Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра ІПІ

Звіт

з лабораторної роботи № 5 з дисципліни
«Основи програмування 2. Модульне програмування»

„ Успадкування та поліморфізм”

Виконав *ІП-Левченко Владислав В’ячеславович*
(шифр, прізвище, ім'я, по батькові)

Перевірила *Вечерковська Анастасія Сергіївна*
(прізвище, ім'я, по батькові)

Київ 2022

Код c++

parentclass.h

```
#pragma once

#include <ctime>
#include <stdlib.h>
#include <iostream>
#include <iomanip>

class TMatrix {
protected:
	int row;
	int col;
public:
	TMatrix(int, int);
	virtual void valueUp() = 0;
	virtual void valueDown() = 0;
	virtual float countAverage() = 0;
	virtual void setValue(int, int) = 0;
};
```

parentclass.cpp

```
#include "parentclass.h"

TMatrix::TMatrix(int Row, int Col)
{
	this->row = Row;
	this->col = Col;
}
```

subclasses.h

```
#pragma once

#include "parentclass.h"

class intMatrix : public TMatrix {

private:
	int** iMat;
public:
	intMatrix(int, int);
	intMatrix(const intMatrix&);
	void valueUp() override;
	void valueDown() override;
	float countAverage() override;
```

```
        void setValue(int, int) override;
        void printMatrix();
        ~intMatrix();
};

class floatMatrix : public TMatrix {
private:
        float** fMat;
public:
        floatMatrix(int, int);
        floatMatrix(const floatMatrix&);
        void valueDown() override;
        void valueUp() override;
        float countAverage() override;
        void setValue(int, int) override;
        void printMatrix();
        ~floatMatrix();
};
```

subclasses.cpp

```
#include "subclasses.h"

/*Методи класу intMatrix*/

intMatrix::intMatrix(int Row, int Col) : TMatrix(Row, Col)
{
        iMat = new int* [row];
        for (int i = 0; i < row; i++)
        {
                iMat[i] = new int[col];
                for (int j = 0; j < col; j++)
                {
                        iMat[i][j] = 0;
                }
        }
}

intMatrix::intMatrix(const intMatrix& copy) : TMatrix(copy.row,
copy.col)
{
        iMat = new int * [row];
        for (int i = 0; i < row; i++)
        {
                iMat[i] = new int[col];
                for (int j = 0; j < col; j++)
                {
                        iMat[i][j] = copy.iMat[i][j];
                }
        }
}

intMatrix::~intMatrix()
```

```
{
	for (int i = 0; i < row; i++)
	{
		delete[] iMat[i];
	}
	delete[] iMat;
}

void intMatrix::valueUp()
{
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++)
		{
			iMat[i][j] += 9;
		}
	}
}

void intMatrix::valueDown()
{
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++)
		{
			iMat[i][j] -= 5;
		}
	}
}

float intMatrix::countAverage()
{
	float acc = 0;
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++)
		{
			acc += iMat[i][j];
		}
	}
	acc /= row * col;
	return acc;
}

void intMatrix::setValue(int min, int max)
{
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++)
		{
			iMat[i][j] = int(((float(rand()) /
float(RAND_MAX)) * (max - min + 1)) + min);
		}
```

```
	}
}

void intMatrix::printMatrix()
{
	std::cout << "\n";
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++)
		{
			std::cout << std::setw(5) << iMat[i][j];
		}
		std::cout << "\n";
	}
}

/*Методи класу floatMatrix*/

floatMatrix::floatMatrix(int Row, int Col) : TMatrix(Row, Col)
{
	fMat = new float* [row];
	for (int i = 0; i < row; i++)
	{
		fMat[i] = new float[col];
	}
}

floatMatrix::floatMatrix(const floatMatrix& copy) :
TMatrix(copy.row, copy.col)
{
		fMat = new float* [row];
		for (int i = 0; i < row; i++) {
			fMat[i] = new float[col];
			for (int j = 0; j < col; j++) {
				fMat[i][j] = copy.fMat[i][j];
			}
		}
}

floatMatrix::~floatMatrix()
{
	for (int i = 0; i < row; i++)
	{
		delete[] fMat[i];
	}
	delete[] fMat;
}

void floatMatrix::valueDown()
{
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++)
```

```
			{
				fMat[i][j] -= 5;
			}
		}
}

void floatMatrix::valueUp()
{
		for (int i = 0; i < row; i++)
		{
			for (int j = 0; j < col; j++)
			{
				fMat[i][j] += 9;
			}
		}
}

float floatMatrix::countAverage()
{
		float acc = 0;
		for (int i = 0; i < row; i++)
		{
			for (int j = 0; j < col; j++)
			{
				acc += fMat[i][j];
			}
		}
		acc /= row * col;
		return acc;
}

void floatMatrix::setValue(int min, int max)
{
		for (int i = 0; i < row; i++)
		{
			for (int j = 0; j < col; j++)
			{
				fMat[i][j] = ((float(rand()) / float(RAND_MAX))
* (max - min)) + min;
			}
		}
}

void floatMatrix::printMatrix()
{
		std::cout << "\n";
		for (int i = 0; i < row; i++)
		{
			for (int j = 0; j < col; j++)
			{
				std::cout << std::fixed << std::setprecision(1)
<< std::setw(6) << fMat[i][j];
			}
```

```
                std::cout << "\n";
        }
}
```

