Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

Дисципліна: Об’єктно-орієнтоване програмування

**Лабораторна робота №5**

**Тема:** **«**ПЕРЕВАНТАЖЕННЯ ОПЕРАТОРІВ**»**

|  |
| --- |
| Виконав: ст. гр. КI-23 |
| Чепіль В.О. |
| Перевірив викладач:  Козірова Н. Л. |
|  |

Кропивницький 2024

**Тема:** Перевантаження операторів

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

**Варіант:** 4

**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:**

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.

**Лістинг завдання 1(task1):**

**Polynom.h**

#ifndef POLYNOM\_H

#define POLYNOM\_H

#include <string>

#include <vector>

using namespace std;

class Polynom {

public:

Polynom();

Polynom(const string& expression);

string getCorrectData();

Polynom operator+(const Polynom& other) const; // перевантаження оператора +

Polynom operator-(const Polynom& other) const; // перевантаження оператора -

Polynom operator\*(const Polynom& other) const; // перевантаження оператора \*

string add(const Polynom& other) const;

string subtract(const Polynom& other) const;

string multiply(const Polynom& other) const;

private:

string correctData;

vector<string> xGroup;

vector<string> yGroup;

vector<string> xPowerGroup;

vector<string> yPowerGroup;

vector<string> xyGroup;

vector<string> xyPowerGroup;

vector<string> constantGroup;

string checkData(const string& getData);

void divisionIntoGroups(const string& polynom);

void changeSignInGroups();

void changeSign(string& term);

string processSimpleGroup(const vector<string>& group, char var, char action) const;

string processGroupWithPower(const vector<string>& group, char var1, char var2, char action, bool isPower) const;

string multiplyTerms(const string& term1, const string& term2, char var, bool isPower) const;

int extractCoefficient(const string& term) const;

};

string solutionPolynomials(const Polynom &value1, char action, const Polynom &value2);

#endif // POLYNOM\_H

**Polynom.cpp:**

#include "polynom.h"

#include <iostream>

#include <cctype>

using namespace std;

Polynom::Polynom() {}

Polynom::Polynom(const string& expression) {

correctData = checkData(expression);

if (correctData != "Invalid data!") {

divisionIntoGroups(correctData);

} else {

cout << correctData << endl;

}

}

string Polynom::getCorrectData()

{

return correctData;

}

// Перевантаження оператора +

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

Polynom result;

result.correctData = this->add(other);

return result;

}

// Перевантаження оператора -

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

Polynom result;

result.correctData = this->subtract(other);

return result;

}

// Перевантаження оператора \*

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

Polynom result;

result.correctData = this->multiply(other);

return result;

}

string Polynom::checkData(const string& getData) {

char allowSymbols[] = {'x', 'y', 'z', '-', '+', '\*', '^'};

bool checkSymbol = false;

string correctData;

for (char c : getData) {

if (isdigit(c)) {

checkSymbol = true;

correctData += c;

} else {

checkSymbol = false;

for (char d : allowSymbols) {

if (c == d) {

correctData += c;

checkSymbol = true;

break;

}

}

}

if (!checkSymbol && c != ' ') {

return "Invalid data!";

}

}

return correctData;

}

void Polynom::divisionIntoGroups(const string& polynom) {

string currentTerm;

for (size\_t i = 0; i < polynom.size(); ++i) {

char c = polynom[i];

if (c == '+' || c == '-') {

if (!currentTerm.empty()) {

if (currentTerm.find("xy^") != string::npos) xyPowerGroup.push\_back(currentTerm);

else if (currentTerm.find("xy") != string::npos) xyGroup.push\_back(currentTerm);

else if (currentTerm.find("x^") != string::npos) xPowerGroup.push\_back(currentTerm);

else if (currentTerm.find("y^") != string::npos) yPowerGroup.push\_back(currentTerm);

else if (currentTerm.find("x") != string::npos) xGroup.push\_back(currentTerm);

else if (currentTerm.find("y") != string::npos) yGroup.push\_back(currentTerm);

else constantGroup.push\_back(currentTerm);

}

currentTerm = c;

} else {

currentTerm += c;

