Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №5 з дисципліни

«Основи програмування-2.

Модульне програмування»

«Успадкування та поліморфізм»

Варіант 19

Виконав студент: ІП-11 Лисенко Андрій Юрійович

Перевірила: Вітковська Ірина Іванівна

Київ 2021

**Лабораторна робота №5**

**Успадкування та поліморфізм**

**Мета роботи**

Вивчити механізми створення і використання класів та об’єктів.

**Індивідуальне завдання**

**Варіант 19**

Створити клас TMatrix, який представляє матрицію і містить методи збільшення / зменшення всіх її елементів на вказану величину та обчислення їх середнього арифметичного. Реалізувати класи-нащадки, що представляють матриці з елементами цілого та дійсного типів. Випадковим чином створити по m матриць кожного виду. Елементи цілочисельних матриць збільшити на 9, а дійсних зменшити на 5. Знайти матрицю, середнє арифметичне елементів якої є найменшим.

**Постановка задачі**

Створимо базовий клас TMatrix. Цей клас матиме методи для збільшення / зменшення елементів, виведення на екран та знаходження середнього арифметичного. У головній програмі не будуть створюватися об’єкти цього класу, тому конструктори винесемо у класи-нащадки – IntMatrix та FloatMatrix. Таким чином, ці класи матимуть весь потрібний функціонал для роботи програми. Створимо віртуальну функцію fillMatrix у базовому класі та перевизначимо її у класах-нащадках.

**Побудова математичної моделі**

Складемо таблицю змінних

|  |  |  |  |
| --- | --- | --- | --- |
| Змінна | Тип | Ім’я | Призначення |
| Розмір масиву | *Int* | *SIZE* | Зберігати розмір масиву матриць |
| Кількість рядків та колонків | *Int* | *ROWS, COLS* | Зберігати розмірність матриць |
| елементи матриці | *T\*\** | *Matrix* | зберігати елементи матриці |
| об'єкти класу матриць | *IntMatrix[], FloatMatrix[]* | *Ints, floats* | зберігатимуть атрибути та методи цього класу |
| Мінімальне середнє | *Float* | *Min\_average* | Значення мінімального значення серед усіх середніх арифметичних елементів матриць |
| Індекс мінімального середнього | *Int* | *Min\_idx* | Індекс мінімального середнього арифметичного |

**Псевдокод**

**Основна програма**

ints, floats = array of FloatMatrix and IntMatrix

for int\_matrix, float\_matrix in ints and floats:

int\_matrix.increase(5)

float\_matrix.decrease(9)

averages = matrix.get\_average for all matrices ints and floats

min\_average, min\_idx = min(averages)

output min\_average

output ints[min\_idx] if min\_average is int else output floats[min\_idx]

**Конструктор *FloatMatrix***

for row from 0 to size:

for col from 0 to size

matrix[row][col] = random float from 0 to 10

**Конструктор *IntMatrix***

for row from 0 to size:

for col from 0 to size

matrix[row][col] = random int from 0 to 10

**Метод Matrix get\_average**

Counter = 0

for element in matrix:

counter += element

return counter / (rows \* cols)

**Метод *Matrix print***

for row from 0 to size:

for col from 0 to size:

output matrix[row][col]

**Метод *Matrix increase (value)***

For element in matrix:

Element += value

**Метод *Matrix decrease (value)***

For element in matrix:

Element -= value

**Код на С++**

**main.cpp**

#include "IntMatrix.h"

#include "FloatMatrix.h"

int main() {

srand(time(nullptr)); // NOLINT(cert-msc51-cpp)

const int SIZE = 5;

const int ROWS = 4;

const int COLS = 4;

const int DECREASE\_VALUE = 5;

const int INCREASE\_VALUE = 9;

IntMatrix ints[SIZE];

FloatMatrix floats[SIZE];

cout << "Matrices:\n";

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

cout << "\n" << i + 1 << " (int):\n";

ints[i] = IntMatrix(ROWS, COLS);

ints[i].fillMatrix();

ints[i].increase(INCREASE\_VALUE);

ints[i].print();

cout << "\n" << i + 1<< " (float):\n";

floats[i] = FloatMatrix(ROWS, COLS);

floats[i].fillMatrix();

floats[i].decrease(DECREASE\_VALUE);

floats[i].print();

}

double minAverage = ROWS \* COLS \* 10;

int indexMin = 0;

bool isMinInt = false;

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

if (ints[i].getAverage() < minAverage) {

minAverage = ints[i].getAverage();

isMinInt = true;

indexMin = i;

}

if (floats[i].getAverage() < minAverage) {

minAverage = floats[i].getAverage();

isMinInt = false;

indexMin = i;

}

}

cout << "\nMin value: " << minAverage << ". Matrix (" << indexMin + 1 << "):\n";

if (isMinInt)

ints[indexMin].print();

else

floats[indexMin].print();

}

**TMatrix.h**

#pragma once

#include <iostream>

#include <iomanip>

using std::cout;

template <typename T>

class TMatrix {

public:

TMatrix() = default;

TMatrix(int n, int m): n(n), m(m) {

matrix = new T\* [n];

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

matrix[i] = new T [m];

}

virtual void fillMatrix() {};

T getAverage() {

T counter = 0;

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

for (int j = 0; j < m; j++)

counter += matrix[i][j];

return counter / (n \* m);

}

void print() {

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

for (int j = 0; j < m; j++)

cout << std::setw(5) << std::setprecision(2) << matrix[i][j];

cout << "\n";

}

}

void increase(T value) {

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

for (int j = 0; j < m; j++)

matrix[i][j] += value;

}

void decrease(T value) {

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

for (int j = 0; j < m; j++)

matrix[i][j] -= value;

}

protected:

T \*\*matrix;

int n{};

int m{};

};

**IntMatrix.h**

#pragma once

#include "TMatrix.h"

class IntMatrix : public TMatrix<int> {

public:

IntMatrix(int n, int m) : TMatrix<int>(n, m) {}

void fillMatrix() override {

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

for (int j = 0; j < m; ++j)

matrix[i][j] = rand() % 10; // NOLINT(cert-msc50-cpp)

}

IntMatrix() : TMatrix<int>() {}

};

**FloatMatrix.h**

#pragma once

#include "TMatrix.h"

class FloatMatrix : public TMatrix<float> {

public:

FloatMatrix(int n, int m) : TMatrix<float>(n, m) {}

void fillMatrix() override {

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

for (int j = 0; j < m; ++j)

matrix[i][j] = float(rand() % 10) + float(rand() % 10) / 10; // NOLINT(cert-msc50-cpp)

}

FloatMatrix() : TMatrix<float>() {}

};

**Висновок**

Підчас виконання цієї роботи, я розробив практичні навички роботи з класами та їх успадкуванням, а саме створення окремих класів для спеціалізації базового класу Matrix – FloatMatrix та IntMatrix. Я використував відкрите успадкування тому, що класи-нащадки потребують доступу до полів та методів базового класу. Також я використав поліморфізм для заповнення матриць класів FloatMatrix та IntMatrix*.*