIcon(flies[flyLevel].getFlyPixmap()));  
 ui->flyObject->setIconSize(QSize(64, 64));  
}

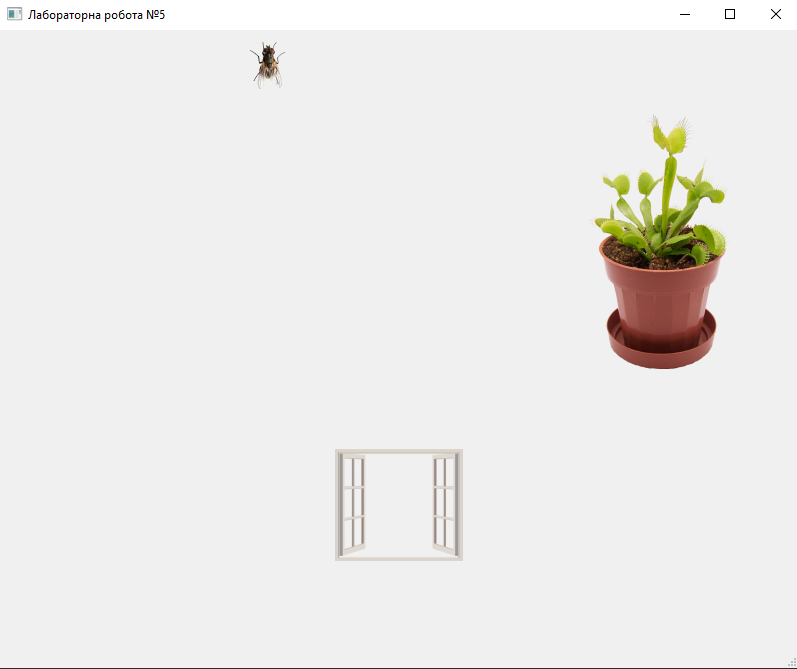
*Лістинг mainwindow.h*

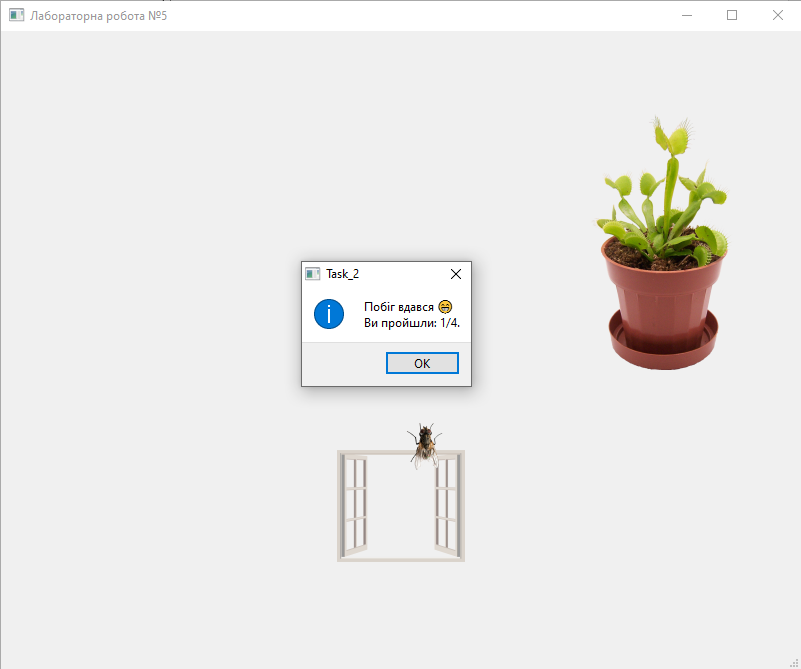
#ifndef MAINWINDOW\_H  
#define MAINWINDOW\_H  
  
#include <QMainWindow>  
#include "fly.h"  
  
#define FLY\_MOVE\_DIST\_CONDITION 70  
  
#define MAX\_LEVEL 4  
  
QT\_BEGIN\_NAMESPACE  
namespace Ui {  
class MainWindow;  
}  
QT\_END\_NAMESPACE  
  
class MainWindow : public QMainWindow  
{  
 Q\_OBJECT  
  
public:  
 MainWindow(QWidget \*parent = nullptr);  
 ~MainWindow();  
  
private:  
 Ui::MainWindow \*ui;  
  