}

}

if (!currentTerm.empty()) {

if (currentTerm.find("xy^") != string::npos) xyPowerGroup.push\_back(currentTerm);

else if (currentTerm.find("xy") != string::npos) xyGroup.push\_back(currentTerm);

else if (currentTerm.find("x^") != string::npos) xPowerGroup.push\_back(currentTerm);

else if (currentTerm.find("y^") != string::npos) yPowerGroup.push\_back(currentTerm);

else if (currentTerm.find("x") != string::npos) xGroup.push\_back(currentTerm);

else if (currentTerm.find("y") != string::npos) yGroup.push\_back(currentTerm);

else constantGroup.push\_back(currentTerm);

}

}

void Polynom::changeSignInGroups() {

for (auto& term : xGroup) changeSign(term);

for (auto& term : yGroup) changeSign(term);

for (auto& term : xPowerGroup) changeSign(term);

for (auto& term : yPowerGroup) changeSign(term);

for (auto& term : xyGroup) changeSign(term);

for (auto& term : xyPowerGroup) changeSign(term);

for (auto& term : constantGroup) changeSign(term);

}

void Polynom::changeSign(string& term) {

if (term[0] == '+') term[0] = '-';

else if (term[0] == '-') term[0] = '+';

else term = "-" + term;

}

string Polynom::processSimpleGroup(const vector<string>& group, char var, char action) const {

int sum = 0;

for (const auto& term : group) {

char sign = '+';

int coefficient = 0;

bool hasNumber = false;

for (char ch : term) {

if (isdigit(ch)) {

coefficient = coefficient \* 10 + (ch - '0');

hasNumber = true;

} else if (ch == '+' || ch == '-') {

sign = ch;

}

}

if (!hasNumber) coefficient = 1;

if (action == '+') sum += (sign == '-') ? -coefficient : coefficient;

else if (action == '-') sum += (sign == '-') ? coefficient : -coefficient;

}

string result;

if (sum < 0) result = to\_string(sum) + var + " ";

else if (sum > 0) result = "+" + to\_string(sum) + var + " ";

return result;

}

string Polynom::processGroupWithPower(const vector<string>& group, char var1, char var2, char action, bool isPower) const {

int sum = 0;

for (const auto& term : group) {

char sign = '+';

int coefficient = 0;

bool hasNumber = false;

for (char ch : term) {

if (isdigit(ch)) {

coefficient = coefficient \* 10 + (ch - '0');

hasNumber = true;

} else if (ch == '+' || ch == '-') {

sign = ch;

} else if (ch == '^') break;

}

if (!hasNumber) coefficient = 1;

if (action == '+') sum += (sign == '-') ? -coefficient : coefficient;

else if (action == '-') sum += (sign == '-') ? coefficient : -coefficient;

}

string result;

if (sum != 0) {

if (isPower && var2 != 'q') result = (sum < 0) ? to\_string(sum) + var1 + var2 + "^2 " : "+" + to\_string(sum) + var1 + var2 + "^2 ";

else if (!isPower && var2 != 'q') result = (sum < 0) ? to\_string(sum) + var1 + var2 + " " : "+" + to\_string(sum) + var1 + var2 + " ";

else result = (sum < 0) ? to\_string(sum) + var1 + "^2 " : "+" + to\_string(sum) + var1 + "^2 ";

}

return result;

}

string Polynom::add(const Polynom& other) const {

if (correctData == "Invalid data!") return "Invalid data!";

Polynom resultAdd;

resultAdd.divisionIntoGroups(this->correctData);

resultAdd.divisionIntoGroups(other.correctData);

string resultString;

int sumConstant = 0;

for (const string& term : resultAdd.constantGroup) sumConstant += stoi(term);

if (sumConstant != 0) resultString += (sumConstant < 0) ? to\_string(sumConstant) + " " : "+" + to\_string(sumConstant) + " ";

resultString += processSimpleGroup(resultAdd.xGroup, 'x', '+');

resultString += processSimpleGroup(resultAdd.yGroup, 'y', '+');

resultString += processGroupWithPower(resultAdd.xPowerGroup, 'x', 'q', '+', true);

resultString += processGroupWithPower(resultAdd.yPowerGroup, 'y', 'q', '+', true);

resultString += processGroupWithPower(resultAdd.xyGroup, 'x', 'y', '+', false);

resultString += processGroupWithPower(resultAdd.xyPowerGroup, 'x', 'y', '+', true);

if (resultString.back() == ' ') resultString.pop\_back();

return resultString;

