**Міністерство освіти і науки України**

**Національний університет “Львівська політехніка”**

**Інститут комп’ютерних наук та інформаційних технологій**

***Кафедра ПЗ***

**Звіт**

до лабораторної роботи №10

на тему «Шаблони класів»

з дисципліни “Об’єктно-орієнтоване програмування”

**Виконав:**

студент групи ПЗ-11

Ясногородський Н.В

**Перевірив**:

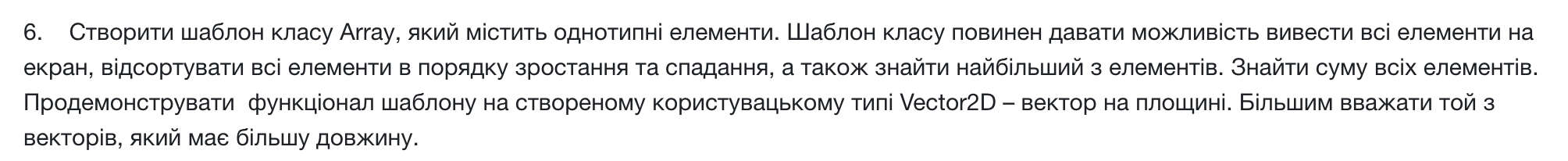
 доц. Коротєєва Т.О.

Львів

**2022**

##### **Тема.** Шаблони класів. **Мета.** Навчитись створювати шаблони класу та екземпляри шаблонів.

##### **Завдання для лабораторної роботи:**

Створити шаблон класу та продемонструвати його роботу за індивідуальним варіантом. Оформити звіт до лабораторної роботи. Звіт має містити варіант завдання, код розробленої програми, результати роботи програми (скріншоти), висновок.

**Теоретичні відомості:**

Прикладом шаблону в реальному житті є трафарет — об’єкт, в якому прорізаний малюнок/візерунок/символ. Якщо прикласти трафарет до іншого об’єкту і розпорошити фарбу, то отримаємо цей же малюнок фарбою, якщо засипати піском, то отримаємо інший варіант того самого малюнка і т.п. Ми зможемо зробити десятки таких малюнків різних кольорів і різних основ! При цьому нам потрібен лише один трафарет.

У мові C++ шаблони функцій — це функції, які служать взірцем для створення інших подібних функцій. Головна ідея — створення функцій без вказівки точного типу(ів) деяких або всіх змінних. Для цього ми визначаємо функцію, вказуючи тип параметра шаблону, який використовується замість будь-якого типу даних. Після того, як ми створили функцію з типом параметра шаблону, ми фактично створили «трафарет функції».

При виклику шаблону функції, компілятор використовує «трафарет» в якості зразка функції, замінюючи тип параметра шаблону на фактичний тип змінних, переданих у функцію!

Розробка шаблонів функцій дозволяє створювати узагальнені за алгоритмом функції, які можуть працювати для різних типів даних (як для вбудованих, та і для користувацьких).

Для оголошення шаблону функції використовується ключове слово template , далі в трикутних дужках записується тип параметру шаблону <typename T> або <class T> . У мові C++ прийнято називати типи параметрів шаблонів великою літерою T, але можна використовувати будь-який ідентифікатор.

Якщо потрібно кілька типів параметрів шаблону, то вони розділяються комами: template <typename T1, typename T2>

Коли компілятор зустрічає виклик шаблону функції, він копіює шаблон функції і замінює типи параметрів шаблону функції фактичними (переданими) типами даних. Функція з фактичними типами даних називається екземпляром шаблону функції (або *«об’єктом шаблону функції»*).

Якщо створити шаблон функції, але не викликати його, то екземпляри цього шаблону створені не будуть.

Шаблони функцій працюють як з вбудованими типами даних (char, int, double тощо), так і з класами. Екземпляр шаблону компілюється як звичайна функція. Будь-які оператори або виклики інших функцій, які присутні в шаблоні функції, повинні бути визначені для роботи з фактичними типами даних.

Переваги: Шаблони функцій економлять багато часу, тому що шаблон ми пишемо тільки один раз, а використовувати можемо з різними типами даних. Шаблони функцій набагато спрощують подальшу підтримку коду, і вони безпечніші, тому що немає необхідності виконувати вручну перевантаження функції, копіюючи код і змінюючи лише типи даних, коли потрібна підтримка нового типу даних.