functions.h

```
#pragma once

#include "parentclass.h"
#include "subclasses.h"

#include <iostream>
#include <string>
#include <ctime>
#include <vector>

void enterSize(int&, int&);
int getAmount();
void getRandMinMax(int&, int&);

bool isNumber(std::string&);

std::vector<intMatrix> generateIntMatrixes(int, int, int, int, int);
intMatrix generateSingleIntMatrix(int, int, int, int);

std::vector<floatMatrix> generateFloatMatrixes(int, int, int, int, int);
floatMatrix generateSingleFloatMatrix(int, int, int, int);

void toPrintIntMatrix(std::vector<intMatrix>);
void toPrintFloatMatrix(std::vector<floatMatrix>);

std::vector<intMatrix> toUpInt(std::vector<intMatrix>);
std::vector<floatMatrix> toDownFloat(std::vector<floatMatrix>);

float findAverage(std::vector<intMatrix>, std::vector<floatMatrix>);
```

functions.cpp

```
#include "functions.h"

void enterSize(int& row, int& col)
{
        std::string rows, cols;
        std::cout << "Enter the number of rows: "; std::cin >> rows;
        while (!isNumber(rows) || std::stoi(rows) < 1) {
                std::cout << "Enter the correct value: ";
                std::cin >> rows;
        }

        std::cout << "Enter the number of cols: "; std::cin >> cols;
        while (!isNumber(cols) || std::stoi(cols) < 1) {
                std::cout << "Enter the correct value: ";
                std::cin >> cols;
        }
```

```
        row = std::stoi(rows);
        col = std::stoi(cols);
}

int getAmount()
{
        int amount = rand() % 4 + 1;
        return amount;
}

void getRandMinMax(int &Min, int &Max)
{
        std::string min, max;
        std::cout << "\nEnter the min number for rand: "; std::cin >> min;
        while (!isNumber(min)) {
                std::cout << "Enter the correct value: ";
                std::cin >> min;
        }

        std::cout << "Enter the max number for rand: "; std::cin >> max;
        while (!isNumber(max) || stoi(max)<stoi(min)) {
                std::cout << "Enter the correct value: ";
                std::cin >> max;
        }

        Min = std::stoi(min);
        Max = std::stoi(max);
}

bool isNumber(std::string& num) {
        int st = 1;
        for (char i : num) {
                if (i == '-' && st == 1) { continue; }
                if (!isdigit(i)) return false;
                st++;
        }
        return true;
}

std::vector<intMatrix> generateIntMatrixes(int row, int col, int amount, int
min, int max)
{
        std::vector<intMatrix> intMatrixes;
        intMatrix matrix(row, col);
        for (int i = 0; i < amount; i++)
        {
                matrix = generateSingleIntMatrix(row, col, min, max);
                intMatrixes.push_back(matrix);
        }
        return intMatrixes;
}

intMatrix generateSingleIntMatrix(int row, int col, int min, int max)
{
        intMatrix matrix(row, col);
        matrix.setValue(min, max);

        return matrix;
```

```
}

std::vector<floatMatrix> generateFloatMatrixes(int row, int col, int amount, int
min, int max)
{
	std::vector<floatMatrix> intMatrixes;
	floatMatrix matrix(row, col);
	for (int i = 0; i < amount; i++)
	{
		matrix = generateSingleFloatMatrix(row, col, min, max);
		intMatrixes.push_back(matrix);
	}
	return intMatrixes;
}


floatMatrix generateSingleFloatMatrix(int row, int col, int min, int max)
{
	floatMatrix matrix(row, col);
	matrix.setValue(min, max);

	return matrix;
}

void toPrintIntMatrix(std::vector<intMatrix> matrixes)
{
	for (int i = 0; i < matrixes.size(); i++)
	{
		std::cout << "\nint matrix #" << i + 1<<": ";
		matrixes[i].printMatrix();
	}
}

void toPrintFloatMatrix(std::vector<floatMatrix> matrixes) {

	for (int i = 0; i < matrixes.size(); i++)
	{
		std::cout << "\nfloat matrix #" << i + 1 << ": ";
		matrixes[i].printMatrix();
	}
}

std::vector<intMatrix> toUpInt(std::vector<intMatrix> matrixes) {

	for (int i = 0; i < matrixes.size(); i++)
	{
		matrixes[i].valueUp();
	}
	return matrixes;
}

std::vector<floatMatrix> toDownFloat(std::vector<floatMatrix> matrixes) {

	for (int i = 0; i < matrixes.size(); i++)
	{
		matrixes[i].valueDown();
	}
	return matrixes;
}
```

```
float findAverage(std::vector<intMatrix> intmatrixes, std::vector<floatMatrix>
floatmatrixes) {

	float min_counter_int = floatmatrixes[0].countAverage();
	float min_counter_float = intmatrixes[0].countAverage();
	float avFl = 0, avInt = 0;
	int findex = 0;
	int index = 0;

	for (int i = 0; i < intmatrixes.size(); i++)
	{
		avFl = floatmatrixes[i].countAverage();
		avInt = intmatrixes[i].countAverage();
		if (avInt < min_counter_int) { min_counter_int = avInt; index =
i; }
		if (avFl < min_counter_float) { min_counter_float = avFl; findex
= i; }
	}

	if (min_counter_float <= min_counter_int) {
		std::cout << "\n\nFloat matrix #" << findex + 1 << " with min
av:\n"; floatmatrixes[findex].printMatrix(); return min_counter_float;}
	if(min_counter_float > min_counter_int) { std::cout << "\n\nInt matrix
#" << index + 1 << " with min av:\n"; intmatrixes[index].printMatrix(); return
min_counter_int; }
}
```