 QPoint flyInitialPos;  
  
 int flyLevel;  
  
 std::vector<Fly> flies;  
  
 bool event(QEvent \* event) override;  
  
 int generateRandomXY();  
 double calculateDistance(QPoint &firstPoint, QPoint &secondPoint);  
  
 void updateFlyLevel();  
  
};  
#endif // MAINWINDOW\_H

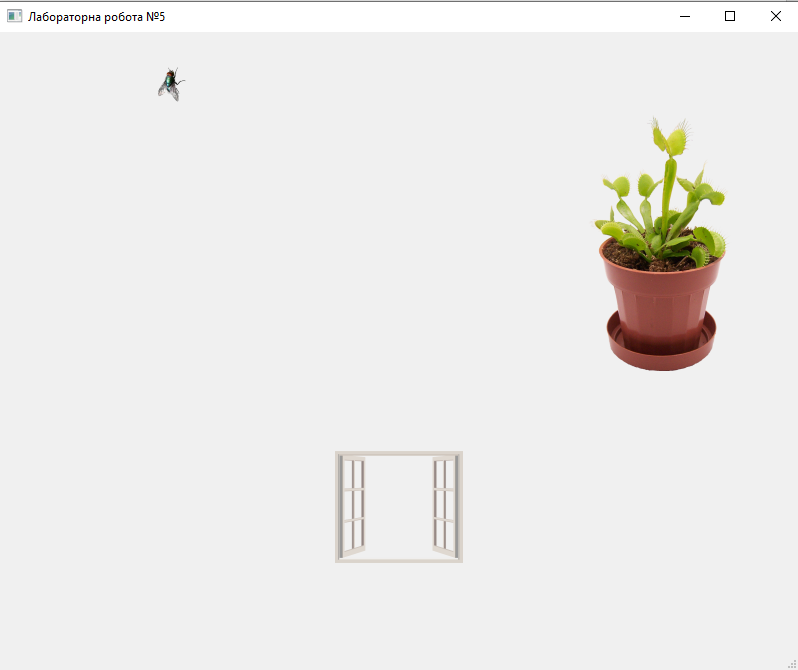
*Лістинг mainwindow.ui*

<?xml version="1.0" encoding="UTF-8"?>  
<ui version="4.0">  
 <class>MainWindow</class>  
 <widget class="QMainWindow" name="MainWindow">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>800</width>  
 <height>638</height>  
 </rect>  
 </property>  
 <property name="windowTitle">  
 <string>Лабораторна робота №5</string>  
 </property>  
 <widget class="QWidget" name="centralwidget">  
 <layout class="QVBoxLayout" name="verticalLayout">  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout\_3">  
 <item>  
 <widget class="QPushButton" name="flyObject">  
 <property name="minimumSize">  
 <size>  
 <width>64</width>  
 <height>64</height>  
 </size>  
 </property>  
 <property name="styleSheet">  
 <string notr="true">background-color: rgba(255, 255, 255, 0);  
border: none;</string>  
 </property>  
 <property name="text">  
 <string/>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <spacer name="horizontalSpacer\_4">  
 <property name="orientation">  
 <enum>Qt::Orientation::Horizontal</enum>  
 </property>  
 <property name="sizeHint" stdset="0">  
 <size>  
 <width>40</width>  
 <height>20</height>  
 </size>  
 </property>  
 </spacer>  
 </item>  
 </layout>  
 </item>  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout">  
 <item>  
 <spacer name="horizontalSpacer">  
 <property name="orientation">  
 <enum>Qt::Orientation::Horizontal</enum>  
 </property>  
 <property name="sizeHint" stdset="0">  
 <size>  
 <width>40</width>  
 <height>20</height>  
 </size>  
 </property>  
 </spacer>  
 </item>  
 <item>  
 <widget class="QPushButton" name="trapPlantObject">  
 <property name="minimumSize">  
 <size>  
 <width>256</width>  
 <height>256</height>  
 </size>  
 </property>  
 <property name="styleSheet">  
 <string notr="true">background-color: rgba(255, 255, 255, 0);  
border: none;</string>  
 </property>  
 <property name="text">  
 <string/>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout\_2">  
 <item>  
 <spacer name="horizontalSpacer\_5">  
 <property name="orientation">  
 <enum>Qt::Orientation::Horizontal</enum>  
 </property>  
 <property name="sizeHint" stdset="0">  
 <size>  
 <width>40</width>  
 <height>20</height>  
 </size>  
 </property>  
 </spacer>  
 </item>  
 <item>  
 <widget class="QPushButton" name="windowObject">  
 <property name="minimumSize">  
 <size>  
 <width>256</width>  
 <height>256</height>  
 </size>  
 </property>  
 <property name="styleSheet">  
 <string notr="true">background-color: rgba(255, 255, 255, 0);  
border: none;</string>  
 </property>  
 <property name="text">  
 <string/>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <spacer name="horizontalSpacer\_2">  
 <property name="orientation">  
 <enum>Qt::Orientation::Horizontal</enum>  
 </property>  
 <property name="sizeHint" stdset="0">  
 <size>  
 <width>40</width>  
 <height>20</height>  
 </size>  
 </property>  
 </spacer>  
 </item>  
 </layout>  
 </item>  
 </layout>  
 </widget>  
 <widget class="QMenuBar" name="menubar">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>800</width>  
 <height>18</height>  
 </rect>  
 </property>  
 </widget>  
 <widget class="QStatusBar" name="statusbar"/>  
 </widget>  
 <resources/>  
 <connections/>  
</ui>