У шаблонів функцій є кілька недоліків:

По-перше, деякі старі компілятори можуть не підтримувати шаблони функцій або підтримувати, але з обмеженнями. Однак зараз це вже не така проблема, як раніше.

По-друге, шаблони функцій часто видають божевільні повідомлення про помилки, які набагато складніше розшифрувати, ніж помилки звичайних функцій.

По-третє, шаблони функцій можуть збільшити час компіляції і розмір коду, тому що один шаблон може бути «реалізований» і перекомпільований в декількох файлах.

Дані недоліки досить незначні в порівнянні з потужністю і гнучкістю шаблонів функцій!

**Результат:**

array.cpp

#ifndef \_\_ARRAY\_H

#define \_\_ARRAY\_H

#include <algorithm>

#include <cmath>

#include <numeric>

#include <random>

#include <string>

#include <vector>

template <class T>

class Array {

public:

std::vector<T> values;

void sort\_asc() {

sort\_desc();

std::reverse(values.begin(), values.end());

}

void sort\_desc() { std::sort(values.begin(), values.end()); }

T sum() { return std::reduce(values.begin(), values.end()); }

T find\_max() { return \*std::max\_element(values.begin(), values.end()); }

void push(T const &v) { values.push\_back(v); }

};

class Vector2D {

public:

double x, y;

Vector2D() {

std::random\_device rd;

std::default\_random\_engine eng(rd());

std::uniform\_real\_distribution<double> distr(0, 100);

x = distr(eng);

y = distr(eng);

}

Vector2D operator+(const Vector2D &v) {

Vector2D temp;

temp.x = x + v.x;

temp.y = y + v.y;

return temp;

}

double calc\_length() const { return std::sqrt(x \* x + y \* y); }

bool operator<(const Vector2D &v) const {

return calc\_length() < v.calc\_length();

}

bool operator>(const Vector2D &v) const { return \*this > v; }

std::string to\_string() const {

return "x=" + std::to\_string(x) + " y=" + std::to\_string(y) +

" length=" + std::to\_string(calc\_length()) + "\n";

}

};

#endif

widget.cpp

#include "widget.h"

#include <QFile>

#include <QGridLayout>

#include <QTextStream>

#include <random>

void Widget::on\_output() {

array.values.clear();

for (auto i = 0; i < 10; i++) array.push(\*new Vector2D());

this->results\_output->setMarkdown(

QString("### Array of Vector2D:\n\n"

"\* Biggest Vector2D x=%1 y=%2\n"

"\* Vector2D sum (resulting vector length)=%3\n")

.arg(QString::number(array.find\_max().x))

.arg(QString::number(array.find\_max().y))

.arg(QString::number(array.sum().calc\_length())));

print\_array();

}

void Widget::on\_sort\_asc() {

array.sort\_asc();

print\_array();

}

void Widget::on\_sort\_desc() {

array.sort\_desc();

print\_array();

}

void Widget::print\_array() {

std::string array\_as\_string;

for (auto v : array.values) array\_as\_string += v.to\_string();

this->array\_output->setText(QString::fromStdString(array\_as\_string));

}

Widget::Widget(QWidget \*parent) : QWidget(parent) {

auto \*main\_layout = new QGridLayout;

this->output\_btn = new QPushButton("Start");

this->sort\_asc\_btn = new QPushButton("Sort asc");

this->sort\_desc\_btn = new QPushButton("Sort desc");

this->array\_output = new QTextEdit;

this->array\_output->setReadOnly(true);

this->results\_output = new QTextEdit;

this->results\_output->setReadOnly(true);

main\_layout->addWidget(this->array\_output, 0, 0);

main\_layout->addWidget(this->output\_btn, 0, 1);

main\_layout->addWidget(this->results\_output, 0, 2);

main\_layout->addWidget(this->sort\_asc\_btn, 1, 0);

main\_layout->addWidget(this->sort\_desc\_btn, 1, 2);

connect(this->output\_btn, &QPushButton::released, this, &Widget::on\_output);

connect(this->sort\_asc\_btn, &QPushButton::released, this,

&Widget::on\_sort\_asc);

connect(this->sort\_desc\_btn, &QPushButton::released, this,

&Widget::on\_sort\_desc);

setLayout(main\_layout);

}

main.cpp

#include "widget.h"