main

```
#include "functions.h"

int main()
{
    srand(time(nullptr));

    int rows, cols, min, max, amount;
    enterSize(rows, cols);
    amount = getAmount();
    getRandMinMax(min, max);

    std::cout << "\n--------Info--------\n";
    std::vector<intMatrix> intMatrixes = generateIntMatrixes(rows, cols, amount,
min, max);
    std::vector<floatMatrix> floatMatrixes = generateFloatMatrixes(rows, cols,
amount, min, max);
    toPrintIntMatrix(intMatrixes);
    toPrintFloatMatrix(floatMatrixes);

    floatMatrixes = toDownFloat(floatMatrixes);
    intMatrixes = toUpInt(intMatrixes);

    std::cout << "\n--------After change--------\n";
    toPrintIntMatrix(intMatrixes);
    toPrintFloatMatrix(floatMatrixes);

    float minAv = findAverage(intMatrixes, floatMatrixes);
```

```
    std::cout << "\nThe minimal average value is " << minAv;

    std::cin.ignore(2);
}
```

Випробування:

```
C:\Users\Admин\source\repos\c++ lab 5 final\Debug\c+
Enter the number of rows: 2
Enter the number of cols: 2

Enter the min number for rand: -10
Enter the max number for rand: 10

--------Info--------

int matrix #1:
   -3    9
   -2   10

float matrix #1:
   4.8  -0.0
   8.3   3.8

--------After change--------

int matrix #1:
    6   18
    7   19

float matrix #1:
  -0.2  -5.0
   3.3  -1.2


Float matrix #1 with min av:

  -0.2  -5.0
   3.3  -1.2

The minimal average value is -0.8
```

Код python

classes.py

```python
from abc import ABC, abstractmethod
import random


class TMatrix(ABC):

    def __init__(self, rows, cols, minRange, maxRange):
        self._rows = rows
        self._cols = cols
        self.M = self.generate_matrix(minRange, maxRange)

    @abstractmethod
    def generate_matrix(self, minRange, maxRange):
        pass

    def get_rows(self):
        return self._rows

    def get_cols(self):
        return self._cols

    def display(self):
        for i in range(self._rows):
            for j in range(self._cols):
                self.M[i][j] = round(self.M[i][j], 2)
        for i in self.M:
            print(*i)
        print()

    def increment(self, num):
        for i in range(self._rows):
            for j in range(self._cols):
                self.M[i][j] = self.M[i][j] + num

    def decrement(self, num):
        for i in range(self._rows):
            for j in range(self._cols):
                self.M[i][j] = self.M[i][j] - num

    def avg(self):
        sum = 0
        average = 0
        for i in range(self._rows):
            for j in range(self._cols):
                sum = sum + self.M[i][j]
        average = sum / (self._rows * self._cols)
        return average


class intMatrix(TMatrix):
    def __init__(self, rows, cols, minRange, maxRange):
        super().__init__(rows, cols, minRange, maxRange)

    def generate_matrix(self, minRange, maxRange):
        return [[random.randint(minRange, maxRange) for i in range(self._cols)]
for j in range(self._rows)]


class floatMatrix(TMatrix):
```

```
    def __init__(self, rows, cols, minRange, maxRange):
        super().__init__(rows, cols, minRange, maxRange)

    def generate_matrix(self, minRange, maxRange):
        return [[random.uniform(minRange, maxRange) for i in range(self._cols)]
for j in range(self._rows)]
```