}

string Polynom::subtract(const Polynom& other) const {

Polynom subtract, resultSubtract;

subtract.divisionIntoGroups(other.correctData);

subtract.changeSignInGroups();

resultSubtract.divisionIntoGroups(this->correctData);

resultSubtract.constantGroup.insert(resultSubtract.constantGroup.end(), subtract.constantGroup.begin(), subtract.constantGroup.end());

resultSubtract.xGroup.insert(resultSubtract.xGroup.end(), subtract.xGroup.begin(), subtract.xGroup.end());

resultSubtract.yGroup.insert(resultSubtract.yGroup.end(), subtract.yGroup.begin(), subtract.yGroup.end());

resultSubtract.xPowerGroup.insert(resultSubtract.xPowerGroup.end(), subtract.xPowerGroup.begin(), subtract.xPowerGroup.end());

resultSubtract.yPowerGroup.insert(resultSubtract.yPowerGroup.end(), subtract.yPowerGroup.begin(), subtract.yPowerGroup.end());

resultSubtract.xyGroup.insert(resultSubtract.xyGroup.end(), subtract.xyGroup.begin(), subtract.xyGroup.end());

resultSubtract.xyPowerGroup.insert(resultSubtract.xyPowerGroup.end(), subtract.xyPowerGroup.begin(), subtract.xyPowerGroup.end());

string resultString;

int sumConstant = 0;

for (const string& term : resultSubtract.constantGroup) sumConstant += stoi(term);

if (sumConstant != 0) resultString += (sumConstant < 0) ? to\_string(sumConstant) + " " : "+" + to\_string(sumConstant) + " ";

resultString += processSimpleGroup(resultSubtract.xGroup, 'x', '+');

resultString += processSimpleGroup(resultSubtract.yGroup, 'y', '+');

resultString += processGroupWithPower(resultSubtract.xPowerGroup, 'x', 'q', '+', true);

resultString += processGroupWithPower(resultSubtract.yPowerGroup, 'y', 'q', '+', true);

resultString += processGroupWithPower(resultSubtract.xyGroup, 'x', 'y', '+', false);

resultString += processGroupWithPower(resultSubtract.xyPowerGroup, 'x', 'y', '+', true);

if (resultString.back() == ' ') resultString.pop\_back();

return resultString;

}

string Polynom::multiply(const Polynom& other) const {

Polynom result;

for (const auto& term1 : xGroup) {

for (const auto& term2 : other.xGroup) result.xPowerGroup.push\_back(multiplyTerms(term1, term2, 'x', true));

}

for (const auto& term1 : xGroup) {

for (const auto& term2 : other.constantGroup) result.xGroup.push\_back(multiplyTerms(term1, term2, 'x', false));

}

for (const auto& term1 : constantGroup) {

for (const auto& term2 : other.constantGroup) result.constantGroup.push\_back(multiplyTerms(term1, term2, ' ', false));

}

string resultString;

resultString += processGroupWithPower(result.xPowerGroup, 'x', 'q', '+', true);

resultString += processSimpleGroup(result.xGroup, 'x', '+');

resultString += processSimpleGroup(result.constantGroup, ' ', '+');

if (resultString.back() == ' ') resultString.pop\_back();

return resultString;

}

string Polynom::multiplyTerms(const string& term1, const string& term2, char var, bool isPower) const {

int coef1 = extractCoefficient(term1);

int coef2 = extractCoefficient(term2);

int resultCoef = coef1 \* coef2;

if (isPower) return to\_string(resultCoef) + var + "^2";

else if (var == ' ') return to\_string(resultCoef);

else return to\_string(resultCoef) + var;

}

int Polynom::extractCoefficient(const string& term) const {

int coefficient = 0;

bool negative = false;

for (char ch : term) {

if (ch == '-') negative = true;

else if (isdigit(ch)) coefficient = coefficient \* 10 + (ch - '0');

}

coefficient = coefficient == 0 ? 1 : coefficient;

return negative ? -coefficient : coefficient;