#include <QApplication>

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

{

QApplication a(*argc*, *argv*);

Widget w;

w.show();

*return* a.exec();

}

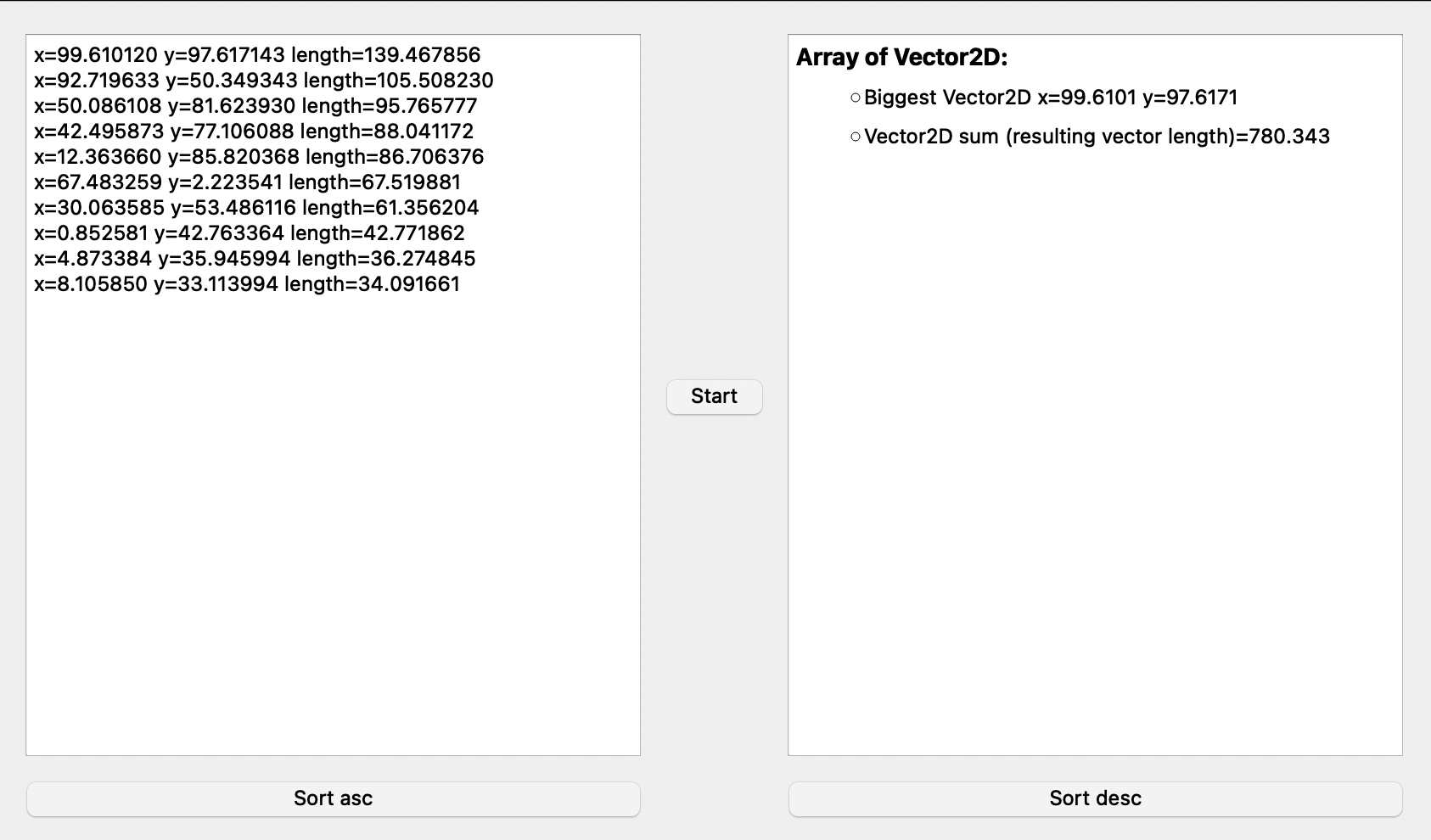


Рис.1. Виконання программи

**Висновок:**

##### У ході лабораторної роботи №10 я навчився створювати шаблони класу та екземпляри їх екземпляри на прикладі шаблону класу Array, що використовує вбудовані типи та користувацький Vector2D.