functions.py

```
from classes import *

def createIntMat(matrixes, r, c ,minR, maxR, am):
    print("Int matrixes:\n")
    for i in range(am):
        matrixes[i] = intMatrix(r, c, minR, maxR)
        print(i + 1, ")")
        matrixes[i].display()

def createFloatMat(matrixes, r, c ,minR, maxR, am):
    print("Float matrixes:\n")
    for i in range(am, 2 * am):
        matrixes[i] = floatMatrix(r, c, minR, maxR)
        print(i + 1, ")")
        matrixes[i].display()

def intUpValue(matrixes, am):
    print("Int matrices:\n")
    for i in range(am):
        matrixes[i].increment(9)
        print(i + 1, ")")
        matrixes[i].display()

def intDownValue(matrixes, am):
    print("Float matrices:\n")
    for i in range(am, 2 * am):
        matrixes[i].decrement(5)
        print(i + 1, ")")
        matrixes[i].display()

def findAverage(matrixes, am):
   minAv = matrixes[0].avg()
   index = 1

   for i in range(2 * am):
       if matrixes[i].avg() < minAv:
           minAv = matrixes[i].avg()
           index = i
   print("Matrix with the min av #", index + 1)
```

```
    matrixes[index].display()
    return (minAv, index + 1);

def enterMinMax():
    minR = int(input("\nEnter the min number for rand: "))
    maxR = int(input("Enter the max number for rand: "))
    while(maxR < minR or maxR == minR):
        maxR = int(input("Enter correct number: "))
    return (minR, maxR);

def enterRowCol():
    row = int(input("Enter number of rows: "))
    while(row < 0):
        row = int(input("Enter correct number: "))

    col = int(input("Enter number of cols: "))
    while(col < 0):
        row = int(input("Enter correct number: "))

    return (row, col);
```

main

```
from classes import *
from functions import *

def main():

    amount = random.randint(1, 4)
    r, c = enterRowCol()
    minR, maxR = enterMinMax()
    print()

    matrixes = [None for i in range(2 * amount)]

    createIntMat(matrixes, r, c, minR, maxR, amount)
    createFloatMat(matrixes, r, c ,minR, maxR, amount)

    print("Changed matrices:\n")

    intUpValue(matrixes, amount)
    intDownValue(matrixes, amount)

    min, index = findAverage(matrixes, amount)
    print("The smallest average:", round(min, 2), "has matrix",
index, "\n")

if __name__ == '__main__':
    main()
```

Випробування:

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python
Enter number of rows: 2
Enter number of cols: 2

Enter the min number for rand: -10
Enter the max number for rand: 10

Int matrixes:

1 )
-8 -4
8 3

2 )
2 7
-9 6

3 )
3 -6
10 -3

4 )
-3 -5
-9 0

Float matrixes:

5 )
8.89 -5.27
-4.69 3.06

6 )
2.61 1.29
7.61 -5.15

7 )
4.07 -9.16
0.66 -8.53

8 )
2.33 -7.26
-8.14 -3.04

Changed matrices:

Int matrices:

1 )
1 5
17 12
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_6

1 )
1 5
17 12

2 )
11 16
0 15

3 )
12 3
19 6

4 )
6 4
0 9

Float matrices:

5 )
3.89 -10.27
-9.69 -1.94

6 )
-2.39 -3.71
2.61 -10.15

7 )
-0.93 -14.16
-4.34 -13.53

8 )
-2.67 -12.26
-13.14 -8.04

Matrix with the min av # 8
-2.67 -12.26
-13.14 -8.04

The smallest average: -9.03 has matrix 8
```

Висновки:

Під час виконання п’ятої лабораторної роботи я навчився механізмам створення та використання класів об’єктів. Також набув навичок успадкування, поліморфізму та створення абстрактного класу.