}

string solutionPolynomials(const Polynom& value1, char action, const Polynom& value2) {

if (action == '+') return value1.add(value2);

else if (action == '-') return value1.subtract(value2);

else if (action == '\*') return value1.multiply(value2);

return "Invalid operation!";

}

**main.cpp:**

#include "polynom.h"

#include <iostream>

int main() {

Polynom polynom1("-3x^2 + 3xy^2 - 30x - 40 - 5y - 3y^2 - 15xy");

Polynom polynom2("x^2 - 4x^2 + 5x - 5 - 35 + y + 2y^2 + 10xy - 5xy^2");

Polynom polynom3("2x -4");

Polynom polynom4("x + 1");

string resultAdd = solutionPolynomials(polynom1, '+', polynom2);

cout << "result add : " << resultAdd << endl;

string resultMultiply = solutionPolynomials(polynom3, '\*', polynom4);

cout << "result multiply : " << resultMultiply << endl;

string resultSubtract = solutionPolynomials(polynom3, '-', polynom4);

cout << "result subtruct : " << resultSubtract << endl;

// За допомогою перевантаження операторів

cout << endl << "Using operator overloading" << endl;

Polynom result;

result = polynom1 + polynom2;

cout << "result add operation: " << result.getCorrectData() << endl;

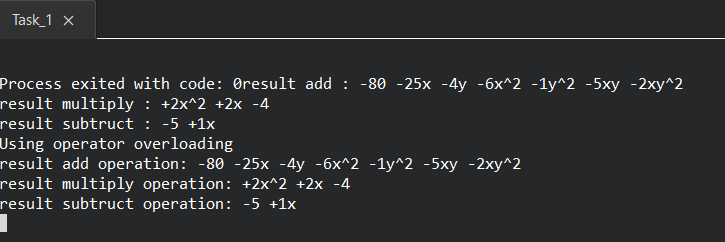
result = polynom3 \* polynom4;

cout << "result multiply operation: " << result.getCorrectData() << endl;

result = polynom3 - polynom4;

cout << "result subtruct operation: " << result.getCorrectData() << endl;

return 0;

} 

**Лістинг завдання 2(task2):**

**Fly.h**

#ifndef FLY\_H

#define FLY\_H

#include <QLabel>

#include <QCursor>

#include <QPixmap>

#include <QPoint>

#include <QRandomGenerator>

class Fly : public QLabel {

public:

Fly(QWidget\* parent = nullptr);

virtual void moveRandomly();

void moveFly();

};

#endif // FLY\_H

**Fly.cpp**

#include "fly.h"

Fly::Fly(QWidget\* parent) : QLabel(parent) {

setPixmap(QPixmap(":/pictures/fly.png").scaled(24, 24));

setGeometry(100, 100, 24, 24);

}

void Fly::moveRandomly() {

int X = QRandomGenerator::global()->bounded(-5, 5);

int Y = QRandomGenerator::global()->bounded(-5, 5);

int newX = qBound(0, x() + X, parentWidget()->width() - width());

int newY = qBound(0, y() + Y, parentWidget()->height() - height());

move(newX, newY);

}

void Fly::moveFly()

{

QPoint cursorPos = parentWidget()->mapFromGlobal(QCursor::pos());

int mouseX = cursorPos.x();

int mouseY = cursorPos.y();

int flyX = this->x();

int flyY = this->y();

int distanceX = mouseX - flyX;

int distanceY = mouseY - flyY;

int distanceSquared = distanceX \* distanceX + distanceY \* distanceY;

if (distanceSquared < 2500)

{

flyX += (distanceX < 0) ? 10 : -10;

flyY += (distanceY < 0) ? 10 : -10;

}

else

{

this->moveRandomly();

return;

}

flyX = qBound(0, flyX, parentWidget()->width() - width());

flyY = qBound(0, flyY, parentWidget()->height() - height());

this->move(flyX, flyY);

}

**Game.h**

#ifndef GAME\_H

#define GAME\_H

#include <QMainWindow>

#include "level.h"

#include <QPixmap>

class Game : public QMainWindow {

Q\_OBJECT

public:

Game(QWidget\* parent = nullptr);

private slots:

void onLevelCompleted();

void gameOver();

private:

int currentLevelNumber;

Level\* currentLevel = nullptr;

QCursor cursor;

void startLevel(int levelNumber);

};

