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.

6. Створити шаблон класу Array, який містить однотипні елементи. Шаблон класу повинен давати можливість вивести всі елементи на екран, відсортувати всі елементи в порядку зростання та спадання, а також знайти найбільший з елементів. Знайти суму всіх елементів. Продемонструвати функціонал шаблону на створеному користувацькому типі Vector2D – вектор на площині. Більшим вважати той з векторів, який має більшу довжину.

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.