*Результат виконання*







**Завдання 3**

*Лістинг customstring.cpp*

#include "customstring.h"  
  
#include <string.h>  
  
CustomString::CustomString(const char \*content) {  
 contentSize = strlen(content) + 1;  
 this->content = new char[contentSize];  
 strncpy(this->content, content, contentSize - 1);  
 this->content[contentSize - 1] = '\0';  
}  
  
CustomString::~CustomString() {  
 delete[] content;  
}  
  
CustomString CustomString::operator+(const CustomString &other) const {  
 int newSize = contentSize + other.contentSize - 1;  
 char \*newContent = new char[newSize];  
  
 strncpy(newContent, content, contentSize - 1);  
 strncpy(newContent + contentSize - 1, other.content, other.contentSize);  
  
 CustomString result(newContent);  
 delete[] newContent;  
 return result;  
}  
  
bool CustomString::operator==(const CustomString &other) const {  
 return strcmp(content, other.content) == 0;  
}  
  
bool CustomString::operator!=(const CustomString &other) const {  
 return !(\*this == other);  
}  
  
char &CustomString::operator[](int index) {  
 if (index < 0 || index >= contentSize - 1) {  
 throw "Index out of range";  
 }  
  
 return content[index];  
}  
  
std::ostream &operator<<(std::ostream &os, const CustomString &object) {  
 os << object.content;  
 return os;  
}

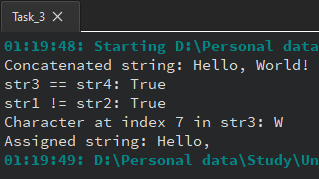
*Лістинг customstring.h*

#ifndef CUSTOMSTRING\_H  
#define CUSTOMSTRING\_H  
  
#include <iostream>  
  
class CustomString  
{  
public:  
 CustomString(const char \*content = "");  
 ~CustomString();  
  
 CustomString operator+(const CustomString &other) const;  
 bool operator==(const CustomString &other) const;  
 bool operator!=(const CustomString &other) const;  
 char &operator[](int index);  
  
 friend std::ostream &operator<<(std::ostream &os, const CustomString &object);  
  
private:  
 char \*content;  
 int contentSize;  
  
 void copyContent(const char \*source);  
  
};  
  
#endif

*Лістинг main.cpp*

#include "customstring.h"  
  
#include <iostream>  
  
int main()  
{  
 CustomString str1("Hello, ");  
 CustomString str2("World!");  
  
 CustomString str3 = str1 + str2;  
 std::cout << "Concatenated string: " << str3 << std::endl;  
  
 CustomString str4("Hello, World!");  
 std::cout << "str3 == str4: " << (str3 == str4 ? "True" : "False") << std::endl;  
 std::cout << "str1 != str2: " << (str1 != str2 ? "True" : "False") << std::endl;  
  
 std::cout << "Character at index 7 in str3: " << str3[7] << std::endl;  
  
 CustomString str5;  
 str5 = str1;  
 std::cout << "Assigned string: " << str5 << std::endl;  
  
 return 0;  
}

*Результат виконання*



**Висновок:**

Виконуючи лабораторну роботу №5, я ознайомився з концепцією перевантаження операторів у мові C++ та закріпив навички їх реалізації. У процесі роботи я перевантажив оператори для задачі з об’єктно-орієнтованого програмування, відокремив муху як окремий клас в грі «Піймай муху», а також розробив клас для роботи з рядками, що включав операції конкатенації, порівняння, доступу до символу за індексом та виведення на екран. Це дозволило поглибити розуміння принципів ООП та перевантаження операторів.