#endif // GAME\_H

**Game.cpp**

#include "game.h"

#include <QMessageBox>

#include "QApplication"

Game::Game(QWidget\* parent) : QMainWindow(parent), currentLevelNumber(1) {

QPixmap cursorImage = QPixmap(":pictures/flySwatter.png");

cursor = QCursor(cursorImage);

setCursor(cursor);

startLevel(currentLevelNumber);

}

void Game::startLevel(int levelNumber) {

if (currentLevel)

{

currentLevel->deleteLater();

}

currentLevel = new Level(levelNumber, this);

setCentralWidget(currentLevel);

setFixedSize(currentLevel->size());

connect(currentLevel, &Level::levelCompleted, this, &Game::onLevelCompleted);

connect(currentLevel, &Level::levelFailed, this, &Game::gameOver);

currentLevel->show();

}

void Game::onLevelCompleted()

{

if (currentLevelNumber > 1)

{

QMessageBox::information(this, "Рівень пройдено", "Ви пройшли гру");

QApplication::quit();

}

else

{

QMessageBox::information(this, "Рівень пройдено", "Ви зловили муху! Переходьте на наступний рівень.");

}

currentLevelNumber++;

startLevel(currentLevelNumber);

}

void Game::gameOver()

{

QMessageBox::information(this, "Рівень провалено", "Муха вилетіла у вікно");

QApplication::quit();

}

**Level.h**

#ifndef LEVEL\_H

#define LEVEL\_H

#include <QWidget>

#include <QLabel>

#include <QTimer>

#include "fly.h"

#include "smartfly.h"

class Level : public QWidget {

Q\_OBJECT

public:

Level(int levelNumber, QWidget\* parent = nullptr);

bool checkIfFlyCaught();

void stopLevel();

signals:

void levelCompleted();

void levelFailed();

private slots:

void updateFliesPosition();

private:

int level;

QLabel\* windowLabel = nullptr;

QLabel\* trapLabel;

Fly\* fly1;

SmartFly\* fly2 = nullptr;

QTimer\* timer;

};

#endif // LEVEL\_H

**Level.cpp**

#include "level.h"

#include <QMessageBox>

#include <QPixmap>

Level::Level(int levelNumber, QWidget\* parent) : QWidget(parent), level(levelNumber)

{

setFixedSize(1000, 600);

if (levelNumber > 1) {

windowLabel = new QLabel(this);

windowLabel->setGeometry(500, 500, 80, 80);

QPixmap windowPixmap(":/pictures/window.png");

windowLabel->setPixmap(windowPixmap.scaled(80, 80, Qt::KeepAspectRatio));

}

trapLabel = new QLabel(this);

trapLabel->setGeometry(600, 100, 80, 80);

QPixmap trapPixmap(":/pictures/trap.png");

trapLabel->setPixmap(trapPixmap.scaled(80, 80, Qt::KeepAspectRatio));

fly1 = new Fly(this);

if (levelNumber > 1)

{

delete fly1;

fly1 = nullptr;

fly2 = new SmartFly(this, windowLabel);

}

timer = new QTimer(this);

connect(timer, &QTimer::timeout, this, &Level::updateFliesPosition);

timer->start(50);

}

bool Level::checkIfFlyCaught()

{

return (fly1 && fly1->geometry().intersects(trapLabel->geometry())) ||

(fly2 && fly2->geometry().intersects(trapLabel->geometry()));

}

void Level::stopLevel()

{

timer->stop();

if (fly1)

{

delete fly1;

fly1 = nullptr;

}

else if (fly2)

{

delete fly2;

fly2 = nullptr;

}

}

void Level::updateFliesPosition()

{

QPoint cursorPos = parentWidget()->mapFromGlobal(QCursor::pos());

QRect gameRect = parentWidget()->rect();

if (!gameRect.contains(cursorPos))

{

if (fly1)

{

fly1->moveRandomly();

}

else if (fly2)

{

fly2->moveFly();

}

return;

}

if (fly2 && fly2->geometry().intersects(windowLabel->geometry()))

{

emit levelFailed();

stopLevel();

return;

}

if (fly1)

{

fly1->moveFly();

}

else if (fly2)

{

fly2->moveFly();

}

if (checkIfFlyCaught())

{

emit levelCompleted();

stopLevel();

}

}

**SmartFly.h**

#ifndef SMARTFLY\_H

#define SMARTFLY\_H

#include "fly.h"

class SmartFly : public Fly

{

public:

SmartFly(QWidget\* parent, QLabel\* target);

void moveRandomly() override;

void moveFly();

private:

QLabel\* windowLabel;

};

#endif // SMARTFLY\_H

**SmartFly.cpp**

#include "smartfly.h"

SmartFly::SmartFly(QWidget\* parent, QLabel\* target) : Fly(parent), windowLabel(target) {}

void SmartFly::moveRandomly() {

int X = QRandomGenerator::global()->bounded(-5, 5);

int Y = QRandomGenerator::global()->bounded(-5, 5);

int newX = qBound(0, x() + X, parentWidget()->width() - width());

int newY = qBound(0, y() + Y, parentWidget()->height() - height());

move(newX, newY);

}

void SmartFly::moveFly()

{

QPoint windowPos = parentWidget()->mapFromGlobal(windowLabel->mapToGlobal(windowLabel->rect().center()));

QPoint cursorPos = parentWidget()->mapFromGlobal(QCursor::pos());

int windowX = windowPos.x();

int windowY = windowPos.y();

int mouseX = cursorPos.x();

int mouseY = cursorPos.y();

int flyX = this->x();

int flyY = this->y();

int distanceWindowX = windowX - flyX;

int distanceWindowY = windowY - flyY;

int distanceCursorX = mouseX - flyX;

int distanceCursorY = mouseY - flyY;

int distanceSquared = distanceCursorX \* distanceCursorX + distanceCursorY \* distanceCursorY;

if (distanceSquared < 2500)

{

flyX += (distanceCursorX < 0) ? 5 : -5;

flyY += (distanceCursorY < 0) ? 5 : -5;

}

else

{

flyX += (distanceWindowX > 0) ? 5 : -5;

flyY += (distanceWindowY > 0) ? 5 : -5;

}

flyX = qBound(0, flyX, parentWidget()->width() - width());

flyY = qBound(0, flyY, parentWidget()->height() - height());

this->move(flyX, flyY);

this->moveRandomly();

}

**main.cpp**

#include <QApplication>

#include "game.h"

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

QApplication app(argc, argv);

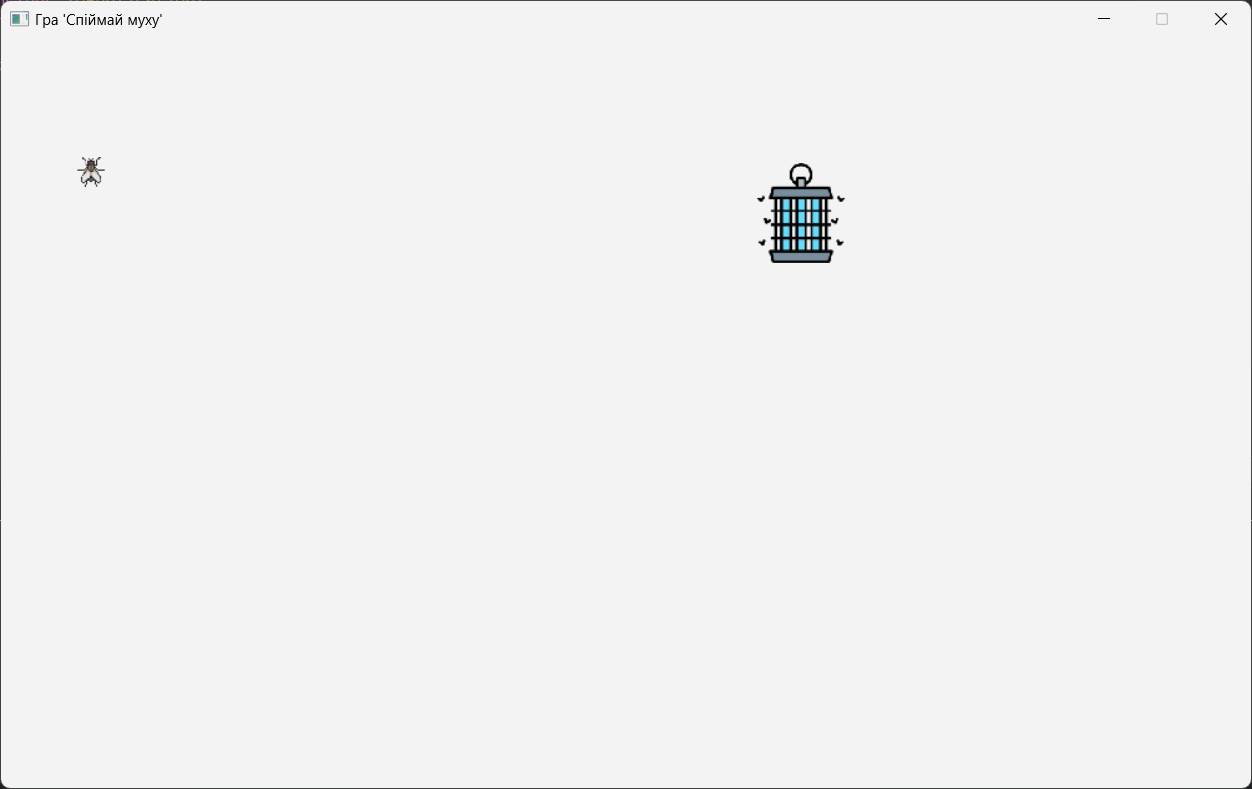
Game game;

game.setWindowTitle("Гра 'Спіймай муху'");

game.show();

return app.exec();

}



**Лістинг завдання 3(task3):**

**String.h**

#ifndef STRING\_H

#define STRING\_H

#include <iostream>

using namespace std;

class String

{

private:

char\* str;

unsigned int length;

unsigned int my\_strlen(const char\* s) const;

void my\_strcpy(char\* dest, const char\* src) const;

void my\_strcat(char\* dest, const char\* src) const;

int my\_strcmp(const char\* s1, const char\* s2) const;

public:

String();

String(const char\* s);

String(const String& other);

~String();

// Оператори

String& operator=(const String& other); // оператор присвоєння

String operator+(const String& other) const; // оператор +

bool operator==(const String& other) const; // оператор порівняння ==

bool operator!=(const String& other) const; // оператор порівняння !=

char& operator[](unsigned int index); // оператор індексації

friend ostream& operator<<(ostream& os, const String& s); // оператор виведення <<

void concat(const String& other);

void remove(unsigned int pos);

int compare(const String& other) const;

void print() const;

unsigned int getLength() const;

static void sortStrings(String arr[], unsigned int size);

};

#endif // STRING\_H

**String.cpp**

#include "String.h"

#include <iostream>

unsigned int String::my\_strlen(const char\* s) const {

if (!s) return 0;

unsigned int len = 0;

while (s[len] != '\0') {

++len;

}

return len;

}

void String::my\_strcpy(char\* dest, const char\* src) const {

if (!dest || !src) return;

while (\*src != '\0') {

\*dest++ = \*src++;

}

\*dest = '\0';

}

void String::my\_strcat(char\* dest, const char\* src) const {

if (!dest || !src) return;

while (\*dest != '\0') {

dest++;

}

while (\*src != '\0') {

\*dest++ = \*src++;

}

\*dest = '\0';

}

int String::my\_strcmp(const char\* s1, const char\* s2) const {

if (!s1 || !s2) return -1;

while (\*s1 && (\*s1 == \*s2)) {

s1++;

s2++;

}

return \*(unsigned char\*)s1 - \*(unsigned char\*)s2;

}

String::String() : str(nullptr), length(0) {}

String::String(const char\* s) {

if (s) {

length = my\_strlen(s);

str = new char[length + 1];

my\_strcpy(str, s);

} else {

str = nullptr;

length = 0;

}

}

String::String(const String& other) {

if (other.str) {

length = other.length;

str = new char[length + 1];

my\_strcpy(str, other.str);

} else {

str = nullptr;

length = 0;

}

}

String::~String() {

delete[] str;

}

String& String::operator=(const String& other) {

if (this == &other) return \*this;

delete[] str;

length = other.length;

str = new char[length + 1];

my\_strcpy(str, other.str);

return \*this;

}

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

String result;

result.length = length + other.length;

result.str = new char[result.length + 1];

my\_strcpy(result.str, str);

my\_strcat(result.str, other.str);

return result;

}

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

return my\_strcmp(str, other.str) == 0;

}

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

return !(\*this == other);

}

char& String::operator[](unsigned int index) {

return str[index];

}

ostream& operator<<(ostream& os, const String& s) {

if (s.str) {

os << s.str;

} else {

os << "Empty string";

}

return os;

}

void String::concat(const String& other) {

if (other.str == nullptr) return;

unsigned int newLength = length + other.length;

char\* newStr = new char[newLength + 1];

if (str != nullptr) {

my\_strcpy(newStr, str);

}

my\_strcat(newStr, other.str);

delete[] str;

str = newStr;

length = newLength;

}

void String::remove(unsigned int pos) {

if (pos >= length || str == nullptr) return;

for (unsigned int i = pos; i < length - 1; ++i) {

str[i] = str[i + 1];

}

str[length - 1] = '\0';

--length;

}

int String::compare(const String& other) const {

return my\_strcmp(str, other.str);

}

void String::print() const {

if (str) {

cout << str;

} else {

cout << "Empty string";

}

}

unsigned int String::getLength() const {

return length;

}

void String::sortStrings(String arr[], unsigned int size) {

for (unsigned int i = 0; i < size - 1; ++i) {

unsigned int minIndex = i;

for (unsigned int j = i + 1; j < size; ++j) {

if (arr[j].compare(arr[minIndex]) < 0) {

minIndex = j;

}

}

if (minIndex != i) {

String temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

}

**main.cpp**

#include "String.h"

#include <iostream>

using namespace std;

int main()

{

String greeting("Hello");

String name("Taras");

String exclamation("!");

// Демонстрація оператора +

String message = greeting + ", " + name + exclamation;

cout << "Concatenated message: " << message << endl;

// Демонстрація оператора порівняння ==

String anotherGreeting("Hello");

if (greeting == anotherGreeting)

{

cout << "Greetings are equal." << endl;

}

// Демонстрація оператора порівняння !=

if (name != greeting)

{

cout << "Name and greeting are different." << endl;

}

// Демонстрація оператора індексації []

cout << "First character of greeting: " << greeting[0] << endl;

// Заміна символу

name[0] = 'a';

cout << "Modified name: " << name << endl;

// Видалення символу з рядка

String sentence("Hello, world!");

sentence.remove(5); // Видалення коми

cout << "Sentence without comma: " << sentence << endl;

// Конкатенація

sentence.concat(String(" How are you?"));

cout << "Extended sentence: " << sentence << endl;

// Масив рядків

String words[] = { String("Banana"), String("Apple"), String("Cherry"), String("Date") };

unsigned int size = 4;

cout << "Unsorted words: ";

for (unsigned int i = 0; i < size; i++)

{

cout << words[i] << " ";

}

cout << endl;

// Сортування масиву рядків

String::sortStrings(words, size);

cout << "Sorted words: ";

for (unsigned int i = 0; i < size; i++)

{

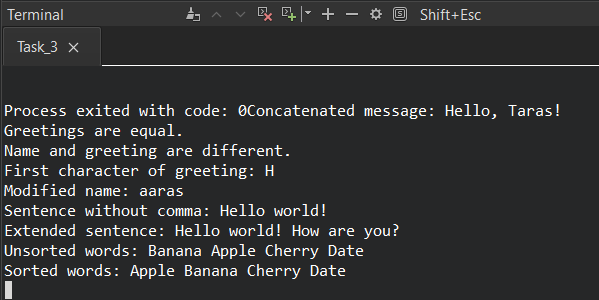
cout << words[i] << " ";

}

cout << endl;

return 0;

}



**Висновок:**

У лабораторній роботі було розглянуто концепцію перевантаження операторів у мові C++. Перевантаження дозволяє змінювати стандартну поведінку операторів для об'єктів класів, що дає можливість створювати зручніші і логічніші операції з власними типами даних. Оператори можуть бути перевантажені як функції-члени класу або як зовнішні функції. Важливо пам'ятати, що не всі оператори можна перевантажити, наприклад, крапка або квадратні дужки. Залишаючи логіку операцій зрозумілою, перевантажені оператори можуть зробити код більш інтуїтивно зрозумілим.